Workload Modeling and Prediction for Workflow Scheduling in Dynamic Grid Environments

Dissertation
zur Erlangung des akademischen Grades
Doktorin der Naturwissenschaften
(Dr. rer. nat.)

der Fakultät für
Mathematik und Informatik
der FernUniversität
in Hagen

Verfasserin: Attiqa Rehman, Master of Science in
Computer Science, MS (CS)
Erstgutachter: Prof. Dr. Wolfram Schiffmann
Zweitgutachter: Prof. Dr. Jörg Keller
Vorgelegt am: 30.08.2013
Mündliche Prüfung: 22.01.2014
This research work is dedicated to my parents, especially to my late father
Acknowledgements

First and foremost, I would like to thank and praise Allah almighty, the only One God, for the opportunities and support I was given along the way. This work would not have been possible otherwise.

This work is carried out during my stay at the research group Rechnerarchitektur, FernUniversität Hagen for acquiring my doctoral degree under the grant of Higher Education Commission (HEC) of Pakistan in collaboration with German Academic Exchange Program (DAAD) Germany. Besides, every doctoral thesis lies years of hard work. Looking back, I can see many people who have given me support and encouragement during this period. It is, therefore, my pleasure to thank all of them.

I am grateful for the great supervision and tremendous support, guidance and encouragement from my supervisor, Prof. Dr. Wolfram Schiffmann. He gave me time from his busy schedule and was always available whenever, I required.

I am also thankful to my second supervisor, Prof. Dr. Jörg Keller. He always managed to attend the meetings arranged for this research work. His kind words and support encouraged me a lot. I am also very grateful to the members of the research group Rechnerarchitektur and specially the Dr. Udo Höning for helpful discussions.

I would like to thank my family members (mother and sisters) for their motivational support throughout this work. Last but not the least; I am grateful for support and patience of my dear husband Syed Sajid Hussain.
Abstract

Many scientific applications utilize grid environments for processing their workloads, which consist of workflows. Grid environments are highly dynamic because the grid resources belong to different administrative domains, and have site-specific scheduling and resource management policies. Workflow scheduling in dynamic grid environments involves mapping the tasks of a workflow to the grid resources with an aim of optimizing certain objectives, e.g., the makespan of the workflow, the utilization of the grid resources, etc.

Designing and evaluating new workflow scheduling algorithms requires comprehensive workload modeling, which is missing in contemporary research. Moreover, conventional grid workflow scheduling algorithms are based on some unrealistic assumptions, which do not necessarily hold in a dynamic grid environment. E.g., some approaches assume that at a time, only one workflow executes in a grid environment, the computation and the communication time of the workflows are known in advance, and the grid resources are always fully available. A workflow-based workload model representing the characteristics of workflows (e.g., workflows’ type, structure, and their requirements) is needed for evaluating the grid workflow scheduling algorithms. However, such a workload model is missing in contemporary research.

This thesis presents a workflow-based workload model using a workflow test bench suite and integrates this workload model in a simulated grid environment. Moreover, this thesis presents the design and development of a decentralized grid workflow scheduler, which is based on predicting the Local Users’ (LUs) load at the grid resources in a dynamic grid environment. The new scheduling algorithms for scheduling the tasks of a workflow to the grid resources in the presence of their LUs’ load are proposed and implemented in the developed grid workflow scheduler. The proposed scheduling algorithms are named as the Local Users Load Prediction-based (LULP)-Max-Min, the LULP-Min-Min and the LULP-Sufferage. Real grid workload traces are used to represent the LUs’ load in the evaluation of the LULP-based scheduling algorithms. The proposed algorithms are compared with the Heterogeneous Earliest Finish Time First (HEFT) scheduling algorithm in terms of the makespan, the data movement time, and the resource effective utilization. The results show that the proposed algorithms achieved low makespan and data movement time. The
A decentralized grid workflow scheduler is extended for predicting the queue wait time of the grid resources. A workflow scheduling algorithm called Queue Wait time Prediction-based Grid Workflow Scheduling algorithm (QWP-GWS) is proposed and implemented in the grid workflow scheduler. The QWP-GWS aims to optimize multiple objectives, i.e., minimizing the makespan while maximizing the resource effective utilization. The results of the QWP-GWS algorithm are compared with the Min-Min, the Max-Min, the Sufferage and the HEFT scheduling algorithms in terms of the makespan, the queue wait time, the resource effective utilization, and the data movement time. The results show that the QWP-GWS algorithm achieved lower makespan without overwhelming the resource effective utilization and the data movement time. The LULP-Max-Min, the LULP-Min-Min, the LULP-Sufferage, and the QWP-GWS algorithms integrate the data saving algorithms proposed in this thesis in order to achieve low data movement time, which results in low makespan.
# Table of Contents

List of Figures .................................................................................................................. xi
List of Tables ..................................................................................................................... xiii
List of Algorithms ............................................................................................................. xv

1. Introduction ..................................................................................................................... 1
   1.1. Background .............................................................................................................. 1
       1.1.1. Grid environment ......................................................................................... 1
       1.1.2. Workflows .................................................................................................. 3
       1.1.3. Grid workflow scheduling ....................................................................... 6
   1.2. Workflow Scheduling in a Dynamic Grid Environment ........................................ 8
       1.2.1. Workflow-based workload modeling ......................................................... 8
       1.2.2. Dynamicity of the grid resources ................................................................. 9
       1.2.3. Multiple workflows .................................................................................... 9
       1.2.4. Queue wait time ......................................................................................... 10
       1.2.5. Data and task co-scheduling ...................................................................... 10
       1.2.6. Resource utilization .................................................................................... 11
   1.3. Research Problems ............................................................................................... 11
       1.3.1. Workflow-based workload modeling ......................................................... 11
       1.3.2. Multiple workflows .................................................................................... 11
       1.3.3. Local Users’ load ....................................................................................... 11
       1.3.4. Data movement time .................................................................................. 11
       1.3.5. Queue wait time ......................................................................................... 11
       1.3.6. Workflow scheduling algorithms ............................................................... 12
   1.4. Proposed Solution ................................................................................................. 12
   1.5. Contributions ....................................................................................................... 14
   1.6. Organization of the Thesis ................................................................................... 15

2. Problem Analysis .......................................................................................................... 18
   2.1. Grid Workflow Scheduling ............................................................................... 18
       2.1.1. Types of the grid ....................................................................................... 18
       2.1.2. Grid resources’ characteristics ................................................................. 19
       2.1.3. Structure of a workflow ............................................................................ 22
       2.1.4. Grid workflow scheduling problem ......................................................... 23
       2.1.5. Grid workflow scheduler’s architecture ................................................... 24
       2.1.6. Grid workflow scheduling algorithms ...................................................... 25
       2.1.7. Information required to a grid workflow scheduling algorithm ............... 26
       2.1.8. Grid workflow scheduling objectives ....................................................... 28
       2.1.9. Data movement time ................................................................................ 32
       2.1.10. Impact of the scheduling decisions on the scheduling objective ............ 33
   2.2. Requirement Analysis ........................................................................................... 35
       2.2.1. Requirements of workflow-based workload modeling ............................... 35
       2.2.2. Requirements for simulating a grid environment ........................................ 38
       2.2.3. Requirement for handling multiple workflows ........................................... 39
       2.2.4. Requirement for handling LUs’ load ......................................................... 40
# Table of Contents

2.2.5. Requirements for handling queue wait time ................................................. 40  
2.2.6. Requirements of the grid workflow scheduling algorithms .......................... 41  
2.2.7. Requirements for handling data movement time ......................................... 42  

2.3. Summary of Requirements ............................................................................. 43  

3. State of the Art ................................................................................................... 44  
  3.1. Workflow Scheduling in Grid Environment .................................................... 44  
    3.1.1. Workflow-based workload modeling ....................................................... 45  
    3.1.2. Workflow scheduling in dynamic grid environment .............................. 46  
  3.2. Summary ......................................................................................................... 51  

4. Grid Simulation Tools and Workflow Test Bench Suites ................................. 52  
  4.1. Essential Grid Abstractions for Grid workflow Scheduling ............................. 52  
  4.2. Requirements for the Grid Simulators ............................................................. 53  
  4.3. A Survey of Grid Simulators .......................................................................... 55  
    4.3.1. Bricks ...................................................................................................... 55  
    4.3.2. GangSim .................................................................................................. 56  
    4.3.3. OptorSim ............................................................................................... 57  
    4.3.4. SimGrid ................................................................................................... 58  
    4.3.5. GridSim ................................................................................................... 59  
    4.3.6. Alea ......................................................................................................... 61  
    4.3.7. Grid scheduling simulator (GSSIM) .......................................................... 62  
    4.3.8. Grid economy simulator (GES) ................................................................. 63  
  4.4. Authenticity and Scalability of the Grid Simulators .......................................... 63  
    4.4.1. Authenticity ............................................................................................. 64  
    4.4.2. Scalability ............................................................................................... 65  
  4.5. Selection of a Simulator for Grid Workflow Scheduling ................................. 66  
  4.6. Workflow Test Bench Suite Selection ............................................................. 68  
    4.6.1. Workload modeling .................................................................................. 68  
    4.6.2. Workflow composition ............................................................................. 68  
    4.6.3. Workflow test bench suites ..................................................................... 69  
  4.7. Summary ......................................................................................................... 70  

5. Workflow-based Workload Modeling ................................................................ 71  
  5.1. HC-test Bench .................................................................................................. 72  
  5.2. Workload Modeling ......................................................................................... 75  
    5.2.1. Workload traces from real grid environments ......................................... 76  
    5.2.2. Mathematical models .............................................................................. 77  
  5.3. Workflow-based Workload Modeling ............................................................. 78  
    5.3.1. Characteristics of a workflow-based workload ......................................... 78  
    5.3.2. Modeling the arrival time of the workflows ............................................. 82  
    5.3.3. Modeling the static characteristics ........................................................... 89  
    5.3.4. Modeling the makespan of the workflows ............................................... 92  
    5.3.5. Summary of workflow-based workload modeling .................................... 97  
  5.4. Generating the Workflow-based Workload Model ......................................... 98  
  5.5. GridSim ........................................................................................................... 102  
  5.6. Layered Architecture of a Simulated Grid Environment ............................... 106
8.4. Grid Resource’s Queue Model Architecture ................................................................. 202
  8.4.1. Variables analysis ........................................................................................................... 204
  8.4.2. Rules for grid resource queue ....................................................................................... 205
  8.4.3. Assumptions .................................................................................................................. 206
  8.5.1. Erlang B formula for grid resource .................................................................................. 208
  8.5.2. Erlang C Formula for grid resource .................................................................................. 212
  8.5.3. Grid resource’s mean queue length ................................................................................ 213
  8.5.4. Grid resource’s mean number of tasks in execution ....................................................... 214
  8.5.5. Grid resource’s mean number of tasks .......................................................................... 215
  8.5.6. Variance of number of tasks in the grid resource’s queue ............................................ 215
  8.5.7. Variance of number of tasks in service at a grid resource ............................................ 216
  8.5.8. Variance of number of tasks at a grid resource ............................................................. 218
  8.5.9. Expected response time of a task submitted to a grid resource ..................................... 219
  8.5.10. Expected queue wait time of a task submitted to the grid resource ................................ 219
8.6. Realization of Queue Wait Time Prediction ........................................................................ 219
8.7. Important models for queue wait time scheduling ............................................................. 222
  8.7.1. Workflow model ........................................................................................................... 223
  8.7.2. Resource model ............................................................................................................ 223
  8.7.3. Average computational and communication weights ...................................................... 224
  8.7.4. Workflow’s tasks rank .................................................................................................. 225
  8.7.5. Critical paths ............................................................................................................... 226
8.8. Scheduling and Data Saving Algorithms ............................................................................ 232
  8.8.1. Queue wait time prediction-based grid workflow scheduling algorithm (QWP-GWS) .... 233
  8.8.2. Receiving a completed task of a workflow .................................................................... 238
  8.8.3. Data files saving algorithm ........................................................................................... 239
  8.8.4. An example scenario ..................................................................................................... 241
8.9. Experiment Design and result Evaluation ............................................................................ 245
  8.9.1. Algorithms to evaluate .................................................................................................. 245
  8.9.2. Workload generation ..................................................................................................... 245
  8.9.3. Resource model ............................................................................................................ 246
  8.9.4. Performance metrics ..................................................................................................... 246
  8.9.5. Results ........................................................................................................................... 248
8.10. Summary .......................................................................................................................... 259
9. Conclusions and Future Work .............................................................................................. 260
  9.1. Summary ........................................................................................................................... 260
  9.2. Contributions and Comparison with State of Art ............................................................. 261
  9.3. Outlook ............................................................................................................................. 263

Bibliography .......................................................................................................................... 265

Appendix A  Configuration Formats for the Workflow-based Workload Model 272
  A.1 Format for Specifying the Workflow-based Workload Configurations ................................ 272
  A.2 Format for Specifying the Grid Resources’ Processing Capabilities .................................... 273
  A.3 Format for Specifying the Grid Resources Links’ Bandwidth and Latency .......................... 274
List of Figures

Figure 1-1: A typical grid environment ................................................................. 3
Figure 1-2: An example workflow ............................................................................ 4
Figure 1-3: Three phases of scheduling .................................................................. 6
Figure 1-4: Overview of prediction-based grid workflow scheduling ....................... 13
Figure 2-1: Types of the grid ................................................................................. 19
Figure 2-2: Resource characteristics ....................................................................... 20
Figure 2-3: Structural components of a workflow ..................................................... 22
Figure 2-4: Grid workflow scheduler’s architecture ............................................... 24
Figure 2-5: Dynamism of the grid workflow scheduling algorithm .............................. 25
Figure 2-6: Information required to a grid workflow scheduling algorithm .............. 27
Figure 2-7: Classification of the scheduling objectives ............................................. 29
Figure 2-8: Classification of the scheduling objectives ............................................. 30
Figure 2-9: Data movement methods ....................................................................... 32
Figure 3-1: Key area of research ............................................................................ 44
Figure 5-1: Workflow-based workload model in a simulated grid environment .......... 71
Figure 5-2: Structure of the HC-test bench adapted from [Hönig and Schiffmann, 2004] 72
Figure 5-3: A workflow in the ETG format ............................................................. 74
Figure 5-4: Static characteristics of a workflow-based workload .............................. 80
Figure 5-5: Dynamic characteristics of workflow-based workload ........................... 81
Figure 5-6: Point process adapted from [Li, 2008] .................................................. 83
Figure 5-7: Non-homogeneous Poisson process construction .................................... 86
Figure 5-8: Conceptual overview of the workflow-based workload model ............... 98
Figure 5-9: Schema of workflow-based workload model ........................................ 102
Figure 5-10: Layered architecture of the GridSim [Sulistio et al., 2008] .................. 103
Figure 5-11: Layered architecture of the simulated grid environment ....................... 107
Figure 5-12: Class diagram of the workflow-based application model for GridSim .... 108
Figure 5-13: Workflow classes’ percentage in Workflow-based workload model ...... 111
Figure 5-14: CDF of number of tasks in the workflows of workload model ............... 112
Figure 5-15: CDF of number of Edges(dependencies) in the workflows of workload model 113
Figure 5-16: Workflow arrival rate of an hourly interval ........................................ 114
Figure 6-1: A general prediction process ............................................................... 117
Figure 6-2: Prediction-based grid scheduling process ............................................. 118
Figure 6-3: Conceptual view of decentralized grid workflow scheduler .................... 122
Figure 6-4: Abstract view of prediction-based grid workflow scheduler .................... 125
Figure 6-5: Class diagram of prediction classes .................................................... 130
Figure 6-6: Data movement related classes ............................................................ 131
Figure 6-7: Schedule generation related classes ...................................................... 136
Figure 6-8: Dispatcher class ................................................................................ 139
Figure 6-9: Result gatherer related classes ............................................................ 140
Figure 6-10: Grid workflow scheduler class ............................................................ 141
Figure 7-1: Conceptual view of local user load prediction-based scheduling ............ 148
Figure 7-2: Workflow execution process .................................................................. 178
Figure 7-3: A workflow .......................................................................................... 180
Figure 7-4: Makespan vs. total number of concurrent workflows under different mean arrival rate per Interval .......................................................... 189
Figure 7-5: Makespan difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................ 190
Figure 7-6: Data movement time vs. total number of concurrent workflows under different mean arrival rate per interval ................................................................. 191
Figure 7-7: Data movement time difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per interval ................................................................. 192
Figure 7-8: Resource effective Utilization (%) vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 193
Figure 8-1: Queue model .................................................................................. 200
Figure 8-2: Poisson split .................................................................................. 202
Figure 8-3: Poisson merge ............................................................................... 203
Figure 8-4: Resource R, queue model architecture ............................................. 203
Figure 8-5: State transition diagram of resource R, with m processors ................ 208
Figure 8-6: Addition of dummy entry and exit tasks in a workflow ...................... 226
Figure 8-7: A workflow structure .................................................................... 229
Figure 8-8: All critical paths in a workflow ......................................................... 231
Figure 8-9: A matching approach .................................................................... 235
Figure 8-10: Makespan vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 252
Figure 8-11: Makespan difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per Interval ................................................................. 253
Figure 8-12: Data movement time vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 254
Figure 8-13: Data Movement time difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 255
Figure 8-14: Queue wait time vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 256
Figure 8-15: Computation to data movement time ratio vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 257
Figure 8-16: Resource effective utilization (%) vs. total number of concurrent workflows under different mean arrival rate per Interval ........................................................................ 258
Figure A-1: Format for specifying the Workflow-based workload model configurations .... 272
Figure A-2: Format for specifying the grid resources’ processing capabilities ........ 273
Figure A-3: Format for specifying the grid resources’ links bandwidths ............... 274
Figure A-4: Format for specifying the grid resources’ links latencies ................. 274
List of Tables

Table 3-1: Workload modeling requirements supports by researchers (R1-R4) ........................................ 45
Table 3-2: Requirements supported by previous researches (R6-R12) .............................................................. 50
Table 4-1: Evolution of GridSim from version (1.0-5.0) [GridSim, 2012] ........................................................... 59
Table 4-2: Simulators characteristic summary (A) ................................................................................................ 66
Table 4-3: Simulators characteristic summary (B) .............................................................................................. 67
Table 4-4: Simulators against identified requirements (SR1-SR8) ................................................................. 68
Table 4-5: Test benches characteristics ............................................................................................................. 69
Table 5-1: List of notations for workflow-based workload model ....................................................................... 97
Table 5-2: Percentage of different characteristics' workflows ........................................................................ 113
Table 5-3: Lower bound of computation time, data movement time and makespan ........................................... 115
Table 5-4: Upper bound of computation, communication time and makespan ................................................. 115
Table 5-5: Mean value of computation, communication time and makespan .................................................. 116
Table 7-1: List of notations .................................................................................................................................. 152
Table 7-2: Example resource configurations .................................................................................................... 179
Table 7-3: Bandwidth of the links among grid resources .................................................................................. 179
Table 7-4: Latencies of the links among grid resources .................................................................................... 179
Table 7-5: LUs' load at the grid resource ............................................................................................................ 180
Table 7-6: Completion time of task t^1 .............................................................................................................. 180
Table 7-7: Data Location .................................................................................................................................... 181
Table 7-8: Data Transfer Time of t^2 and t^3 .................................................................................................... 181
Table 7-9: Completion time of t^2 and t^3 .......................................................................................................... 181
Table 7-10: Data location of files of task t^2 ..................................................................................................... 182
Table 7-11: Completion time of task t^4 ........................................................................................................... 182
Table 7-12: Completion time of task t^5 ............................................................................................................ 182
Table 7-13: Resource configurations ................................................................................................................. 184
Table 7-14: Prediction accuracy (%) of LULP-Min-Min .................................................................................... 194
Table 7-15: Prediction accuracy (%) of LULP-Sufferage ................................................................................ 194
Table 7-16: Prediction accuracy (%) of LULP-Max-Min .................................................................................. 194
Table 8-1: List of notations for queue wait time at a grid resource ................................................................. 207
Table 8-2: List of notations queue wait time prediction-based scheduling algorithm .................................... 222
Table 8-3: Example resource configurations .................................................................................................... 228
Table 8-4: Unit data transfer time ..................................................................................................................... 228
Table 8-5: Latencies among resources ............................................................................................................. 228
Table 8-6: Mean computation time .................................................................................................................. 229
Table 8-7: Mean communication time ............................................................................................................. 230
Table 8-8: Ranks of tasks ................................................................................................................................. 230
Table 8-9: Grid resources' queue wait time ....................................................................................................... 241
Table 8-10: Resources' ECP_t(R_i) values ......................................................................................................... 241
Table 8-11: Workflow task assignment step 1 ................................................................................................ 241
Table 8-12: Locations of data files produced by t^1 ......................................................................................... 242
Table 8-13: Workflow task assignment step 2 ................................................................................................ 242
Table 8-14: Locations of data files produced by t^2, t^3 and t^4 ..................................................................... 243
Table 8-15: Workflow tasks assignment step 3 .............................................................................................. 243
Table 8-16: Locations of data files produced by t^5, t^6 and t^7 ................................................................... 243
Table 8-17: Workflow tasks assignment step 4 .............................................................................................. 243
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-18</td>
<td>Locations of data files produced by $t_2$ and $t_3$</td>
<td>244</td>
</tr>
<tr>
<td>8-19</td>
<td>Workflow tasks assignment step 5</td>
<td>244</td>
</tr>
<tr>
<td>8-20</td>
<td>Resource configurations</td>
<td>246</td>
</tr>
<tr>
<td>8-21</td>
<td>Prediction accuracy (%) of QWP-GWS</td>
<td>251</td>
</tr>
</tbody>
</table>
# List of Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1</td>
<td>Poisson simulation_1</td>
<td>87</td>
</tr>
<tr>
<td>5-2</td>
<td>Poisson simulation_2</td>
<td>88</td>
</tr>
<tr>
<td>6-1</td>
<td>Receiving a task of a workflow method</td>
<td>143</td>
</tr>
<tr>
<td>6-2</td>
<td>Grid Workflow Scheduler body method</td>
<td>144</td>
</tr>
<tr>
<td>7-1</td>
<td>Poisson variant generation</td>
<td>157</td>
</tr>
<tr>
<td>7-2</td>
<td>Produced files saving algorithm for local user prediction-based scheduler</td>
<td>169</td>
</tr>
<tr>
<td>7-3</td>
<td>Save the file on the resource specified by resource ID</td>
<td>169</td>
</tr>
<tr>
<td>7-4</td>
<td>Start procedure LULP scheduling algorithms</td>
<td>173</td>
</tr>
<tr>
<td>7-5</td>
<td>LULP-Sufferage scheduling algorithm</td>
<td>174</td>
</tr>
<tr>
<td>7-6</td>
<td>LULP-Min-Min Scheduling algorithm</td>
<td>175</td>
</tr>
<tr>
<td>7-7</td>
<td>LULP-Max-Min scheduling algorithm</td>
<td>176</td>
</tr>
<tr>
<td>7-8</td>
<td>Receiving a completed task of a workflow</td>
<td>177</td>
</tr>
<tr>
<td>8-1</td>
<td>Find Critical Path</td>
<td>227</td>
</tr>
<tr>
<td>8-2</td>
<td>Queue wait time aware workflow scheduling algorithm</td>
<td>237</td>
</tr>
<tr>
<td>8-3</td>
<td>Schedule a ready List</td>
<td>238</td>
</tr>
<tr>
<td>8-4</td>
<td>Receive a completed task of a workflow</td>
<td>239</td>
</tr>
<tr>
<td>8-5</td>
<td>Produced files saving algorithm for queue wait time prediction-based scheduler</td>
<td>240</td>
</tr>
<tr>
<td>8-6</td>
<td>Save the file on the resource specified by resource ID</td>
<td>241</td>
</tr>
</tbody>
</table>
1. Introduction

This chapter provides a high-level description of the work presented in this thesis. Firstly, the key concepts like grid computing, workflows, and grid workflow scheduling are introduced. Section 1.2 describes grid workflow scheduling in a dynamic grid environment, and identifies the research issues considered in this thesis. This section concludes by articulating concrete research questions addressed in this thesis. Section 1.4 describes the proposed solutions, and section 1.5 lists the core contributions of this thesis. Section 1.6 concludes this chapter by presenting an overview of the following chapters.

1.1. Background

To provide computing as a utility at every doorstep (like other utilities e.g., electricity, water, etc.) was fantasized by Leonard Kleinrock in 1969, as reported by [Kleinrock, 2003]:

“As of now, computer networks are still in their infancy, but as they grow up and become sophisticated, we will probably see the spread of, ‘computer utilities’, which, like present electric and telephone utilities, will service individual homes and offices across the country.” (p. 2)

Continuous technological advancements and novel application requirements have led the dream of Kleinrock to become a reality. In the past decades, there has been a paradigm shift from desktop-based personal computing to parallel and distributed computing and grid computing.

1.1.1. Grid environment

The ever increasing computational power and high speed networks are primary reasons of the emergence of a computational grid. The aim of a computational grid is to form a Virtual Organization (VO) of the geographically dispersed computational, data, network resources, and users with an aim to efficiently utilize the resources [Foster et al., 2001]. Such a VO is dynamically formed by resources from different administrative domains having site-specific policies (i.e., the scheduling algorithm, the load balancing algorithm, etc.). Therefore, it is a flexible and coordinated resource sharing among the various users [Foster et al., 2002].
The grid can be categorized according to the provided services (e.g., data grid, computational grid, media grid, and bioinformatics grid etc.), the scope of resource sharing (i.e., dedicated to the grid, and non-dedicated to the grid) and the underlying network (i.e., organizational grid, enterprise grid, and worldwide grid). Many fields of science (like astronomy, bioinformatics, high energy physics, climate modeling, earthquake engineering, and medical sciences etc.) have utilized the grid environments for their applications [Chervenak et al., 1999]. These scientific applications usually consist of multiple tasks, where each task is composed of a series of instructions to be executed on a single computing resource. The scientific applications are categorized with respect to the dependencies among their tasks into three categories, namely: sequential, tightly-coupled, and workflows. The sequential applications are constituted by a batch of tasks having no dependencies, whereas the tightly-coupled applications constitute of highly cohesive tasks having enormous communication requirements. The workflows comprise of interdependent tasks with less frequent communication requirements as compared to the tightly-coupled applications. The focus of this thesis is to provide efficient scheduling algorithms for the workflows in dynamic grid environments.

Figure 1-1 shows a typical grid environment. The resources from different part of the globe register themselves with a Grid Information Service (GIS) to form a grid environment. The grid users utilize these geographically dispersed grid resources for executing their applications (e.g., fractal image compression and genome decoding). The grid users submit their applications (comprising of multiple tasks) to a specific grid scheduler. The grid scheduler collects information about the grid resources from the GIS and maps the tasks of grid users’ application to the grid resources with an aim to optimize a certain objective (e.g., makespan, which is defined as the time elapsed between the start of the execution of the first task and end of the execution of the last task). The grid resources execute the assigned tasks and send the results to the grid scheduler after their successful completion. The grid scheduler collects the results and subsequently sends them to the grid users when their applications finish the execution.
1.1.2. Workflows

Workflows constitute a considerable portion of the scientific applications utilizing the grid environments. Some of the familiar examples of the workflow-based scientific applications
are Maxillo-facial Surgery [Cao et al., 2003] and Magnetic Resonance Imaging (MRI) [Yu, 2007] from medical sciences, Laser Interferometer Gravitational Wave Observatory [Pandey and Buyya, 2008] from physics, and Genome Decoding [Simpson et al., 2004] from bioinformatics. Usually a workflow is represented by the nodes and edges, where the nodes denote the tasks of the workflow and the edges denote the precedence order between the tasks [Kwok and Ahmad, 1999a; Kwok and Ahmad, 1999b]. Workflows can be cyclic or acyclic. This thesis is limited to the study of acyclic workflows, which are referred as workflows simply. Figure 1-2 shows an example structure of a workflow.

![Example workflow diagram](image)

**Figure 1-2: An example workflow**

As shown in Figure 1-2, the workflow consists of a set of tasks, where each task is identified by its task number. The computational weight, denoted by the weight of a node in Figure 1-2, is defined as the computed length of the task [Kwok and Ahmad, 1999b; Fangpeng, 2007]. Similarly, the communication weight, denoted by the weight of an edge between two nodes in Figure 1-2, is defined as the size of data to be transferred between the tasks [Kwok and Ahmad, 1999b; Fangpeng, 2007]. The direction of an edge denotes the precedence order between two tasks of a workflow, where the task at the starting point of the edge is called the parent task and the task at the ending point of the edge is called the child task. E.g., the computational weight of the task number 0 (see Figure 1-2) is seven. The task
number 0 is the parent task of the task number 1 and the task number 2. The communication weight of the edge between the task number 0 and the task number 1 is one.

Typically, the workflow-based applications need a variety of resources with different resource characteristics (e.g., number of processors, processing capabilities, data storage capacity, etc.), which may not be provided at a single site. Moreover, all the tasks of a workflow do not necessarily require the same set of resources. The applications are designed as a workflow for efficient execution on a grid due to following reasons:

1. Parallel execution potentially reduces the overall execution time by assigning the tasks of a workflow-based application to the different grid resources, instead of executing the application on a single resource.

2. If the tasks of a workflow-based application are scheduled on a single resource then the resource failure requires rescheduling the entire application. On the contrary, if the tasks of the application are scheduled on multiple grid resources then a resource failure will only require rescheduling of the failed resource’s task.

3. A workflow can be partitioned and the execution of each partition can be assigned to different team members working on a scientific experiment. In this way, the complexity of control and management of the experiment can be reduced.

4. Different scientists distributed across the globe may collaborate to conduct experiments involving workflow-based applications, thereby increasing the collaborations among the scientific communities.

Workflow management involves workflow scheduling, rescheduling, resource failure management, and data management activities [Yu, 2007; Yu and Buyya, 2006]. The grid workflow scheduling is the primary activity of a grid Workflow Management System (WMS). The grid resources belong to different administrative domains and have site-specific policies (i.e., scheduling algorithm, load balancing algorithm, etc.). Moreover, the grid is highly dynamic and unpredictable by nature and particularly resource failure is common. Therefore, grid workflow scheduling is a challenging task [Caminero et al., 2007].
1.1.3. Grid workflow scheduling

As shown in Figure 1-3, a grid workflow scheduler schedules the tasks of a workflow to the resources of a grid in three phases, i.e., Information Collection, Mapping, and Data Migration and Task Assignment.

The three phases of scheduling are described next.

**Information Collection**

The grid workflow scheduler collects the information about the grid resources (e.g., the grid resource’s availability, its total number of processors, number of free processor, resource architecture, etc.) from the GIS in the Information Collection phase. Based on this information the state of the grid resources is estimated.

**Mapping**

A grid workflow scheduling algorithm (which is a part of the grid workflow scheduler) takes the scheduling decisions of mapping the tasks of the workflow to the grid resources in the Mapping phase. The scheduling decisions are made with an aim of optimizing certain objectives (e.g., the makespan of the workflow). Usually, two important factors are considered for optimizing the makespan of a workflow, namely: the computation time of the tasks of the workflow and the communication time between the tasks (see e.g., [Pandey and Buyya, 2008; Yu et al., 2008]). The computation time of a task of a workflow is calculated using the computational weight of the task (cf. Figure 1-2) and the processing capabilities of the grid resource on which it is mapped. Similarly, the communication time between a child task and one of its parent tasks is calculated using the communication weight of the edge between the tasks and the network characteristics of the grid resources on which the tasks are mapped [Yu et al., 2008].
Data Migration and Task Assignment

The parent tasks produce data for their child tasks. The availability of the data produced by all the parent tasks is a necessary condition for a child task to start its execution. Therefore, this data has to be migrated to the resource at which the child task is scheduled. The *Data Migration and Task Assignment* phase implements the decisions taken in the *Mapping* phase by assigning the tasks of the workflow to the specified grid resources. In addition, the data required to start a task is migrated to the grid resource. Following are the challenges faced while designing a grid workflow scheduling algorithm.

1. A child task has to wait for all its parent tasks to finish their execution before it can start executing. Moreover, the child task can only start its execution after all the data produced by its parent tasks are migrated to the resource (at which the child task is scheduled). E.g., in Figure 1-2 the execution of task number 4 can start only when the tasks 1, 2, and 3 finish their execution, and the produced data is migrated at the grid resource where the task 4 has to start its execution. Hence, the dependency constraints of the tasks must be taken into account while designing a workflow scheduling algorithm.

2. The communication time between the workflow’s tasks (scheduled on the different grid resources) must be taken into account in order to avoid overwhelming the parallelism benefits.

3. A grid workflow scheduling algorithm has to find the task-to-resource mapping for all the tasks of the workflow with an aim to optimize certain objectives like user-specified deadline, Quality of Service (QoS), etc. [Yu and Buyya, 2005a; Yu and Buyya, 2006]. Sometimes these objectives are conflicting, which adds to the complexity of the workflow scheduling problem. E.g., the user-specified deadline and the grid resources’ utilization are conflicting objectives therefore it is hard to find a solution that optimizes both objectives simultaneously.

The next section describes the problem of workflow scheduling in a dynamic grid environment.
1.2. Workflow Scheduling in a Dynamic Grid Environment

A grid workflow schedule contains the task-to-resource mapping for each task of a workflow. A grid workflow schedule can be characterized by the achieved makespan of the workflow. The general problem to determine an optimal schedule is known to be NP-hard [Coffman and Bruno, 1976]. Therefore, several heuristics have been invented in contemporary research approaches to find the grid workflow schedules in a reasonable time (see e.g., [Yu and Buyya, 2006; Yu, 2007]). All these heuristics assume that the computation time of the tasks of a workflow and the communication time between the tasks of the workflow with respect to a grid environment are known in advance. However, this assumption is unrealistic for real grid environments. Even an optimal schedule (based on the precise computation communication time) cannot be materialized due to the performance fluctuations of the grid resources at runtime [Prodan and Fahringer, 2008]. The studies presented in [Kwok and Ahmad, 1999b; Yu and Buyya, 2006] solve the problem of grid workflow scheduling in a static manner without considering the dynamicity of a grid environment. In [Yu and Shi, 2007] the authors attempt to solve the workflow scheduling problem using a hybrid technique, where the initial schedule generation is based on a static approach and the resource failure is handled by dynamic rescheduling. It is therefore evident that traditional grid workflow scheduling approaches oversimplify the problem by taking unrealistic assumptions, and there is a pressing need to develop grid workflow scheduling algorithms for realistic grid environments. The following subsections define the key research issues in the grid workflow scheduling to be considered in this thesis.

1.2.1. Workflow-based workload modeling

The performance of a grid environment depends on its workload, i.e., the workflows submitted to the grid workflow scheduler in a specified time period. Therefore, it is important to explore the characteristics of the workload (e.g., the arrival time of the workflows, the makespan of the workflows, etc.) in order to study the effectiveness of grid workflow scheduling algorithms. Various approaches utilize workload models, which are based on randomly generated workflows (see e.g., [Sakellariou and Zhao, 2004]). Currently, there is no workflow-based workload model available, which can be used to test newly developed workflow scheduling algorithms with respect to the effects of different workload
characteristics. Therefore, this thesis investigates the research issue of workflow-based workload modeling.

1.2.2. Dynamicity of the grid resources

In a grid environment, the grid resources from different organizational domains form a VO. The grid resources may also have Local Users (LUs), who may submit their tasks to the resources. Hence, the tasks of the workflows submitted to the grid resources have to compete with LUs’ tasks. On the contrary, most previous studies (see e.g., [Sakellariou and Zhao, 2004; Li, 2008]) assume the grid resources to be dedicated, i.e., without any LUs. This assumption is not realistic in a real grid environment.

Since the LUs’ load at a grid resource (defined as a function of the number of tasks submitted by the LUs to the grid resource and the tasks’ lengths) may fluctuate over the time, this fluctuation affects the schedule of the workflow’s tasks mapped on the grid resource. Traditional grid workflow scheduling algorithms do not consider the LUs and their (possibly fluctuating) load at the grid resources. This thesis investigates the LUs’ load at the grid resources in grid workflow scheduling.

1.2.3. Multiple workflows

Most of the previous studies consider a centralized grid workflow scheduler that receives the workflows from all grid users (see e.g., [Sakellariou et al., 2007; Yu, 2007; Yu and Shi, 2010]). The assumption of a centralized scheduler can be true in a cluster environment but it does not hold for the grid (which is considered as a cluster of clusters). Moreover, the centralized scheduler approaches assume direct control of grid resources, whereas in real grid environments the resources belong to different administrative domains with site-specific policies. In a real grid environment, the tasks submitted to a grid resource are scheduled within the grid resource according to its site-specific scheduling algorithm.

The previous researches assume that there is only one workflow in execution at a time in the grid (see e.g., [Yu, 2007; Yu and Shi, 2007]), but in real grid environments multiple workflows may execute simultaneously. Therefore, this thesis considers a decentralized architecture of grid workflow scheduler in order to handle multiple workflows.
1.2.4. Queue wait time

The tasks of multiple workflows compete for accessing a grid resource, thereby waiting in the resource’s queue. The time spent by the workflow tasks in the grid resource’s queue is called *queue wait time* at the grid resource. The grid workflow scheduling algorithms being unaware of the other workflows’ tasks in the grid resources’ queues may submit their workflows’ tasks to the fastest grid resources, thereby creating longer queues at the grid resources and hence increasing the queue wait time. Therefore, the makespan of the workflows may be high for the grid workflow scheduling algorithms unaware of the queue wait time. This thesis investigates the *queue wait time* in grid workflow scheduling.

1.2.5. Data and task co-scheduling

Theoretically, the grid provides unlimited computational power, but in reality the grid resources have limited computational capabilities, which must be effectively utilized. As discussed in subsection 1.1.3, the computation time of the tasks of a workflow and the communication time between the tasks are dependent on the characteristics of the grid resources. Some of the early approaches of workflow scheduling do not consider the communication between the tasks of a workflow [Kwok and Ahmad, 1999a]. The authors in [Afzal et al., 2006] assume that the connecting links of the grid resources have unlimited bandwidth with minimum or zero latency and hence the data movement time between the tasks of a workflow is negligible. Both of these assumptions are unrealistic because the communication time may even overwhelm the benefits of parallelization if it is not handled properly [Amalarethinam and Selvi, 2011]. Also, the workflow scheduling algorithms involving advance reservations (see e.g., [Sulistio et al., 2006]) assume that the computation time of the tasks of the workflows and the communication time between the tasks of the workflows are known in advance, which is not realistic again. This thesis considers a realistic grid network configuration with limited amount of links’ bandwidths and network latencies, and proposes the data saving algorithms for minimizing the data movement time among the tasks of a workflow. The data movement time of a child task of a workflow is calculated using the communication time between it and all its parent tasks. Hence, it is evident that a grid workflow scheduling algorithm must consider the data movement time among the tasks of a workflow as well as the tasks’ computation time in order to achieve a lower makespan. This thesis investigates the *data movement time* in grid workflow scheduling.
1.2.6. Resource utilization

A grid workflow schedule can be characterized by the achieved utilization of the grid resources. The resource utilization of a grid resource is defined by [Sabin et al., 2003] as the fraction of the processing capabilities of the grid resource used with respect to the total available processing capabilities of the grid resource in a specific time interval. Similarly, the resource utilization can also be defined for the whole grid environment in terms of the resource utilization of the grid resources in a specific time interval. Most of the grid workflow schedulers consider the makespan of a workflow as an objective without considering the utilization of the grid resources [Wieczorek et al., 2007]. However, the resource utilization of the grid resources is an important issue from the viewpoint of the grid resource provider in a dynamic grid environment. Therefore, this thesis considers the resource utilization as a research issue.

1.3. Research Problems

The section 1.2 presented the key research issues that must be addressed for grid workflow scheduling in a dynamic grid environment. Following are the research problems investigated in this thesis:

1.3.1. Workflow-based workload modeling

RP1. How to model a workflow-based workload model for a dynamic grid environment?

1.3.2. Multiple workflows

RP2. How to handle multiple workflows?

1.3.3. Local Users’ load

RP3. How to handle the LUs’ load at the grid resources?

1.3.4. Data movement time

RP4. How to handle the data movement time of the tasks of a workflow mapped on different grid resources?
1.3.5. Queue wait time

RP5. How to handle the queue wait time of a workflow’s tasks at the grid resources due to workflows’ tasks and LUs’ tasks competing for the grid resources?

1.3.6. Workflow scheduling algorithms

Novel grid workflow scheduling algorithms are required for addressing the above research problems in grid workflow scheduling (Research Problems RP1-RP5). Therefore the overarching research problems investigated in this thesis are:

RP6. How to schedule the tasks of a workflow in the presence of the LUs’ load at the grid resources with an objective of obtaining a lower makespan?

RP7. How to schedule the tasks of a workflow being aware of the queue wait time at the grid resources with the objectives of obtaining a lower makespan and good resource utilization?

1.4. Proposed Solution

This thesis provides a solution to the above research problems (Research Problems RP1-RP7). It proposes the following models:

1. A workflow-based workload model for dynamic grid environments.
2. A prediction model for predicting the LUs’ load at the grid resources.
3. A prediction model for predicting the queue wait time of the task of a workflow at the grid resources due to the multiple competing workflows and the LUs of the grid resources.

This thesis utilizes the above models to design and implement novel grid workflow scheduling algorithms. Together, the models and the scheduling algorithms solve the research problems listed in the preceding section. The proposed workflow-based workload model follows the standard workload format [SWF, 2012] to model the different characteristics of the workload (e.g., the arrival time of a workflow, the number of processors required by a workflow, memory requirement of each task of the workflow, etc.). The workflow-based workload model contains forty-eights thousands workflows, which
belong to the different classes of the HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007]. This workflow test bench is designed for *Heterogeneous Computing* (HC) environment; therefore, it is named as HC-test bench.

Figure 1-4 shows an overview of the proposed solution. It is assumed that each *Grid User* is the owner of many workflows contained in the *Workflow-based Workload Model*. The *Grid User* retrieves the workflows from the *Workflow-based Workload Model* and submits one workflow at a time to a *Prediction-based Grid Workflow Scheduler* according to the specified arrival time of the workflow. The grid resources register themselves with the GIS. The GIS assigns unique Identifier (ID) to each grid resource in the registration process.

![Figure 1-4: Overview of prediction-based grid workflow scheduling](image)

The *Prediction-based Grid Workflow Scheduler* shown in Figure 1-4 implements a scheduling algorithm. First, the *Prediction-based Grid Workflow Scheduler* consults the GIS to acquire the grid resources’ information. Second, the *Prediction-based Grid Workflow Scheduler* utilizes the acquired information as well as the prediction models (i.e., LUs’ load prediction model and the queue wait time prediction model) to map the tasks to the workflow to the grid resources. The objective of this mapping is to minimize the makespan of the workflow.
Third, the Prediction-based Grid Workflow Scheduler dispatches the tasks of the workflow to the grid resources as specified in the schedule. As shown in Figure 1-4, more than one Prediction-based Grid Workflow Schedulers may dispatch the tasks of the workflows to the grid resources. A grid resource sends the results of a successfully completed task of a workflow to the corresponding Prediction-based Grid Workflow Scheduler. There may also be some LUs utilizing the grid resources.

1.5. Contributions

This thesis provides the following contributions:

1. **Workflow-based workload model**: The workflow-based workload model composed of forty eight thousand workflows is designed and implemented. The static characteristics (number of tasks in a workflow, the number of dependencies among the tasks of a workflow, the number of processor required by each task of a workflow, the memory requirements of each task of a workflow) and the dynamic characteristics of the workload (i.e., the arrival time of workflows, computation time of the tasks of a workflow, communication time between the tasks of a workflow, and makespan of a workflow) are modeled for the European Data Grid. In the workload model, the workflows are categorized according to the number of tasks into three classes, namely: small, medium and large.

2. **LUs’ load prediction**: A mathematical model is proposed and implemented to predict the LUs’ load at the grid resources. This model is verified through the real workload traces.

3. **Queue wait time prediction**: A queuing theory-based prediction model is proposed and implemented for predicting the average queue wait time of the grid resources. The prediction model is verified through simulation and real workload traces.

4. **Prediction-based grid workflow scheduler**: The decentralized architecture of prediction-based grid workflow scheduler is designed and implemented. The scheduler uses the proposed workflow-based workload model as well as the LUs’ load and queue wait time prediction models.
5. **Multiple workflows**: The decentralized architecture of the grid workflow scheduler allows concurrent execution of multiple workflows from different grid workflow schedulers.

6. **Novel scheduling and data saving algorithms**: Novel scheduling algorithms are designed and implemented in the prediction-based grid workflow scheduler. The proposed scheduling algorithms are LUs’ Load Prediction (LULP)-based Min-Min, Max-Min and Sufferage (i.e., LULP-Min-Min, LULP-Max-Min and LULP-Sufferage). The LULP-Min-Min, LULP-Max-Min and LULP-Sufferage algorithms utilize the predicted LUs’ load at the grid resources for scheduling the tasks of the workflows on the grid resources. The objective of these algorithms is to achieve a lower makespan of the workflows. However, these algorithms prioritize the tasks of the workflows differently. Another scheduling algorithm, called Queue Wait Time Prediction-based Grid Workflow Scheduling (QWP-GWS), is designed and implemented in the prediction-based grid workflow scheduler. The QWP-GWS algorithm utilizes the predicted queue wait time of the grid resources for scheduling the tasks of the workflow. The objectives of the QWP-GWS algorithm are to achieve a lower makespan of the workflows and better resource utilization of the grid resources. The data saving algorithms are designed and implemented in the prediction-based grid workflow scheduler for reducing the data movement time among the tasks of the workflows. This helps in minimizing the makespan of the workflows.

7. **Scheduling workflow tasks in the presence of LUs**: The grid workflow scheduling algorithms proposed in this thesis schedule the tasks of the workflows to the grid resources while considering the presence of the LUs at the grid resources.

8. **Adaptation**: The prediction-based grid workflow scheduler considers the dynamicity of the grid environment. The prediction-based grid workflow scheduler updates the predictor model in case of any deviation from its predicted values (e.g., the queue wait time, LUs’ load at a grid resource etc.).

1.6. **Organization of the Thesis**

This thesis is organized into the following chapters:
Chapter 2. Problem Analysis: This chapter presents a detailed analysis of the seven research problems (Research Problems RP1-RP6; cf. Section 1.3). First part of the chapter describes the grid workflow scheduling. The second part of this chapter derives requirements for the solution of the research problems.

Chapter 3. State of the Art: This chapter investigates the current approaches with respect to the requirements identified in chapter 2, and identifies the deficits of the current approaches.

Chapter 4. Grid Simulation Tools and Workflow Test Bench Suites: This chapter discusses the existing grid simulators, which can be utilized for testing the grid workflow scheduling algorithms (like Bricks, OptorSim, GangSim, SimGrid, GridSim, Alea, GSSIM, and GES). The simulators are analyzed with respect to the authenticity of the simulators’ results, the scalability of simulation, and the failure modeling in the simulators. Based on this analysis, this chapter provides reasons for the selection of the GridSim simulator for evaluating the grid workflow scheduling algorithms proposed in this thesis. Also, this chapter presents a discussion of the existing test bench suites for modeling grid workflows, and lists the reasons for selecting the HC-test bench.

Chapter 5. Workflow-based Workload Model: This chapter proposes a workflow-based workload model. The essential steps for developing a simulation experiment using the GridSim simulator and the workflow-based workload model are also described in this chapter.

Chapter 6. System Design of the Prediction-based Grid Workflow Scheduler: In this chapter, the architecture of a prediction-based grid workflow scheduler is described. Moreover, a design specification of the parts of the scheduler is presented using the constructs of the Unified Modeling Language (UML).

Chapter 7. Grid Workflow Scheduling Using the Grid Resources’ LUs’ Load Prediction: This chapter proposes a prediction model for predicting the LUs’ load at the grid resources. This prediction model is implemented in the prediction-based grid workflow scheduler (cf. Chapter 6), and the resulting system is utilized to simulate the proposed scheduling algorithms, i.e., LULP-Min-Min, LULP-Max- Min, and LULP -
Sufferage. Moreover, the Heterogeneous Earliest Finish Time First (HEFT) algorithm is also simulated and the results of the proposed algorithms are compared with the results of the HEFT algorithm.

Chapter 8. Queuing Theory-based Grid Workflow Scheduling: This chapter proposes a prediction model for predicting the queue wait time of a task of a workflow in the queue of a grid resource. This prediction model is implemented in the prediction-based grid workflow scheduler (cf. Chapter 6), and the resulting system is utilized to simulate the proposed QWP-GWS scheduling algorithm. Moreover, the Min-Min, Max-Min, Sufferage and HEFT algorithms are also simulated. The results of the QWP-GWS algorithms are compared with the results of the Min-Min, Max-Min, Sufferage and HEFT algorithms.

Chapter 9. Conclusions and Future Directions: This chapter concludes this thesis by presenting a summary of contributions and points to some directions for future research.
2. Problem Analysis

This chapter analyses the seven research problems (Research Problems RP1-RP7) presented in section 1.3, namely: workflow-based workload modeling, LUs, multiple workflows, dynamic grid environment, and workflow scheduling. Firstly, section 2.1 introduces the basic concepts of grid workflow scheduling. Secondly, section 2.2 analyses the research problems and identifies the requirements for solving them. Finally, section 2.3 presents a summary of the identified requirements.

2.1. Grid Workflow Scheduling

This section describes the basic concepts involved in grid workflow scheduling. It is organized as follows: Subsection 2.1.1 describes the types of the grid. Subsection 2.1.2 discusses the characteristics of the grid resources. Subsection 2.1.3 describes the structure of a workflow. Subsection 2.1.4 discusses the grid workflow scheduling problem. Subsection 2.1.5 presents a classification of the grid workflow schedulers. Subsection 2.1.6 describes a categorization of the grid workflow scheduling algorithms. Subsection 2.1.7 describes the information required to a grid workflow scheduling algorithm for taking the scheduling decisions. Subsection 2.1.8 classifies the scheduling objectives considered by the grid workflow scheduling algorithms, whereas subsection 2.1.9 highlights the issues related to the data movement among the tasks of a workflow. Finally, subsection 2.1.10 discusses the impact of the scheduling decisions taken by the grid workflow scheduling algorithms on the considered scheduling objective.

2.1.1. Types of the grid

The aim of a grid is to provide computation as a utility to the individuals and communities in their homes and offices like electricity, water and telephone [Foster et al., 2002]. The grid is defined by [Foster and Kesselman, 2000] as:

“[It is a] hardware and software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.” (p. 3)
The above definition provides the notion of the grid as a VO of globally distributed computational and data resources having their site-specific policies, giving a consistent single resource view to the users. Such a pervasively available grid is utilized by the scientific communities for the execution of the applications requiring the high-end computational capabilities. The grid provides the reliability insofar as it provides assurances of the best-effort performance to the users. Therefore, the users can access (relatively) inexpensive computational capabilities for their applications instead of requiring an expensive single ‘super resource’. This traditional view of the grid is to share the resources among different scientific communities. Therefore, such a grid is called the community grid.

Buyya and Murshed proposed the idea of using the metaphors of economics (e.g., budget, deadline) for resource management in the grid environment [Buyya and Murshed, 2002]. This essentially means that the computational capabilities of the grid resources are not shared but they are considered as a commodity available on a cost. This view of the grid treats computing as a utility provided on commercial basis. Therefore, such a grid is called the utility grid. Figure 2-1 shows the classification of the grid into the two main classes, i.e., Community Grid and Utility Grid.

![Figure 2-1: Types of the grid](image)

The focus of this thesis is to develop novel grid workflow scheduling algorithms for a class of the non-commercial scientific applications, i.e., workflows. Therefore, this thesis limits its scope to the community grid, which is referred as the grid in the subsequent discussion.

2.1.2. Grid resources’ characteristics

The grid resources are different from the resources in conventional parallel and distributed computing environments (like clusters) in many aspects e.g., the grid resources may belong to different parts of the globe and are different from each other in terms of their
characteristics (i.e., processing capabilities, network characteristics etc.). Figure 2-2 list different resource characteristics of the grid resources, namely: Diversity, Site-specific Policies, Dynamicity, and Type. Each of these resource characteristics is described next.

**Figure 2-2: Resource characteristics**

**Diversity**

Since many organizations share their resources to form a grid, the grid resources have a diversity of resource characteristics (in terms of their processing capabilities, data storage capacity, and network characteristics) as shown in Figure 2-2. A grid resource can be a Personal Computer (PC), a cluster, or a supercomputer. However, in a grid all or some
resources may be *Homogenous*. It is also possible that all the grid resources are *Heterogeneous*.

**Site-specific policies**

Since the grid resources are owned by different organizations, these organizations have site-specific policies (e.g., scheduling algorithm, resource sharing) for managing these resources. E.g., the organizations may specify different resource sharing mechanisms for their grid resources. As shown in Figure 2-2 a grid resource may be *Dedicated, Non-dedicated*, or *Prioritized* as specified by the resource sharing mechanism of the respective organization.

1. **Dedicated**: An organization may allow a resource to execute the grid’s tasks only assuming the absence of LUs. Such grid resources are always available with their full capacities (processing capabilities, storage, etc.).

2. **Non-dedicated**: Another organization may allow the LUs’ tasks and the grid’s tasks to compete with each other for a resource and the resource is allocated on first come first serve basis. Such grid resources are not available with their full capacities, i.e., they are non-dedicated.

3. **Prioritized**: An organization may prioritize the LUs’ and grid’ tasks. A higher priority may be assigned to the tasks of the LUs as compared to the tasks of the grid. In the other case, some organizations may assign a higher priority to the grid’s tasks of some time-critical applications. The reader is referred to [Ma and Buyya, 2005; Wiriyaprasit and Muangsin, 2004] for a detailed discussion on the priority-based resource sharing mechanisms.

**Dynamicity**

Some approaches of the grid workflow scheduling assume the grid resources to be *static* for the sake of simplicity (see e.g., [Yu, 2007; Yu and Buyya, 2006]). However, the grid resources are high dynamic and their characteristics (e.g., the grid resources’ load) may vary over the time as discussed by [Caminero et al., 2007]. A grid resource may be available with its full capacity when it joined the grid, but it may become overloaded with its LUs’ load after some time. Similarly, a grid resource, which was initially heavily loaded, may turn lightly loaded or even totally idle over the time. Another example of time-varying characteristic of the grid
resources is their network. The grid resources are connected through the shared network links and they may also be involved in forwarding the packets not necessarily belonging to them. Since the network traffic may vary over the time, changing the network characteristics of the grid resources.

**Type**

As shown in Figure 2-2 there may be *Computation, Communication, and Storage* resources in a grid. The grid applications utilize these resources according to their requirements.

### 2.1.3. Structure of a workflow

As described in section 1.1.2, the workflows can be represented by the nodes, the edges and their respective weights (see Figure 1-2 for an example workflow). The structural components of a workflow are the number of tasks and edges in the workflow, the computational weight on the tasks and the communication weights on the edges. Figure 2-3 shows the structural components of a workflow.

![Figure 2-3: Structural components of a workflow](image)

The workflows can be classified into different classes according to the *number of tasks*, *the number of edges*, *the computational weights* on the tasks, and the *communication weights* on the edges [Hönig and Schiffmann, 2004; Hönig, 2007]. Such a classification of the workflows is required for evaluating the performance of grid workflow scheduling algorithms. Selecting a variety of workflows belonging to the different classes avoids any biasedness in the evaluation.
2.1.4. Grid workflow scheduling problem

The grid workflow scheduling problem is defined by [Yu and Buyya, 2005b] as:

"[Workflow scheduling] focuses on mapping and managing the execution of inter-dependent tasks on shared resources that are not directly under [the workflow system's] control." (p. 7)

The above definition indicates that the tasks of a workflow are mapped to the shared grid resources according to the computation, communication and data storage requirements of the tasks. Such mapping aims to optimize a certain objective (e.g., makespan). Considering the dynamic nature of the grid resources, the grid workflow scheduling process needs to manage the execution of the workflows’ tasks (e.g., by rescheduling the tasks mapped to a failed resource, by rescheduling the tasks missing the expected completion time, etc.).

Formally, a workflow is represented by a directed acyclic task graph $G(T, E)$, where $T$ is a finite set of the tasks of a workflow and $E$ denote the edges between the tasks of the workflow (representing their dependencies). $R$ is a finite set of the grid resources. The aim of the grid workflow scheduling process is to map each task from the set $T$ to a suitable resource from the set $R$ with an objective to optimize the makespan of the workflow. In order to find an optimum makespan all possible task-to-resource mappings must be considered. The number of such mappings equals to $M^N$, where $M$ is the number of resources in the set $R$ and $N$ is the number of tasks in the set $T$. Since it is not possible to find all possible mappings in a reasonable time, grid workflow scheduling with an optimum makespan is an $N$-P complete optimization problem [Coffman and Bruno, 1976]. Therefore, the objective of the grid workflow scheduling process is reformulated to find the task-to-resource mappings for each task of a workflow by approximating the solution. The workflow scheduling process must also consider the precedence order between the tasks of a workflow, which is defined by the dependencies among the tasks of the workflow. A task of a workflow cannot execute until all of its parent tasks complete their execution and the data required to start its execution is transferred to the grid resource where it has to start its execution.
2.1.5. Grid workflow scheduler’s architecture

As shown in Figure 2-4, a grid workflow scheduler can be categorized into the *Centralized*, *Hierarchical* and *Decentralized* categories with respect to its architecture. The decentralized grid workflow schedulers can be Cooperative or Non-cooperative. Next, each of these categories of the grid workflow schedulers’ architecture is described.

![Figure 2-4: Grid workflow scheduler’s architecture](image)

**Centralized**

In the centralized architecture, there is only one central grid workflow scheduler present in a grid. All the workflows are submitted to the central grid workflow scheduler. Conventional problem of scalability is associated with central grid workflow scheduler architectures. Moreover, the decision of task-to-resource mappings for the tasks of a workflow requires the information about the grid resources and workflows. The amount of information increases nonlinearly as the number of resources and the number of workflows increases. Hence, it is evident that the centralized architecture scheduler is suitable only for the workflows having less number of tasks and the grids having less number of resources.
Hierarchical

In the hierarchical architecture, more than one grid workflow schedulers work in a hierarchy in a grid. The top most grid workflow scheduler is the master of all the underlying grid workflow schedulers in the hierarchy. The benefit of the hierarchical architecture is that more than one grid workflow scheduling algorithms can be implemented at the different levels of the hierarchy. However, the failure of the master grid workflow scheduler failure may lead to the failure of all the grid workflow schedulers present in the hierarchy.

Decentralized

In the decentralized architecture there can be more than one grid workflow schedulers present in a grid, where all of them are equal, as opposed to the hierarchical architecture. The decentralized architecture does not suffer from the scalability problem as central and hierarchical architecture does. The decentralized grid workflow schedulers may share the information among each other [Ranjan et al., 2008], this type of the grid workflow schedulers are called cooperative. On the contrary, the decentralized grid workflow schedulers not sharing information among each other are called non-cooperative as shown in Figure 2-4.

2.1.6. Grid workflow scheduling algorithms

As shown in Figure 2-5, contemporary grid workflow scheduling algorithms can be categorized as Static, Dynamic, or Hybrid.

![Figure 2-5: Dynamism of the grid workflow scheduling algorithm](image)

Next, each of the categories of the grid workflow scheduling algorithms is described.
Static

The static grid workflow scheduling algorithm prepares a schedule, which contains the task-to-resource mappings for each task of a workflow before the workflow starts its execution (see e.g., [Yu and Buyya, 2006; Yu, 2007; Wieczorek et al., 2007]). A static grid workflow scheduling algorithm assumes that the accurate and precise information (e.g., the computation time of the tasks of a workflow on each grid resource, the start time of a task at a grid resource etc.) is known in advance. However, this assumption is unrealistic for a real grid having grid resources having dynamic resource characteristics (see the discussion in subsection 2.1.2).

Dynamic

A dynamic grid workflow scheduling algorithm postpones the scheduling decisions of mapping the tasks of a workflow to the grid resources as long as possible. The scheduling decision to map a task to a grid resource is made just before the task’s execution starts. The examples of the dynamic grid workflow scheduling algorithms are Min-Min, Max-Min, Sufferage and XSufferage [Maheswaran et al., 1999; Yu et al., 2008; Rehman, 2010].

Hybrid

A hybrid grid workflow scheduling algorithm combines the properties of a static and dynamic grid workflow scheduling algorithm. An example hybrid grid workflow scheduling algorithm is presented in [Yu and Shi, 2007], where the tasks of a workflow are scheduled statically but they are dynamically rescheduled in order to cope with the dynamicity of the grid resources (i.e., failure of the grid resource).

2.1.7. Information required to a grid workflow scheduling algorithm

The grid workflow scheduling algorithm requires information about the workflows as well as about the grid resources in order to take the scheduling decisions, where some of the information may be incomplete (e.g., a grid resource’s load) or dynamic (e.g., the resource availability of a grid resource). The information required to a grid workflow scheduling algorithm can be categorized as shown in Figure 2-6.
Next, the categorization of the information required to a grid workflow scheduling algorithm (Figure 2-6) is described.

**Workflow information**

The information required to make the scheduling decisions related to a workflow may include:

- the expected execution time of a task of the workflow each of the grid resources,
- the expected data movement time between the tasks of the workflow scheduled on two different grid resources,
- the number of processors required for the tasks of the workflow tasks, and
- a specific Operating System (OS) and hardware architecture of the grid resource required for executing the tasks of the workflow, etc.
Resource information

The information required to make the scheduling decisions related to a grid resource may include:

- the availability of a grid resource
- the number of processors in the grid resource,
- the number of free processors available in the grid resource,
- the information about the LUs’ load at the grid resource during a particular time,
- the information about the grid resource’s queue (e.g., the number of tasks waiting in the queue, the queue wait time, etc.),
- the level of trust of a grid resource as defined in [Azzedin and Maheswaran, 2002], and
- the chance of resource failure

Some of this information can be obtained from the GIS (like the number of processors in a grid resource), whereas and some of the information must be estimated. Usually, prediction techniques are used for estimating the missing or incomplete information (see e.g., [Sonmez et al., 2010]). These prediction techniques may be based on the historical data analysis and/or analytical modeling techniques [Yu and Buyya, 2005b].

2.1.8. Grid workflow scheduling objectives

A grid workflow scheduling algorithm optimizes certain objective while finding the task-to-resource mapping for the tasks of a workflow. The most common objective considered in many the workflow scheduling algorithms is the makespan [Fangpeng, 2007]. The makespan is defined as the time elapsed between the start of the execution of the first task of a workflow and the end of the execution of the last task of the workflow. Other objectives considered by the grid workflow scheduling algorithms include:

- the deadline specified for the execution of a workflow,
- the budget (in terms of the money to be paid) assigned for the execution of the workflow, and
the Quality of Service (QoS) parameters to be considered (e.g., the robustness of the grid resources executing the tasks of a workflow; [Yu and Buyya, 2006]).

Some grid workflow scheduling algorithms consider multiple objectives (see e.g., [Sakellariou et al., 2007; Yu and Buyya, 2006]). As shown in Figure 2-7, the grid workflow scheduling algorithms are categorized into two main classes with respect to the optimized scheduling objectives. The classes are Single Objective Optimizer and Multi Objective Optimizer as shown in Figure 2-7.

![Figure 2-7: Classification of the scheduling objectives](image)

The optimized scheduling objective may belong to a stakeholder in the grid environment. As shown in Figure 2-8, there are three main stakeholders involved in a grid environment, namely: the Grid User, the Grid Resource Provider and the Grid Management System, shortly referred as the Grid.

Figure 2-8 shows a classification of the scheduling objectives according to the perspective of the stakeholders involved in a grid environment. This classification is described next.

**Grid**

As shown in Figure 2-8, examples of the grid-wide scheduling objectives considered by a grid workflow scheduling algorithm may include:

- maximizing the throughput of a grid (i.e., number of tasks completed per unit time), and
- maximizing the number of trusted sites resource as defined in [Azzedin and Maheswaran, 2002] (the failure proof grid resources) in a grid, etc.
30  Chapter 2  Problem Analysis

As shown in Figure 2-8, the scheduling objectives from the perspective of a grid user (who is considered as the owner of a workflow) may include:

- achieving a lower makespan for the workflow,
• achieving a lower data movement time of the tasks of the workflow,
• executing the workflow with less budget,
• executing the workflow within a deadline, and
• executing the workflow so that certain QoS parameters (fault tolerance) are met, etc.

Since this thesis focuses the community grid, the proposed grid workflow scheduling algorithms aim at achieving a lower makespan.

**Grid resource provider**

As shown in Figure 2-8, the scheduling objectives from the perspective of the grid resource providers may include:

• achieving a maximum revenue by executing the tasks of a workflow,
• increasing the level of trust of the resource, and
• achieving a maximum resource utilization, etc.

The revenue earned by a grid resource provider is a focus of the utility grid. However, the grid resources are shared in a community grid and the grid resource providers generally want to achieve maximum resource utilization.

The scheduling objectives from the different stakeholders’ perspective may be conflicting with each other e.g., the tasks of a workflow need to be scheduled on the resources having highest processing capabilities in order to achieve a lower makespan. However, such a schedule leads to poor resource utilization since the resources having less processing capabilities will remain idle. Similarly, the grid users may want to execute a workflow with a lower makespan as well as within a specific budget, but assigning all the tasks of the workflow to the grid resources having high processing capabilities (in order to achieve a lower makespan) will exceed the budget (since the grid resources with high processing capabilities are usually costly). Due to the conflict between the objectives of different stakeholders, optimization of more than one objective in a grid workflow scheduling is a challenging and complex task [Yu, 2007].
2.1.9. Data movement time

The data movement time between the tasks of the workflow scheduled on different grid resources is an important factor in grid workflow scheduling. It affects the scheduling objectives (like makespan). A task of a workflow task is ready for execution when all of its parent tasks complete their executions and the data required by the task is available. A workflow task (having more than one parent tasks) has to wait for the completion of all its parent tasks. Some of these parent tasks may have completed their execution whereas others may be waiting for execution or they may be in execution. The data produced by the parent tasks has to be saved in order for it to be available for the child task later on. Usually, the grid workflow scheduling approaches assume that the data generated by the parent tasks is immediately used by the child task (see e.g., [Sakellariou et al., 2007]); therefore they do not address the data storage issues. Other approaches (e.g., [Ramakrishnan et al., 2007; Pandey and Buyya, 2008]) only focus on addressing the data storage issues without considering the scheduling objectives (like makespan, resources utilization). The three methods used to handle the data movement in a grid workflow scheduling are Centralized, Intercede and Peer to Peer as shown in Figure 2-9.

![Figure 2-9: Data movement methods](image-url)

Next, the methods of handling the data movement are described.
Centralized

The centralized data movement is the simplest way to handle the communication. Each communication between the tasks of a workflow is handled through a central point in this method. The central point can also be a centralized scheduler. Therefore, this method suffers from the problems of a centralized architecture (see the discussion in subsection 2.1.5).

Intercede

The data movement between the tasks of a workflow is handled using the Replica Catalog (RC) in the intercede method (see e.g., Pegasus system [Deelman et al., 2005]). The grid workflow scheduling process may become more complex in this case because the issues related to RC optimization need to be considered as well.

Peer to Peer

In the peer to peer method the data produced by the parent tasks is saved on any grid resource after it has completed its execution. This data is then directly transferred to the grid resource on which the child task is about to start its execution. Although this method seems to not suffering from the usual problems involved in the centralized and the intercede data movement methods, but this method assumes that the grid resources have the data storage units in order to save the data produced by the parent tasks.

2.1.10. Impact of the scheduling decisions on the scheduling objective

A grid workflow scheduling algorithm may take the scheduling decision to map the tasks of a workflow to the grid resources considering only the current task of the workflow (i.e., local decision), or considering all tasks of the workflow (i.e., global decision). The scheduling decision whether taken locally or globally impacts the considered scheduling objectives. Next, both these types of decisions are described.

Local scheduling decision

In case of the local scheduling decision, the grid workflow scheduling algorithm is concerned with mapping an individual task of a workflow to a grid resource and ignores the impact of
the task-to-resource mapping on the scheduling objective considered for the entire workflow. E.g., a grid workflow scheduling algorithm may map each task of a workflow on a grid resource where the task completes in less time in order to achieve a minimum completion time for the task. However, in this process the data movement time between the tasks of the workflow may get ignored. This may result a worse schedule in terms of the makespan of the workflow.

Global scheduling decision

In case of the global scheduling decision, the grid workflow scheduling algorithm maps the tasks of a workflow to the grid resources while considering the scheduling objective with respect to the entire workflow. E.g., a grid workflow scheduling algorithm considering the makespan as a scheduling objective takes the scheduling decisions for the task-to-resource mapping for each task of the workflow in such a way that the makespan of the workflow is minimized.
2.2. Requirement Analysis

This section explores the seven research problems (Research Problems RP1-RP7; cf. Section 1.3) and derives requirements on acceptable solutions. This section is organized as follows: Subsection 2.2.1 identifies the requirements of workflow-based workload modeling (Research Problem RP1). Subsection 2.2.2 identifies the requirement for a simulating a grid environment, which is needed for evaluating the grid workflow scheduling algorithms. Subsections 2.2.3-2.2.5 identify the requirements for a grid workflow scheduler in a dynamic grid environment (Research Problem RP2-RP5). Finally, subsection 2.2.6 drives the requirements for novel grid workflow scheduling algorithms (Research Problems RP6 and RP7).

2.2.1. Requirements of workflow-based workload modeling

Contemporary approaches evaluate grid workflow scheduling algorithms using randomly generated workflows (see e.g., [Sakellariou and Zhao, 2004; Yan and Chapman, 2007; Arabnejad, 2012]). These approaches do not follow any standardization with respect to the workflow generation, the structure of the workflows, and the representation of the workflows [Hirales-Carbajal et al., 2010]. However, a workflow-based workload is required for an efficient design and objective evaluation of novel grid workflow scheduling algorithms.

The characteristics of a workflow-based workload can be categorized into two categories, namely: the static characteristics and the dynamic characteristics. The static characteristics do not change over the time. They include the structure of the contained workflows (the number of tasks in a workflow, the dependencies among the tasks, etc.; see subsection 2.1.3) and the requirements of the workflows including:

- the number of processors required by each task of a workflow,
- the memory requirement of each task of a workflow, and
- other requirements (like the OS and the hardware architecture of the grid resource, etc.).

The dynamic characteristics of the workflow-based workload are dependent on the processing capabilities and network characteristics of the grid resources in a grid
environment. The dynamic characteristics change over the time because of the change in the processing capabilities and network characteristics of the grid resources. The dynamic characteristics include:

- the arrival time of the workflows at a grid workflow scheduler in a grid environment, and
- the makespan of each workflow for a grid environment.

Following are the three methods for generating the workflow-based workload:

1. **Real workload traces**: A workload can be generated by recoding the job logs of the computational resources of a grid. However, real workload traces contain workflows as well as other types of jobs (e.g., bags of tasks, pilot), and identifying workflows from these workload traces is difficult [Iosup and Epema, 2011]. Moreover, the job logs of the resources may also contain redundant information (e.g., the jobs which could not be successfully completed).

2. **Workflow of an application**: A workload can be generated from a particular application e.g., Maxillo-facial surgery [Cao et al., 2003], magnetic resonance imaging (MRI) [Yu, 2007], laser interferometer gravitational wave observatory [Pandey and Buyya, 2008] and Genome decoding [Simpson et al., 2004] etc.

3. **Synthetic workflow-based workload**: A workload can also be synthetically generated by modeling the characteristics of the workflow-based workload. The benefit of synthetic workloads over real workload traces is that the synthetic workloads can be easily managed and controlled. The synthetic workloads may also consider diversity in the structure of the workflows.

Since the real workload traces contain workflows as well as other types of jobs (e.g., bags of tasks, pilot), identifying workflows from these workload traces is difficult [Iosup and Epema, 2011]. Moreover, the job logs of the resources may also contain redundant information (e.g., the jobs which could not be successfully completed). The workload of a workflow-based application cannot be used to evaluate a grid workflow scheduling algorithm due to the lack of impartiality required for the evaluation. Since the synthetic workflow-based workload considers the diversity in the structure of the workflow, it provides the impartiality required
for evaluating novel grid workflow scheduling algorithms. However, a synthetic workload model considering the characteristics of the workload is missing in contemporary research.

A synthetic workflow-based workload model is required for designing and objectively evaluating novel grid workflow scheduling algorithms in order to avoid any biasedness in the evaluation (Research Problem RP1). Such a workload model needs to consider the static as well as dynamic characteristics of the workflow-based workload model. The static characteristics include the structure of the workflows and the requirements of the workflows.

As described in 2.1.3 the structure of a workflow is defined in terms of the number of tasks in a workflow, the computational weights of the tasks and the communication weights of the edges between the tasks. The workflows can be classified according to their structure [Hönig and Schiffmann, 2004; Hönig, 2007]. Such a classification needs to be considered for generating a synthetic workflow-based workload model. Therefore, the identified requirement is:

**R1.** A workflow-based workload model containing the workflows with a diverse structure is required.

The workflow-based workload model must also model the static characteristics of the contained workflows (i.e., the structure of a workflow, and the number of processors and the memory requirement of the tasks of a workflow). So, the following requirement is identified on the workflow-based workload model (Requirement R1):

**R2.** The workflow-based workload model must model the static characteristics of the contained workflows.

Since the dynamic characteristics of the workload are dependent on the processing capabilities and the network characteristics of the resources of the grid environment, the workflow-based workload model must model the dynamic characteristics once the specification of the grid environment is provided to it. Next, the requirements on modeling the dynamic characteristics of the workflow-based workload are identified.

The workflows arrive in a grid environment by following a certain pattern representing the behavior of the grid user [Baghban and Rahmani, 2008]. The synthetic workflow-based
workload must consider the arrival pattern in order to mimic the grid users’ behavior in a real grid environment [Feitelson, 2002]. Therefore, the following requirement is identified on the workflow-based workload model (Requirement R1):

**R3.** The workflow-based workload model must model the arrival time of the contained workflows in a grid environment.

As defined by the workflow scheduling research problems (Research problems RP6 and RP7), this thesis investigates the makespan optimization in grid workflow scheduling. The workflow-based workload must model the makespan of each of the contained workflows under ideal conditions in the grid environment (i.e., the dedicated grid resources and the network links among the resources). So, the following requirement is identified on the workflow-based workload model (Requirement R1):

**R4.** The workflow-based workload model must model the makespan of the contained workflows in a grid environment.

### 2.2.2. Requirements for simulating a grid environment

A repeatable and controllable test environment is needed for an effective evaluation of grid workflow scheduling algorithms [Buyya and Murshed, 2002; Quétier and Cappello, 2005]. The same set of grid resources having the specific resource characteristics and network configurations is required multiple times in order to evaluate a grid scheduling algorithm. However, due to the highly dynamic nature of the grid, resource availability is not guaranteed in a real grid environment [Buyya and Murshed, 2002]. If for the sake of argument, it is assumed that resources with similar characteristics will be available, the problem persists as the grid resources may now have other network configurations, which effects the communication time between the tasks of a workflow, thereby affecting the makespan of the workflow.

Building an elaborated grid test environment is time and budget consuming, as suggested by [Takefusa et al., 1999; Buyya and Murshed, 2002; Sulistio et al., 2008]. Simulation is an attractive alternative for evaluating novel grid scheduling algorithms [Quétier and Cappello, 2005] since it mimics the real grid as closely as possible. As defined by the workflow scheduling research problems (Research problems RP6 and RP7), this thesis proposes novel
grid workflow scheduling algorithms. A dynamic grid environment needs to be simulated in order to evaluate the proposed algorithms. So, the identified requirement is:

R5. A dynamic grid environment is required to be simulated.

2.2.3. Requirement for handling multiple workflows

Some of contemporary approaches assume the execution of a single workflow at a time in a grid environment (see e.g., [Sakellariou et al., 2007; Yu, 2007; Yu and Shi, 2010]). However, there can be multiple workflows (Research Problem RP2) in execution at a time in real grid environment. Subsection 2.1.5 discussed the three possible architectures of a grid workflow scheduler, namely: centralized, hierarchical and decentralized. The centralized grid workflow schedulers utilize one of the following three methods for handling multiple workflows.

1. **Backfilling**: In this method the workflows are scheduled using a static scheduling algorithm or through advance reservation. First, a single workflow is scheduled in the grid environment. Second, the left over time slots of the grid resources are filled by scheduling further workflows (see e.g., [Sulistio et al., 2006]). The tasks of the workflows are then executed according to the prepared schedule.

2. **Sequential workflow execution**: Multiple workflows can be executed in a grid environment one after another in the sequential workflow execution method. However only one workflow executes at a time.

3. **Merging**: In this method two or more workflows without changing their dependency constrains to form a large workflow [Zhao and Sakellariou, 2006; Sulistio et al., 2006]. The large workflow is then scheduled and executed in a grid environment.

The backfilling method can only be used in conjunction with a static scheduling algorithm or under the assumption that all the grid resources provide advance reservation support. However, the backfilling method cannot be used in case of dynamic scheduling and not necessarily all the grid resources provide advance reservation. In case of the sequential workflow execution a workflow must wait for the completion of the workflows which arrived before it in the grid environment. This may yield in longer wait time for the workflows to start their execution. In the merging method two or more workflows are merged together and scheduled as a single workflow. This method can be used for small workflows (having
few tasks) and less number of workflows. As discussed in subsection 2.1.5, the information required to perform the scheduling decisions (i.e., the task-to-resource mappings) increases non-linearly as the number of workflows or the number of tasks in the workflows increase. Moreover, the centralized and the hierarchical grid workflow schedulers suffer from the single point of failure problem as described in subsection 2.1.5. Therefore, a decentralized architecture of the grid workflow scheduler is required for handling multiple workflows in a dynamic grid environment. Therefore, the identified requirement is:

**R6.** The grid workflow schedulers having the decentralized architecture are required for handling multiple workflows.

### 2.2.4. Requirement for handling LUs’ load

Subsection 2.1.7 described the information about the grid resources required by the grid workflow scheduler. As shown in Figure 1-1, the grid workflow scheduler acquires this information from the GIS. However, since the grid resources belong to different administrative domains and they are subjected to site-specific policies, some of the information available at the GIS may be missing, incomplete, or outdated. E.g., the LUs’ load at the grid resources may vary over the time and the site-specific policies may not allow the sharing of the information about the LUs’ load with the GIS. Therefore, the grid workflow scheduler must estimate the information about the LUs’ load at the grid resources (Research Problem RP3). As discussed in subsection 1.2.2, the LUs’ load at a grid resource is a function of the number of tasks submitted by the LUs to the grid resource and the tasks’ lengths. Therefore, the following requirements are identified on the grid workflow scheduler (Requirement R6):

**R7.** A grid workflow scheduler must estimate the number of tasks at a grid resource submitted by the LUs on each grid resource.

**R8.** A grid workflow scheduler must estimate the tasks’ length of the tasks submitted by the LUs on each grid resource.

### 2.2.5. Requirements for handling queue wait time

Usually, the grid workflow scheduling approaches assume that a task of a workflow scheduled to a grid resource begins its execution immediately (see e.g., [Yu, 2007;
Sakellariou et al., 2007]). However, this is an unrealistic assumption because the tasks of the workflows have to wait in a grid resource’s queue due to the competing tasks of the workflows (cf. Subsection 2.2.3) and the LUs (cf. Subsection 2.2.4). The queue wait time is the time spent by the tasks of the workflows while waiting in a grid resource’s queue. The information about the queue wait time is required by the grid workflow scheduler while taking the scheduling decisions. Therefore, the grid workflow scheduler must estimate the queue wait time of a task of the workflow on a grid resource (Research Problem RP4). Therefore, the following requirement is identified on the grid workflow scheduler (Requirement R6):

**R9.** A grid workflow scheduler must estimate the queue wait time of each task of a workflow on each grid resource.

### 2.2.6. Requirements of the grid workflow scheduling algorithms

Subsection 2.1.6 listed three types of grid workflow scheduling algorithms found in contemporary literature, namely: static, dynamic and hybrid. The static grid workflow scheduling algorithms assume the computation time of the tasks of a workflow and the communication time between the tasks are known in advance. Similarly, the hybrid grid workflow scheduling algorithms are also based on similar assumptions. However, these assumptions are unrealistic for a dynamic grid environment. The dynamicity of a grid environment in terms of the varying LUs’ load at the grid resources requires a grid workflow scheduling algorithm. Such a grid workflow scheduling algorithm needs to be designed and implemented as a part of the grid workflow scheduler estimating the LUs’ load at the grid resources (Requirements R6). Therefore, the identified requirement is:

**R10.** A grid workflow scheduling algorithm is required, which utilizes the estimated LUs’ load at the grid resources with the scheduling objective to obtain a lower makespan.

In a dynamic grid environment the tasks of a workflow must wait in the grid resources’ queue due to the existence of the competing tasks (belonging to other workflows and/or the LUs). The longer queue wait time at a grid resource indicates that the grid resource is currently heavily loaded. So, a grid workflow scheduling algorithm needs to be designed and implemented as a part of the grid workflow scheduler. The proposed grid workflow
scheduling algorithm needs to utilize the estimated queue wait time on the grid resources (Requirements \textbf{R9}) for mapping the tasks of the workflows to the grid resources in order to balance the load at the grid resources thereby achieving good resource utilization. Nevertheless, the primary scheduling objective of the proposed grid workflow scheduling algorithm should still be to achieve a lower makespan. Hence, the proposed grid workflow scheduling algorithm must consider multiple (apparently conflicting) scheduling objectives. So, the identified requirement is:

\textbf{R11.} A grid workflow scheduling algorithm is required, which utilizes the estimated queue wait time on the grid resources with the scheduling objectives to obtain a lower makespan and good resource utilization.

\section*{2.2.7. Requirements for handling data movement time}

As discussed in subsection 2.1.6, in a dynamic scheduling algorithm the scheduling decisions to map the tasks of the workflows to the grid resources are postponed as long as possible. Therefore, the data produced by the parent tasks for the child tasks needs to be saved. This data is required for the child task its execution. A data saving mechanism is, therefore, required to save the data produced by the tasks of the workflows. This data saving mechanism needs to be integrated in the grid workflow scheduling algorithm in order to reduce the data movement time, which may help in achieving the scheduling objective of minimizing the makespan of a workflow. Therefore, the identified requirement on the grid workflow scheduling algorithms (Requirements \textbf{R10} and \textbf{R11}) is:

\textbf{R12.} A grid workflow scheduling algorithm must utilize a data saving mechanism in order to reduce the data movement time.
2.3. Summary of Requirements

The identified requirements in the previous section are summarized as follows:

**R1.** A workflow-based workload model containing the workflows with a diverse structure is required.

**R2.** The workflow-based workload model must model the static characteristics of the contained workflows.

**R3.** The workflow-based workload model must model the arrival time of the contained workflows in a grid environment.

**R4.** The workflow-based workload model must model the makespan of the contained workflows in a grid environment.

**R5.** A dynamic grid environment is required to be simulated.

**R6.** The grid workflow schedulers having the decentralized architecture are required for handling multiple workflows.

**R7.** A grid workflow scheduler must estimate the number of tasks at a grid resource submitted by the LUs on each grid resource.

**R8.** A grid workflow scheduler must estimate the tasks’ length of the tasks submitted by the LUs on each grid resource.

**R9.** A grid workflow scheduler must estimate the queue wait time of each task of a workflow on each grid resource.

**R10.** A grid workflow scheduling algorithm is required, which utilizes the estimated LUs’ load at the grid resources with the scheduling objective to obtain a lower makespan.

**R11.** A grid workflow scheduling algorithm is required, which utilizes the estimated queue wait time on the grid resources with the scheduling objectives to obtain a lower makespan and good resource utilization.

**R12.** A grid workflow scheduling algorithm must utilize a data saving mechanism in order to reduce the data movement time.
3. State of the Art

This chapter outlines the most relevant approaches to the research questions (RP1-RP7) presented in section 1.3. Section 3.1 presents an overview of some contemporary researches regarding grid workflow scheduling. Section 3.1.2 analyses previous researches towards the workflow scheduling in dynamic grid environment in the light of the identified requirements (Requirements R1-R12, cf. Section 2.3). The research deficits in previous research work will also be discussed in this chapter.

3.1. Workflow Scheduling in Grid Environment

This thesis attempts to answer the research questions namely, workflow-based workload modeling and multiple workflow scheduling on the grid resources in the presence of their LUs. This section presents a brief overview on previous research in the area of grid workflow scheduling. The previous researches are keenly examined for the research areas as shown in Figure 3-1.

---

**Figure 3-1: Key area of research**
3.1.1. Workflow-based workload modeling

Parallel computers’ workloads are extensively studied and remained the hot research topics for the computer scientists. With the development of grid environments their workloads are also intensively investigated (examples of such studies are [Song et al., 2004; Song et al., 2005; Mudnic et al., 2011; Iosup and Epema, 2011; da Silva and Glatard, 2013]). Some available grid workloads are the logs from different grid resources and examples of such workloads are [DAS-2, 2012; Grid5000, 2012] etc. The grid researchers can use these available workloads for the evaluation of their new developed algorithms (like scheduling, load balancing, and fault-tolerance). However, the newly developed algorithms need to tune a certain parameter and study its impact on others. In fact, the real grid traces does not allow such tuning if they are used directly. So workload modeling is required. As far as workflows are concerned, there is no workflow-based workload model present (which contain all the information required to test the workflow scheduling algorithms e.g., the structural information of workflows), moreover, the presence of workflows in the existing workloads can be determined [Iosup and Epema, 2011; da Silva and Glatard, 2013]. So it is highly recommended to present a workflow-based workload model for grid environments. In the absence of information regarding workflows in workloads, the only way is to present a synthetic workflow-based workload model.

Although a workflow-based workload model is not present, which provide the modeling of static and dynamic characteristics but the test benches of workflows are available which can be used to generate the workflows e.g., [Kwok and Ahmad, 1999a; Hönig and Schiffmann, 2004; Hönig, 2007; STG, 2012]. The workflow test benches suites are analyzed under the requirement identified in subsection 2.2.1 (Requirements R1-R4). The comparison of these test benches in the light of identified requirements are given in Table 3-1.

<table>
<thead>
<tr>
<th>Researchers/Studies</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kwok and Ahmad, 1999a]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[STG, 2012]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Hönig and Schiffmann, 2004; Hönig, 2007]</td>
<td>+</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Legend: a requirement is supported (+), is not supported (-), is supported to some extent (o)
### 3.1.2. Workflow scheduling in dynamic grid environment

The grid workflow scheduling is an N-P complete problem as explained in 2.1.4. Many researchers attempted to find the solutions of this problem using heuristic and meta-heuristic approaches, e.g., [Yu et al., 2005; Yu and Buyya, 2005b; Yu and Buyya, 2006; Yu and Buyya, 2005a; Yu et al., 2007; Sakellariou and Zhao, 2004; Afzal et al., 2006] and many more. Some studies solving the problem of grid workflow scheduling and prediction-based scheduling are discussed next.

[Topcuouglu et al., 2002] presented the static scheduling algorithms named as the Heterogeneous Earliest Finish Time First (HEFT) and the Critical Path on a Processor (CPOP). These algorithms prioritize the tasks of a workflow on the basis of their rank values, and schedule the tasks of the workflow on the grid resources where the tasks can complete in minimum time. However, these scheduling algorithms do not consider the dynamicity of the grid environment. In the CPOP algorithm the priority is assigned to the tasks on the critical path. The tasks having the rank value equal to the entry node are considered as the critical paths tasks. However, this way of critical path may fail as in reality since there may be some tasks having rank values equal to the entry task but they are not the critical path tasks. The algorithms presented in [Topcuouglu et al., 2002] do not consider any data saving mechanism to save the data produced during the execution of the workflows.

[Wiriyaprasit and Muangsin, 2004] presents the priority scheduling algorithms for sequential jobs. It is shown that the grid jobs (sequential jobs in this case) submitted to the grid resources suffer from longer wait times in the resources’ queues because the grid resources prioritize the LUs’ jobs. [Wiriyaprasit and Muangsin, 2004] considered the non-dedicated nature of the grid resources, however their scheduling approach considers only sequential jobs.

[Afzal et al., 2006] presented the architecture of a grid workflow scheduler as a queuing network. However, they neither considered the LUs’ load at the grid resources nor the data movement time of workflows. Moreover, only one workflow is considered for scheduling at a time, whereas in a real grid environment several workflows may compete for the grid resources.
Two different workflow scheduling approaches are implemented and evaluated using the randomly generated workflows by [Sakellariou et al., 2007]. This study, like the other conventional scheduling algorithms, assumes fully dedicated grid resources. This assumption does not hold in real grid environments. Considering the non-availability of grid resources for testing purposes, the authors of [Sakellariou et al., 2007] used simulation to evaluate their scheduling algorithms. They also considered the execution of a single workflow at a time. They did not handle the data movement time.

Yu and Shi presented a scheduling algorithm along with the system design [Yu and Shi, 2007]. In [Yu and Shi, 2007], initially the workflow tasks are scheduled statically but rescheduling decisions are made dynamically. However, neither the issue of multiple workflows nor the data movement is addressed.

[Yu, 2007] solved the grid workflow scheduling problem with the budget and deadline constraints, but she always consider that the grid resources are idle, i.e., having no LUs’ load is present on them. Moreover, [Yu, 2007] used the workflows from a particular application for the evaluation of the proposed scheduling algorithms. The research outcomes of these algorithms may not hold for other workload (i.e., the workflows from other applications or from a test bench). A central scheduler is used which handles only one workflow at a time to overcome the complexity. The handling of multiple workflow and issues related to multiple workflows scheduling is still missing in [Yu, 2007]. Due to the static scheduling approaches proposed in [Yu, 2007], the data movement among the tasks of the workflow is not efficiently handled.

[Wu et al., 2007] presented a queue model for grid scheduling. In [Wu et al., 2007], the authors assumed that the grid resources operate with different service rate. Each resource’s service time follows an exponential distribution. There is only one queue, which receives the jobs from the grid users, and all the grid resources can access this queue. The authors evaluated the system by simulating only three grid resources. Also, the characteristics of jobs are not defined by the authors.

In [Ranjan et al., 2008], the authors proposed a decentralized grid workflow scheduling algorithm, where different schedulers share schedule information. However, the authors of [Ranjan et al., 2008] do not consider the grid resources’ load fluctuation due to their LUs.
The problem of handling multiple workflows is solved by implementing a decentralized grid scheduler but the prediction of the queue wait time at the grid resources is still missing. The authors proposed a scheduling algorithm but they did not explain the handling of data movement between the dependent tasks of the workflows.

In [Baghban and Rahmani, 2008], the authors proposed a decentralized scheduling architecture but the decentralized schedulers do not share information regarding the schedule. The designed scheduler has the necessary parts needed to schedule the jobs, however the load fluctuation of the resources is not considered. It is recommended that the decentralized schedulers which are not sharing the schedule information have a grid resource queue predictor module to be aware of other competing users (workflow application owners) present in a grid environment but this module is missing [Baghban and Rahmani, 2008]. Moreover this study only considers the sequential jobs.

[Yu and Shi, 2010] proposed a queue wait time aware grid workflow scheduling model with tunable processors demand. This is a novel approach and it handles dynamicity at the end of processors demand. However, the data movement time is not intelligently handled, as suggested by the reported results. This approach has a potential of producing significant results in terms of the workflows’ makespan if the data movement time is reduced in their algorithm.

[Xi et al., 2011] predicted the queue wait time of the grid resources of the Open System Grid (OSG). The authors used a time series method to predict the queue wait time and scheduled the tasks of the workflows by creating multiple replicas on different grid resources. The author used the job rejection ratio and the makespan to evaluate the performance of their proposed scheduling scenarios. However, the results are not compared with any of the existing workflow scheduling algorithms which do not provide insight of the significance of the proposed algorithm. Moreover, in this study it is considered that at a time a single workflow is in execution, whereas in real grid environment multiple workflows may execute concurrently. The replication-based scheduling algorithm creates the replicas of the same task on different grid resources and the earliest completion of the task is considered, thereby discarding the other replicas. Clearly, this method utilizes the grid resources poorly, since the resources assigned to the replicated tasks can be potentially utilized by some other tasks.
Arabnejad presented an overview of the list-based workflow scheduling algorithms including the CPOP, the HEFT, the Heterogeneous Critical Parents Tree (HCPT), the High Performance Task Scheduling (HPS), the Performance Effective Task Scheduling (PETS) and the Lookahead [Arabnejad, 2012]. None of these workflow scheduling algorithms considers the LUs and the queue wait time at the grid resources. Moreover, at a time only one workflow is considered by these scheduling algorithms.

[Salinas et al., 2012] presented a prediction model to predict the load at the grid resources and the availability of the grid resources. The prediction model periodically gets the information of the grid resources’ load and forecasts the load at the grid resources for a future time interval. The predicted information is used by a scheduling algorithm to map the tasks on the grid resources. The performance of the prediction-based scheduling algorithm is compared with the random scheduling algorithm. The random scheduling algorithm randomly selects a resource and maps a task on the selected grid resource. The scheduling algorithm presented in [Salinas et al., 2012] is limited to the independent tasks.

[Tao et al., 2013] presented a reliability prediction model to predict the failure of grid resources during the execution of a task of a workflow. A grid workflow scheduling algorithm is presented which uses the predicted reliability cost of the grid resources and takes the workflow’s task to resource mapping decision such that the tasks execute without failure. The authors used a sequential data movement mechanism. However, in real grid environments a grid resource may receive the data from many grid resources.

[Kavitha and Sankaranarayanan, 2013] presented a load prediction method based on the double exponential smoothing. The results of the study show that the jobs placement decisions utilize the load predictions in order to increase the successful completion rate of jobs. However, the jobs considered in this study are not the workflows.

[Ghorbannia Delavar and Aryan, 2013] presented the Heurisitic Scheduling based on the Genetic Algorithm (HSGA) for scheduling workflows in the large scale distributed computing environments like grids and clouds. The authors showed that the HSGA algorithm produced the lowest makespan as compared to other scheduling algorithms used for comparison. However, this study is based on the assumption that the grid resources are fully available for the workflow execution, which is not realistic.
Most of the research studies do not consider the dynamic grid environments’ characteristics, like the grid resources’ LUs, multiple competing workflows’ tasks, and the data movement time as explained in 2.1.2-2.1.10. Moreover, the conventional workflow scheduling algorithms use the static methods of scheduling, where the entire workflow schedule is determined before it starts the execution as discussed in 2.1.6. The static scheduling algorithms cannot cope with the dynamicity of the grid environment. The dynamicity involved in grid is mostly due to the grid resources’ load fluctuations and/or the grid resources’ failure.

The dynamic scheduling algorithms which can use the prediction model to predict the future state of the system and cope with the system changes seamlessly are best to solve the grid workflow scheduling problem. The scope of this thesis is limited in sense that it only deals the load fluctuations of the grid resources and it is assumed that the grid resources do not fail. The load at a grid resource consists of 1) the tasks from the LUs and 2) the tasks submitted by multiple competing workflow schedulers. So, the aim of this thesis is to design, develop and test the proposed grid workflow scheduling algorithms, which can cope with the dynamic nature of grids.

Table 3-2 summarizes the analyzed studies according to the identified requirements (Requirements R6-R12; cf. section 2.3).

<table>
<thead>
<tr>
<th>Researchers/study</th>
<th>R6</th>
<th>R7</th>
<th>R8</th>
<th>R9</th>
<th>R10,R11</th>
<th>R12</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Topcuouglu et al., 2002]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Wiriyaprasit and Muangs, 2004]</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Afzal et al., 2006]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Yu, 2007]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Sakellariou et al., 2007]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Yu and Shi, 2007]</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Wu et al., 2007]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Baghban and Rahmani, 2008]</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Ranjan et al., 2008]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Yu and Shi, 2010]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Xi et al., 2011]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Arabnejad, 2012]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Salinas et al., 2012]</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Tao et al., 2013]</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Kavitha and Sankaranarayanan, 2013]</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Ghorbannia Delavar and Aryan, 2013]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Legend: a requirement is supported (+), not supported (-), is supported to some extent (o)
3.2. Summary

This chapter analyzed contemporary research approaches of grid workflow-based workload modeling and workflow scheduling with respect to the requirements identified in chapter 2, and identified the deficits in the existing research approaches. Some research approaches provide support for individual requirements. Nevertheless, no research is found which supports all the requirements. Specifically, the grid workflow scheduling algorithms considering the fluctuating load at the grid resources and the queue wait time at the grid resources due to their LUs as well as multiple competing workflows are missing. As discussed in this chapter, a decentralized grid workflow scheduler is required for evaluating new grid workflow scheduling algorithms which use prediction models to cope with the dynamicity of grid environment.
4. Grid Simulation Tools and Workflow Test Bench Suites

In subsection 2.2.2 the requirement (Requirement R5) for a simulated grid environment is identified. Such simulated grid environment is required to test the novel grid workflow scheduling algorithms. Developing a simulator to provide a simulated grid environment for the evaluation of grid workflow scheduling algorithms from scratch is a difficult as the designing and development cost are comparable to building the actual system [Sulistio et al., 2004]. Therefore, it is necessary to resort to one of existing simulators for this purpose.

In sections 4.1 the essential parts of a grid simulator required for a grid workflow scheduling are discussed. The section 4.2 identifies the requirements for a grid simulator. A survey of the existing grid simulators are presented in section 4.3. The authenticity and scalability of the grid simulators are explained in section 4.4. Finally, the selections of a grid simulator and a test bench suite for workflows generation are described in sections 4.5 and 4.6 respectively.

4.1. Essential Grid Abstractions for Grid workflow Scheduling

When a workflow application uses a grid for execution, the tasks are executed on the computational resources of the grid and the data files generated by the parent tasks are saved on some storage resources and then transferred to the resources where the child tasks have to start their execution. Therefore, along with the computing resources the file storage and transfer are two important factors. Hence, to test a workflow scheduling strategy a simulation environment with abstraction for computational, data and network resources should be selected. For the ease of analysis some simulator provide the automatic file generation, which contains the measured metrics like makespan or performance of a particular resource etc. So it can be concluded that the following are the basics entities for a simulator used to test the grid workflow scheduling strategies.

**Resources:** Resources in the grid can be single computers, multiprocessor machines, clusters, and even supercomputers. These resources may be geographically dispersed,
belong to different administrative domains and time zones. In case of workflows, the resources are also used to save the data files so the resources have some storage capacities.

**Grid information service (GIS):** GIS maintains resources’ characteristics, like workload of the resources, the processing capacity of resources, etc. For resource discovery, the grid resources’ information is delivered by the GIS to a grid resource broker [Xie et al., 2006].

**Data locator (DL):** The grid data locator is used to keep track that which file (generated by a parent task) is saved on which resource. This information is later used by the grid workflow scheduler while scheduling a child task.

**Grid resource broker (GRB):** GRB takes the information about resources from GIS and submits the workflow’ tasks to a resource fulfilling the user’s requirements like the deadline to complete a workflow, budget affordable to execute a workflow and Quality of Service (QoS) parameters. The GRB provides an interface between user and grid resources, thereby hiding the underlying complexities of the grid environment from user [Afgan, 2004].

**Grid user tasks:** The grid users of a grid submit their jobs (which are essentially workflows in this case) to the GRB. These workflows submitted by a grid user use the different part of a grid like GRB, DL etc., in order to execute on the grid resources.

**Network:** When the parents and child tasks of a workflow are schedule at different resources of the grid. In these cases, the files generated by parent tasks need to be transfer to the child tasks resources by mean of network. So the network is the most essential part of the grid in case of workflow scheduling. It is used for communication between the grid entities like user, GRB, GIS resources etc.

### 4.2. Requirements for the Grid Simulators

Following requirements (keeping in mind the above essential grid entities) are identified for an ideal grid simulator for this research.

**SR1.** It should be able to simulate any arbitrary number of heterogeneous resources ranging from supercomputers, clusters, personal computers to personal digital assistants. These resources may have a variety of resource characteristics (e.g., processing speed, storage space, workload patterns etc.). This requirement deals
directly with the simulated algorithm, some algorithms may need small number of resources and some algorithms may need a large number of resources with a variety of types. Similarly, an algorithm may need to be evaluated under specific workload pattern of resources.

SR2. It should provide abstractions to simulate resources dispersed across the globe with different time zones. These resources may be managed by different administrative domains with their specific local policies (e.g., policies regarding scheduling and load balancing). The facility to simulate the resources with different time zones helps the researcher developing a scheduler for grid applications. As different time zones mean, that the resources have different peak load and off load times.

SR3. It should be able to simulate various grid users with a variety of behaviors. The users can submit their jobs (which are workflows in this case) with specific requirements e.g. deadline, budget and/or QoS parameters.

SR4. The GRB should arrange the Virtual Organization (VO) of available resources to execute the submitted jobs. GRB should also facilitate the grid user to submit his jobs and receive the results without knowing technical details of the VO. The simulator should facilitate plugging-in new algorithms regarding scheduling, rescheduling load-balancing etc., so that new techniques can be tested.

SR5. The resources in the VO should also execute the jobs, which are dependent on each other. This may involve file transfer between the resources executing dependent jobs. The file transfer in real grid is done through the network with variable configurations, so the simulator must be able to provide network abstractions (e.g., routers, switches, message packetization, etc.).

SR6. To facilitate statistical study of the features like execution time of a particular job on different resources or load patterns of a particular resource at different times, the simulator must support report generation.

SR7. The results produced by a simulation experiment must be stable enough so that when repeated on another grid test environment (e.g., another simulator or a real
grid test bed), should be the similar (with an assumption that basic assumptions of the experiments were the same).

SR8. The grid simulator must be able to simulate a workload. This workload may be synthetic or/and a real workload trace.

4.3. A Survey of Grid Simulators

Sulistio et al. [Sulistio et al., 2004] surveyed the design issues of grid simulators Bricks [Takefusa et al., 1999], SimGrid [Casanova, 2001], GridSim [Buyya and Murshed, 2002; Caminero et al., 2007; Sulistio et al., 2008; Sulistio et al., 2007]. Quétier and Cappello [Quétier and Cappello, 2005] surveyed the grid simulators GangSim, OptorSim, Bricks, SimGrid and GridSim. They also discuss the advantages and limitations of simulators, emulators and real grid platforms.

In this section the survey of grid simulators is extended by including other simulators like GSSIM [Krzysztof et al., 2007], Alea [Klusácek et al., 2007] and GES [Vanmechelen et al., 2008]. The motivation for this survey is to facilitate the researchers choosing an appropriate grid simulator (and directly using it or extending it to fulfill their needs as [Flahive et al., 2006; Krzysztof et al., 2007; Quétier and Cappello, 2005] did). More specifically, the survey presented here analyses the grid simulators to select a suitable grid simulator to use it for studying grid workflow scheduling algorithms in a dynamic grid environment. Also, the authenticity, scalability and failure modeling of simulators are studied.

The approach of survey first discusses the simulators main components. Secondly, compare them with respect to the supported grid types and grid algorithms. Moreover, comment on the authenticity of results achieved by simulators, their scalability and their failure models are presented. In last, the characteristics of the discussed simulators and their support to the identified requirements (SR1-SR8) are summarized.

4.3.1. Bricks

Bricks [Takefusa et al., 1999] was designed to test scheduling algorithms with dead line constraints on computational grids. Bricks was also used to perform experiments for the network weather service (NWS). It is implemented in the Java programming language [Sulistio et al., 2004]. Bricks follow client server architecture. Major components of Bricks are
Network Monitor, Server Monitor, ResourceDB (Resource Database), Predictor, Scheduler, Server and Client [Takefusa et al., 1999].

The **Network Monitor** component measures the network bandwidth and latency of the grid. The measured values are stored into the **ResourceDB**. The **Server Monitor** component measures performance, load and availability of server machines. The measured values are also stored in the ResourceDB. The ResourceDB values are then used by the predictor and the scheduler components for the scheduling decisions and forecasts. The information about resources at a particular time is retrieved from the ResourceDB, and the **Predictor** component uses this information to predict the availability of the servers. The **Scheduler** takes the information from the Predictor and the ResourceDB to schedule the job to a suitable server.

### 4.3.2. GangSim

GangSim [Dumitrescu and Foster, 2005] is a policy driven grid simulator, inheriting the features of the Ganglia monitoring toolkit for VO. It is used for testing grid scheduling algorithms [Quétier and Cappello, 2005]. The components of GangSim are Sites, External Scheduler (ES), Data Scheduler (DS), Local Scheduler (LS), Monitoring Data Points (MDP), Site Policy Enforcement Point (S-PEP) and VO Policy Enforcement Points (V-PEP) [Dumitrescu and Foster, 2005; Quétier and Cappello, 2005].

Each **Site** is a collection of resources. A Site is characterized by its processing capabilities, space to save data and network configuration. A Site’s policy specifies the amount of processing power and space allowed to be used by each VO. Submitted jobs of a user are grouped together to form workloads. ES, LS, and DS components take part in scheduling a job.

At the **ES** component, the user’s jobs are queued and each job is sent to the best suitable Site selected as per the scheduling policy. At the Site the job is placed in its **LS** component. The **MDP** component gathers data from LS and DS components filters the data and represents them in a uniform manner. There are two categories of the **Policy enforcement points (PEP)** namely; **S-PEP** and **V-PEP**. Each Site has a specific **S-PEP**. The jobs coming to a site are continuously checked at the **S-PEP** and a job is only allowed to execute on the Site if it conforms to the Site policy otherwise it is not started at the Site. The **V-PEP** component
works exactly similar to the S-PEP component, but it works on the VO level. It enforces the user’s policy at job planning time and at the time of selecting jobs for scheduling.

The simulation results of GangSim are compared [Dumitrescu and Foster, 2005] with a real grid test bed called “Grid3”. According to Dumitrescu and Foster [Dumitrescu and Foster, 2005], the simulation results do not match the results of Grid3. The reason according to authors is that GangSim is unable to simulate some of the features of Grid3. For more information about Grid3, the reader can refer to [Emmen, 2004].

4.3.3. OptorSim

OptorSim was designed for simulating the data grid to test the different data replication strategies [H. Bell et al., 2003; Cameron et al., 2003]. The basic components of the OptorSim are sites, Resource Broker (RB), Replica Manager (RM), Replica Optimization Agent and Network links simulator [Cameron et al., 2003]. The detail about its components is described as follows:

**Sites:** The Sites are collections of different types of resources. Each resource can be either a computational resource called Computing Element (CE), or a data resource called Storage Element (SE).

**RB:** The resource broker schedules the user’s job to an appropriate CE.

**RM:** Each Site has a RM component. It provides an interface between CEs and SEs and the Grid for data flow.

**Replica optimization agent:** As a part of the RM, the replica optimization agent is responsible for selection of a replica and automatically deleting a replica when it is not needed anymore.

**Network links:** All the Sites having no CEs and SEs act as network links. Every network link has a certain bandwidth and latency.

---

1 The prototype Grid, called Grid3, uses the Internet to combine the computational resources of 26 universities and national laboratories across the U.S. to serve the computing needs of more than 10 research groups, from the fields of particle physics, astrophysics, bioinformatics and computer science.
Currently [Nicholson, 2008], only one CE per Site is simulated, therefore at a time only one job can be executed on a Site. Executing more than one job on a Site is still considered as the future work.

4.3.4. SimGrid

SimGrid was developed using the ‘C’ language. It provides the core functionalities to build a simulator. Most simulators built on the top of SimGrid were used to test grid scheduling strategies [Quétier and Cappello, 2005]. The first simulator built using SimGrid was PSTSim [Casanova, 2001]. PSTSim is designed to test scheduling algorithms for parametric sweep applications [Casanova et al., 2000] on the computational grid.

Three versions of SimGrid simulator have been released so far. The first version (SimGrid V.1) is unable to simulate network errors, software overheads, batch resource behavior and complex network topologies [Casanova, 2001]. In SimGrid V.2 abstractions for network communication are added [Legrand et al., 2003]. In the latest version (SimGrid V.3) the simulation engine is changed to enhance the scalability and performance of the simulator [SimGrid, 2011].

The two main components of SimGrid V1.0 are Resources and Tasks. Resources are defined by their name and performance metrics. A Task is defined by its name, cost and state. The states of a Task may be ‘not scheduled’, ‘scheduled’, ‘ready’, ‘running’ or ‘complete’ [Casanova, 2001]. In SimGrid V.2 the abstraction models are enhanced [Legrand et al., 2003] from two simple components to five core components namely: Agent, Location, Task, Path and Channel.

Agent: An Agent is an entity, which makes scheduling decisions.

Location: The Location [Legrand et al., 2003] in SimGrid V.2 is same as the Resources [Casanova, 2001] in SimGrid V.1 with some enhanced features (e.g., a Location abstraction is capable of simulating communication over a network link).

Task: A Task is an activity of the simulated application. It can be a computational task or a data transfer task.

Path: The Path abstraction simulates the physical links between the communicating Locations.
Channel: A channel is an abstraction to simulate the communication ports.

There is an API in SimGrid V.3 called ‘SimDag’ to simulate directed acyclic graphs. Another API called ‘GRAS’ is designed for simulating distributed applications. Currently an API for Message Passing Interface (MPI) called SMPI is under development [SimGrid, 2011]. For a detail study of SimGrid APIs the interested reader is referred to [SimGrid, 2011].

4.3.5. GridSim

GridSim is a Java-based grid simulation toolkit, designed to evaluate resource management and scheduling strategies for large scale distributed computing [Buyya and Murshed, 2002]. GridSim is used to simulate the Computational [Buyya and Murshed, 2002], Data [Sulistio et al., 2008] and Semantic Grid [Flahive et al., 2006]. The first version of GridSim (GridSim V. 1.0) was released in 2002. GridSim V. 1.0 was used to evaluate the grid broker Nimrod/G [Buyya et al., 2000], but in [Buyya and Murshed, 2002] Buyya and Murshed claimed that it can be used to simulate other distributed systems like Peer to Peer networks (P2P) as well. The GridSim toolkit was gradually developed to be a fully featured grid simulator. The evolution of GridSim from V. 1.0 to V. 5.0 (latest) [GridSim, 2012] is shown in Table 4.1.

<table>
<thead>
<tr>
<th>Old Version</th>
<th>New Version</th>
<th>Changes Made</th>
</tr>
</thead>
<tbody>
<tr>
<td>GridSim V1.0</td>
<td>GridSim V2.0</td>
<td>1. Separated the GridSim and grid broker APIs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Visual modeler component was added.</td>
</tr>
<tr>
<td>GridSim V2.0</td>
<td>GridSim V3.0</td>
<td>1. Job migration can now be simulated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Abstraction for the advance reservation of resources was added.</td>
</tr>
<tr>
<td>GridSim V3.0</td>
<td>GridSim V4.1</td>
<td>1. Network abstractions were added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. Abstraction for the failure modeling was added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Multiple GIS can be simulated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. Auction models were added.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. Data grid abstractions were added.</td>
</tr>
<tr>
<td>GridSim V4.1</td>
<td>GridSim V5.0</td>
<td>1. The SimJava is changed to pause and resume the simulation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2. A graphical user interface (GUI) graphical user interface for debugging resource allocation policies.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3. Scheduling policies for scheduling the parallel tasks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4. The workload model by Lublin and Feitelson for generating tasks for grid resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5. A data structure to facilitate the scheduling of tasks and advance reservations.</td>
</tr>
</tbody>
</table>
GridSim provides comprehensive abstractions for simulating different classes of heterogeneous computational, data and network resources, users, applications, resource brokers, schedulers and GIS. The basic components of GridSim are Resource, GIS, Network, Grid Resource Broker (GRB), User and Job. Next, the features of each entity of GridSim are discussed in detail.

**Resource:** A Resource in the GridSim can be a computational or a data resource.

- GridSim allows modeling the resources with different resource characteristics, types and time zones [Buyya and Murshed, 2002].
- GridSim enables the simulation of the resources with the workload traces taken from real grid resources [Sulistio et al., 2007].
- The resources can schedule the tasks according to either a space shared or a time shared scheduling policy [Buyya and Murshed, 2002].
- GridSim provides abstractions for the resource failure models [Caminero et al., 2007].
- Advance reservation of resources [Sulistio and Buyya, 2004] can be simulated here.

**GRB:** A GRB schedules the User’s tasks according to the implemented scheduling policy. Researchers can plug-in different scheduling algorithms here.

- GRB supports a reservation-based mechanism for resource allocation [Sulistio and Buyya, 2004].
- It provides clear and well defined interfaces for implementing different resource allocation algorithms.

**GIS:** A GIS has all the information about the resource like which resource is empty, number of free processor at a grid resource, the available memory space at the grid resource etc.

- There can be a single global GIS or multiple regional GIS components.
- When there are more than one GIS then they cooperate with each other by providing the information about the resources registered with them.
User: A User is the owner of tasks or jobs (referred as workflow in this thesis). A User may pay a specific amount of budget in $ to the GRB to execute his job [Buyya and Murshed, 2002] (such a behavior can be termed as economical behavior). A user without dealing with the budget issues can also be simulated (non-economical behavior). The user can assign a deadline to execute his Job. Thus GridSim allows us to model a variety of User behaviors (e.g., economical, non-economical, deadline constrained and without deadline). Task: Each Job is identified by a unique Identification Number (ID). A task has a specific length, number of processors required, input and output data file sizes. The length of a task is represented in millions of instruction “MI” [Buyya and Murshed, 2002]. A task’s history may be recorded. After the end of a simulation experiment, the recorded history of all tasks is reported. From V. 2.0 onwards, a task can also be preempted and migrated between resources during its execution.

Network: GridSim provides extensive network abstractions. It allows the simulation of grid environments with different topological configuration and various network components like routers and links [Sulistio et al., 2007]. In GridSim the network with background network traffic based on a probabilistic distribution can be simulated. This is useful for simulating the data-intensive tasks over a public network where the network is congested [Sulistio et al., 2007].

4.3.6. Alea

Alea is a GridSim-based grid simulator. The basic simulation class in Alea inherits the functionality of the GridSim’s core class called GridSim. Since GridSim V. 3.0 did not provide abstractions for simulating the network, therefore Alea extended GridSim for simulating the network topologies as well. Alea was developed to test the scheduling algorithm and study the effect of network topologies on grid scheduling.

The components of Alea are Job Submission System, Job Generator, Scheduler, and the Resources [Klusácek et al., 2007]. The Job Submission System consists of a Job generator and a Job submission module. The Job Generator component creates the tasks to be scheduled. The Job Submission System sends the tasks description to the Scheduler. The Scheduler component consists of three parts. The first part is responsible to handle the communication between the Scheduler and the Job Submission System. The second part is a centralized
information repository to store the information like workload on a Resource, Idle Resource List and job related information. The third and the last part consist of the scheduling algorithm, to be defined for an experiment. The scheduler maps the jobs to the appropriate Resources.

4.3.7. Grid scheduling simulator (GSSIM)

GSSIM extends the GridSim grid simulator. The basic motivation behind the development of GSSIM was to use real grid workloads as well synthetic workloads to study the behavior of grids under various scheduling algorithm at the GRB level and the resource level. The components of GSSIM are Workload Generator, Resources, Grid Scheduler and Local Scheduler [Krzysztof et al., 2007].

Workload generator: In GSSIM the workloads can be taken from the real grid resources or can be generated synthetically by the Workload Generator component. The workload contains all information about jobs: their structure, dependencies and resource requirements.

Resource: A Resource in GSSIM is defined by its computational and network characteristics (like the number of processors, processing power, support for advance reservation, the network bandwidth etc.).

Grid scheduler: The Grid Scheduler is responsible to map the jobs to appropriate Resources. Different scheduling algorithms can be simulated by modifying the Grid Scheduler.

Local scheduler: This component handles the job scheduling at the local administrative domain at the Resource level. There are two types of local scheduling mechanism provided in GSSIM namely: scheduling with or without QoS guarantee.

The major achievement of GSSIM is that it for the first time separated the grid scheduling into two levels: GRB level and Resource level, and now multiple scheduling strategies can be plugged-in at both levels. Another extension of GridSim (provided by GSSIM), is the abstractions for input data modeling (workload generation) [Krzysztof et al., 2007]. Moreover, GSSIM provides the resource failure models.
4.3.8. Grid economy simulator (GES)

GES is a single-threaded discrete-time-based grid simulator [Depoorter et al., 2008]. GES was developed to study economic grid resource management systems [Vanmechelen et al., 2008] and to compare them with the traditional grid resource management algorithms (e.g., without considering the economic factors).

A layered architecture is used to develop the GES in order to support the reusability and extendibility features [Vanmechelen et al., 2008]. The Domain Layer contains the Consumer, Provider, job and Resource entities. The economic layer of the simulator deals with the economic issues like billing, budget etc. The Bank entity also lies in the Economic Layer. The traditional resource managements without considering the economic issues are handled in the non-Economic Layer. There are two types of Consumers in GES, an Economic Consumer which deals with the economic job metrics (e.g., budget at hand to execute a job etc.) along with the traditional job metrics (e.g., job execution time, job length etc.) and a simple Consumer which only deals with the traditional job metrics. A Provider provides the resources to form a VO [Vanmechelen et al., 2008].

Advance reservation of the Resources can be made through the Future Market layer. The Auctions Layer deals with the auctions protocols. New auctions protocols can be plugged in at this layer. At the Tendering Layer, new negotiation strategies can be plugged in. Simulations can be distributed over multiple processing nodes through its distribution layer. The GUI layer of GES allows the user to create, run and monitor live market scenarios by setting the job, consumer, provider and resource metrics. The GRB (called the Central Controller here) controls the negotiation process, auctioning, billing etc.

Currently, GES provides no network abstractions. It is claimed in [Vanmechelen et al., 2008] that it is planned to provide the abstractions for the distributed execution of the individual simulation entities in the next release of the GES simulator.

4.4. Authenticity and Scalability of the Grid Simulators

Section 4.2 identified the requirements SR1–SR8 that an ideal grid simulator should support for testing the grid workflow scheduling. The requirements SR1, SR2, SR3, and SR5 deal with
the scalability of a simulator and SR7 states that the simulation results may be authenticated.

In the light of these requirements, the authenticity, scalability and supported failure models of the studied simulators (cf. section 4.4.1-4.4.8) are discussed in the following subsections.

4.4.1. Authenticity

As discussed in section 4.1, the need for grid simulators arises due to one or more of the following reasons:

1. Unavailability or at least of limited number of available real grid test beds.
2. Using real grid test beds (if available) for testing novel grid algorithms is very costly and time consuming.
3. A controlled environment is needed for testing some algorithms (e.g., in a scheduling algorithm a resource may be needed without any internal or external work load).
4. Repetition of experiments is almost impossible in a real grid.

There are two ways to authenticate the results acquired from a simulation namely; repeating the same experiment on a real grid test bed or repeating the experiment using another simulator. If an experiment can be repeated on a real grid, the simulation becomes worthless. Because, there is no need of simulation if a real grid test bed is available fulfilling all the testing requirements (e.g., controlled resources, repetition of same grid scenario etc.). Hence, the authentication of the simulation results by repeating the experiment on a real grid test bed is difficult and is never done [Legrand and Quinson, 2006].

The second method to authenticate the simulation results is to re-simulate the same experiment on a different simulator and compare the achieved results. Since different simulators were developed for different purposes using different programming languages and development tools, one can argue that their simulation results may not match due to some constrains of the simulators (like a simulator may be unable to simulate particular abstraction). E.g., OptorSim is used to test the replica optimization techniques and GridSim is used for testing the scheduling algorithms, data replication strategies, etc. therefore, a scheduling algorithm tested using GridSim cannot be simulated in OptorSim due to its
limitation of computational resource abstractions (cf. section 4.3.2). So the authentication by comparing the results achieved by simulating the same algorithm on different simulators is also difficult and is never done.

There is no regulatory authority enforcing the grid researchers to follow the standards while developing a grid simulator [Legrand et al., 2003]. In fact, there exist no such standards as well. Some studies developed the grid simulators only to test their algorithms with one or two grid scenarios, these types of simulators are called “Ad-hoc” or “on demand” simulators. These simulators are never used again in research and not available publically. So these types of simulators are also termed as “throw away” simulators [Legrand et al., 2003]. The results achieved by such simulators are very difficult or impossible to reproduce.

GangSim is the only studied simulator, whose results of simulation are authenticated by repeating the experiment on real grid test bed Grid3. But even here the simulation results deviate from the real grid test bed [Dumitrescu and Foster, 2005].

4.4.2. Scalability

The scalability of a software system is defined as its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged. There are two parts of this definition: the first part refers to scalability of the work while the second part refers to the scalability of the system. In case of grid simulators, the first part of the definition measures the number and type of grid abstractions (resources, user, GIS, GRB and grid jobs) can be simulated, and the second part measures the ease of extension of a grid simulator to accommodate new requirements.

The scalability of grid simulators is only discussed by [Depoorter et al., 2008]. In this study GES, GridSim and SimGrid are compared and GES is shown to be more scalable than the rest. This study partly satisfies the first part of the definition because it only focuses on the number of grid entities. However, GridSim and SimGrid are multi-threaded simulators, where each entity of the grid is placed in a separate thread, whereas GES is a single-threaded grid simulator. GridSim is used for testing data grid strategies (e.g., replica optimization) computational grid strategies (e.g., fault tolerance, scheduling, load balancing etc.) along with the grid economy parameters (e.g., cost and deadline). Whereas in GES, limited types of resources can be simulated and network is not simulated.
GridSim and SimGrid are two mature simulators, which are widely used by researchers, and they have active development status. GridSim and SimGrid are extended by many grid simulators [Klusácek et al., 2007; Krzysztof et al., 2007; Sherwani et al., 2004] and [SimGrid, 2011; Casanova, 2001] respectively. Therefore, GridSim and SimGrid are scalable in view of the second part of the definition.

4.5. Selection of a Simulator for Grid Workflow Scheduling

Many attempts (cf. section 4.3) have been made to provide a general simulation framework. Some of the salient characteristics of the simulators are summarized Table 4-2 and Table 4-3. As shown in Table 4-4, only GangSim and GridSim simulators provide abstractions for data and computational resources. The GridSim also provides abstractions for network resources such as routers, switches etc. The GridSim provides packet level communication abstraction, whereas, the GangSim provides no explicit network abstractions.

<table>
<thead>
<tr>
<th>Simulators</th>
<th>Grid type Simulated</th>
<th>Latest version</th>
<th>Available for download</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bricks</td>
<td>Computational Grid</td>
<td>Not available</td>
<td>No</td>
</tr>
<tr>
<td>GangSim</td>
<td>Computational Grid and Data Grid</td>
<td>No release after first release</td>
<td>Yes</td>
</tr>
<tr>
<td>OptorSim</td>
<td>Data Grid</td>
<td>2.1</td>
<td>Yes</td>
</tr>
<tr>
<td>SimGrid</td>
<td>Computational Grid</td>
<td>3.2</td>
<td>Yes</td>
</tr>
<tr>
<td>GridSim</td>
<td>Computational Grid, Data Grid and Semantic grid</td>
<td>4.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Alea</td>
<td>Computational Grid</td>
<td>3.0</td>
<td>Yes</td>
</tr>
<tr>
<td>GSSIM</td>
<td>Computational Grid</td>
<td>No release after first release</td>
<td>Yes</td>
</tr>
<tr>
<td>GES</td>
<td>Computational Grid</td>
<td>No release after first release</td>
<td>No</td>
</tr>
</tbody>
</table>
4.5 Selection of a Simulator for Grid Workflow Scheduling

Table 4-3: Simulators characteristic summary (B)

<table>
<thead>
<tr>
<th>Simulators</th>
<th>Extended to new simulators</th>
<th>Results compared with real grid</th>
<th>Failure Model</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bricks</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Scheduling algorithms and NWS</td>
</tr>
<tr>
<td>GangSim</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Scheduling algorithms</td>
</tr>
<tr>
<td>OptorSim</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Data Replication strategies</td>
</tr>
<tr>
<td>SimGrid</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Scheduling algorithms</td>
</tr>
<tr>
<td>GridSim</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Scheduling Algorithms, Data Replication strategies, Fault tolerance, Auctioning</td>
</tr>
<tr>
<td>Alea</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Scheduling algorithms</td>
</tr>
<tr>
<td>GSSIM</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Scheduling algorithms</td>
</tr>
<tr>
<td>GES</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Auctions and Economy related algorithms of Grid</td>
</tr>
</tbody>
</table>

The studied simulators are analyzed with respect to the requirements SR1–SR8 (cf. section 4.2) in order to guide the selection decision. The outcome of this analysis is summarized in Table 4-4. The entries in the table contain 0 or 1. An entry having the value 0 means that the studied simulator does not fulfil the corresponding requirement. However an entry having the value 1 signifies that the requirement is fulfilled by the studied simulator. The workflow tasks may need a variety of computational resources for their execution. The resource abstractions available in GridSim allow modeling diverse types and characteristics of the resources (e.g., data resource, computational resource, resources that allows advance reservation, varying processing capacity of resources etc.). It is observed that using GridSim various types of resources (like clusters, multiprocessor and Shared Memory Processor (SMP)) could be simulated. These resources may lie in different time zones and may fail during the execution. GridSim provides relatively better abstractions for modeling the grid network. GridSim has built-in capabilities to record the results, analyzing them statistically and generating reports [Buyya and Murshed, 2002]. Moreover, GridSim is well documented, mature enough, heavily used and readily extendable. So the grid simulator GridSim is selected to evaluate the grid workflow scheduling algorithms by simulation as it fulfills almost all requirements discussed.
4.6. Workflow Test Bench Suite Selection

4.6.1. Workload modeling

The researchers use a real grid resource workloads traces in order to evaluate the performance of the scheduling algorithms. Generating a workload model having all the characteristics of a real workload trace seems to be simple and close to the real scenario. However, these have the following disadvantages.

1. A grid resource configuration may be different than the trace’s resource configuration.

2. A workload trace size i.e., number of jobs in a trace cannot be scaled easily.

3. The original trace may contain some information, which is misleading, e.g., a trace may also contain the information regarding those jobs, which were rejected by a resource or failed due to any other reason.

4. It is not possible to modify the values of a model parameters in case of modeling it directly according to a workload trace. So the researchers are unable to study the influence of an attribute on others by changing one and keeping other constant.

Also, there is no workload trace available which consist of only workflows. So it is necessary to model the workload composed of workflows.

4.6.2. Workflow composition

Traditionally, there are three main methods of workflow composition namely:

Table 4-4: Simulators against identified requirements (SR1-SR8)

<table>
<thead>
<tr>
<th>Simulators</th>
<th>SR1</th>
<th>SR2</th>
<th>SR3</th>
<th>SR4</th>
<th>SR5</th>
<th>SR6</th>
<th>SR7</th>
<th>SR8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bricks</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>GangSim</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OptorSim</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SimGrid</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GridSim</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Alea</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GSSIM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>GES</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Legend: a simulator requirement is supported (1) and not supported (0)
User directed composition: User-directed composition allows users to edit workflows directly through markup languages or graphical tools (e.g., Kepler [Altintas et al., 2004]).

Automatic composition: Automatic composition systems (e.g., Pegasus [Dong, 2007]) generate workflows for users automatically.

Application oriented composition: Some studies (e.g., [Pandey and Buyya, 2008]) used workflows constructed for a specific application like Gravitational Wave Observatory

4.6.3. Workflow test bench suites

The researchers in the field of grid workflow scheduling used different methods of workflow composition and modeling. However, they did not followed the standards about the number of tasks in a workflow, the communication weights of the edges between interdependent tasks, computation weights and number of workflows. In [Sakellariou and Zhao, 2004] used randomly generated workflows of having 50 to 100 tasks and [Pandey and Buyya, 2008] used workflow of particular application. The solutions provided by using above techniques provide optimal solutions not for all types of grid workflow application but to some particular application or type of workflows. So there must be a general test bench suite, which considers the criterion like number of workflows, number of tasks in a workflow, the communication weights of the edges between the interdependent tasks of a workflow or the computation weight of tasks. To provide a general workflow test bench suite the attempts are made in [Kwok and Ahmad, 1999a; Hönig and Schiffmann, 2004; Hönig, 2007; STG, 2012]. In [STG, 2012] a test bench for workflows with up to 5000 tasks is provided but these workflows do not take communication overhead into account. In [Kwok and Ahmad, 1999a] a performance study of 15 heuristic scheduling algorithms is presented, the number of workflows is much lower (≈350). The workflow test bench suites characterization on the basis of number of workflows and communication among the tasks of the workflow is shown in Table 4-5.

<table>
<thead>
<tr>
<th>Researchers</th>
<th>Numbers of workflows</th>
<th>Handled Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Kwok and Ahmad, 1999a]</td>
<td>350</td>
<td>Yes</td>
</tr>
<tr>
<td>[STG, 2012]</td>
<td>5000</td>
<td>No</td>
</tr>
<tr>
<td>[Hönig and Schiffmann, 2004; Hönig, 2007]</td>
<td>7200</td>
<td>Yes</td>
</tr>
</tbody>
</table>
The workflow composition methods discussed so far do not explicitly consider the classification as given in [Hönig and Schiffmann, 2004; Hönig, 2007]. Therefore, a comparable test bench for the evaluation of grid workflow scheduling algorithms is missing. So, HC-test bench is selected for the workflow generation in this research.

4.7. Summary

This chapter describes the essential abstractions required for simulating a grid environment, which can be utilized for evaluating the grid workflow scheduling algorithms. First, the requirements for a simulator, which can be used for simulating a grid environment, are identified. Second, a survey of the existing simulators like Bricks, OptorSim, GangSim, SimGrid, GridSim, Alea, GSSIM, and GES is presented. The considered simulators are analyzed with respect to the identified requirements. Specifically, the authenticity of results achieved by the simulator and their scalability is discussed. The GridSim simulator is selected for simulating a grid environment (Requirement R5), since it fulfils the requirements. Third, the need of a test bench, which can be used to generate the workflows for the evaluation of grid workflow scheduling algorithms, is highlighted. The existing test bench suites are analyzed and the HC-test bench is selected for the workflow generation in this thesis due to its salient features, e.g., the number of workflows, the number of tasks in the workflows, the dependencies between the tasks of the contained workflows etc.
5. **Workflow-based Workload Modeling**

This chapter presents a solution to the research problem of workflow-based workload modeling (Research Problem **RP1**; cf. section 1.3). Subsection 2.2.1 identified four requirements for a workflow-based workload model (Requirements **R1-R4**; cf. section 2.3). This chapter partially addresses the requirement (Requirement **R5**; cf. section 2.3) by providing the essential steps of creating a simulation environment. This chapter proposes a workflow-based workload model as a solution for addressing the four requirements. Figure 5-1 shows an overview of the solution presented in this chapter.

Since the required *Workflow-based Workload Model* must contain workflows having a diverse structure (Requirement **R1**), the *HC-test Bench* is selected a basis for the Workflow-based Workload Model proposed in this chapter. The proposed Workflow-based Workload Model is utilized to create a *Workflow-based Application Model* in the *GridSim* simulator, which can be used for evaluating the grid workflow scheduling algorithms.

This chapter is organized as follows: Section 5.1 presents a brief description of the HC-test bench. Section 5.2 provides a background of contemporary methods of workload modeling. Section 5.3 describes the Workflow-based Workload Model by utilizing the HC-test bench as a basis. Section 5.5 describes the GridSim simulator and utilizes it to create a simulated grid environment.
environment in section 5.6. Section 5.7 validates the requirements of workflow-based workload modeling (Requirements **R1-R4**). Section 5.8 concludes this chapter by presenting a summary.

## 5.1. HC-test Bench

As discussed in section 4.6, the HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007] is selected to be used as a basis for the workflow-based workload proposed in this thesis. This subsection briefly describes the HC-test bench. The directory tree (containing 7200 workflows) which represents the structure of the HC-test bench can be seen in [Hönig and Schiffmann, 2004]. Later on the HC-test bench was adapted to a new test bench [Hönig, 2007], which contains 9600 test cases (workflows) as shown in Figure 5-1.

![Figure 5-2: Structure of the HC-test bench adapted from Hönig and Schiffmann, 2004](image-url)
The test cases (workflows) contained in the test benches are generated synthetically with respect to different target environments (which are homogenous or heterogeneous) [Hönig, 2007]. The HC-test bench is particularly designed for the heterogeneous computing environments therefore it is named as HC-test bench. The test cases of HC-test bench are referred as workflows in this thesis. Workflows contained in HC-test bench have a diverse structure and are generated by varying the different parameters of the workflows (e.g., the number of tasks, the number of edges, the computational weight of the tasks, and the communication weight of the edges) as well as the target environments (e.g., the number of processors, the architecture). Therefore, the generated workflows contain the description of both these types of parameters. The workflows are categorized into different workflow classes as shown in the directory tree (Figure 5-2), where a parameter of the workflows is considered for categorization at each level of the directory tree. The total number of workflows in all the workflow classes at a level is shown on the right side in Figure 5-2. The parameters of workflows considered at each level in the directory tree are briefly described next. An interested reader is referred to [Hönig and Schiffmann, 2004; Hönig, 2007] for detailed discussions.

**Graph Size (GS)** denotes the number of tasks in a workflow. In the HC-test bench, the workflows are categorized into two classes, namely: the GS6_10 workflow class having the workflows with the number of tasks between 6 and 10, and the GS6_250 workflow class having the workflows with the number of tasks between 6 and 250. The tasks in a workflow are numbered. The task number start from 0 and it is incremented for each task in the workflow.

**Consistency** of a workflow is defined in terms of the computation weights calculated for different processors in the target environments. The workflows are categorized into two workflow classes: namely **Consistent** and **Inconsistent**.

**Number of Sons (NoS)** of a workflow denotes the extent to which the tasks are dependent upon each other in the workflow. The meshing degree of a workflow is used as a measure of the dependencies among the tasks by [Hönig and Schiffmann, 2004]. The NoS_Avg, NoS_High, NoS_Low, and NoS_Rand workflow classes categorize the workflows having the average, high, low, and random meshing degree respectively.
Edge Length (EL) is a measure of the distance between the dependent tasks. The workflow classes EL_Avg, EL_Long, EL_Short, and EL_Rand categorize the workflows having average, long, short, and random Edge Lengths respectively.

**Node Weight** denotes the computational weight of a task of a workflow, whereas the **Edge Weight** denotes the communication weight of an edge between two dependent tasks of a workflow. The **Node to Edge Weight** denotes the Computation to Communication Ratio (CCR). Usually only the CCR values are used for generating workflows, however in the HC-test bench the Node and Edge Weights are considered separately in order to produce more diversity in the structure of the workflows [Hönig and Schiffmann, 2004]. The Node and Edge Weights are categorized into **Heavy (H)**, **Light (L)** and **Random (R)**. Different classes of workflows are defined using various combinations of the categories of the Node and Edge Weights. E.g., the workflows categorized in the HNodeHEdge workflow class have tasks and edges with high Node and Edge Weights respectively.

The HC-test bench defines the Extended Task Graph (ETG) format and utilizes it for defining the generated workflows. Figure 5-3 shows an example workflow in the ETG format.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>HCTB2Model</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>13515075</td>
<td>15629733</td>
<td>16000388</td>
<td>18480854</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>12300463</td>
<td>12506786</td>
<td>14613230</td>
<td>14702539</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>15201</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13360733</td>
<td>13597178</td>
<td>14104744</td>
<td>17424592</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>15672</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>8947</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6837504</td>
<td>11714777</td>
<td>12867801</td>
<td>13392230</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>11071107</td>
<td>12128770</td>
<td>16405088</td>
<td>17789748</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>19716</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>16433</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7034805</td>
<td>13995355</td>
<td>15081948</td>
<td>16301030</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15393</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>11358137</td>
<td>11640235</td>
<td>15011805</td>
<td>17861526</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>14302</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5-3: A workflow in the ETG format**
As shown in Figure 5-3, the first row of the ETG format contains two values followed by a string. The first value represents the number of tasks in the workflow. The second value represents the number of processors of the considered target environment, which are considered while generating this workflow. The string specifies the version of the HC-test bench, which contains this workflow. The workflow shown in Figure 5-3 contains seven tasks. It is generated while considering a target environment having four processors and it belongs to the HC-test bench version 2. The rest of the workflow consists of two types of rows, namely: the task rows describing the tasks, and the edge rows describing the edges between the tasks. A task row shown in Figure 5-3 contains six values. The first value of a task row represents the task number, the following four values define the four different computational weight of the task on the four processors of the considered target environment, and the last value represents the number of edges towards the task (representing its dependencies). An edge row shown in Figure 5-3 contains two values, where the first value represents the task number of the parent task, the second value represents the communication weight of the data produced by the said parent. E.g., the task number 1 in the workflow shown in Figure 5-3 is described by a task row followed by an edge row. The task row shows the task number (i.e., 1), the computation weight of the task on the four processors (i.e., 12300463, 12506786, 14613230, and 14702539 respectively), and the number of edges towards it (i.e., 1). The task row of the task number 1 is followed by the edge row of the task. The edge row shows the task number of the parent task (i.e., 0), the communication weight between the parent task and the child task (i.e., 15201). The number of edge rows followed by the task row (corresponding to a task) equals the number of edges (towards the task). Therefore, there is one edge row corresponding to the task number 1 and two edge rows corresponding to the task number 2.

5.2. Workload Modeling

The amount of work done by a grid in a specific time span is referred as the workload of the grid. The workloads can be utilized for objectively evaluating the performance of grid environments. In the previous decades the characterization and modeling of the workloads of grid environments were extensively studied [Song et al., 2004; Song et al., 2005; Iosup and Epema, 2011; Mudnic et al., 2011]. The characteristics of a workload of a grid environment include the system utilization, the size of the workload, jobs’ submission pattern, job type
Chapter 5  Workflow-based Workload Modeling

(parallel, sequential, workflows bag of tasks etc.), runtime and wait times of the jobs, and the requirement of the jobs (memory, Input/output, and network requirements) [Iosup and Epema, 2011].

There have been different methods for generating workloads of grid environments [Hirales-Carbajal et al., 2010]. This section describes contemporary methods of generating workloads and selects a method for generating a workflow-based workload model in order to address the requirements of workflow-based workload modeling (Requirements R1-R4). This section is organized as follows: Subsection 5.2.1 describes the usage of workload traces from real grid environments for generating a synthetic workload. Subsection 5.2.2 describes the generation of a workload using mathematical models and concludes by listing the benefits of this approach.

5.2.1. Workload traces from real grid environments

The workload traces are generated using the job logs of the grid resources of the real grid environments. Such workload traces are shared at the workload sharing portals. A standard trace format is specified by a workload sharing portal and the trace format is followed for defining the workload traces available at the portal. The trace format specifies the structure of the workload trace in order to facilitate the understanding of the workload traces. Each line in a workload trace represents a job, and each column of a line represents a characteristic of the job (e.g., the job arrival time, the wait time of the job before starting its execution, the number of processors requested by the job, etc.).

The workload traces from real grid environments can be used for evaluating the grid scheduling algorithms in one of the following two methods:

1. using the workload traces directly, or

2. generating a synthetic workload by preparing a statistical summary of the characteristics of a workload trace and tuning the characteristics of the workload trace [Kotsis, 1997; Hirales-Carbajal et al., 2010]. Such a synthetic workload may be required for studying the effect of the tuned characteristics of the workload traces on the scheduling algorithm.
However, as observed by [Hirales-Carabajal et al., 2010; Iosup and Epema, 2011], there is no workflow-based workload trace from a real grid environment available, which can be directly used for evaluating the grid workflow scheduling algorithms. Also, the workload traces from real grid environments cannot be used for generating a synthetic workflow-based workload model because of the following reasons:

1. The workload traces from real grid environments also contain the jobs, which could not be completed. Therefore, the contained information can be misleading.

2. It is difficult to determine whether or not a job in a workload trace from a real grid environment is a workflow [Iosup and Epema, 2011].

Another approach to generate a workflow-based workload is to record the information about the workflows in a grid environment [Ostermann et al., 2008]. [da Silva and Glatard, 2013] analyzed the jobs of a real grid environment by recording the job logs of the middleware “gLite.” The authors recoded the workflows as well the other jobs and made them available in the Science Gateway Workload Archive. The workflow archive of the Science Gateway Workload Archive contains 2914 workflows. The size of workflows (in terms of the number of tasks in the workflows) ranges from between 100 and 500. Both of these approaches suffer from the disadvantages of using the workload traces directly from the real grid environments as listed above and discussed in 4.6.1. Moreover, the workflow-based workload does not consider a diverse structure of workflows because it is extracted from the workload trace from a specific real grid environment.

5.2.2. Mathematical models

Mathematical models can be used for generating synthetic workloads [Li et al., 2007]. E.g., the Probability Distribution Function (PDF) can be used for modeling different characteristics of workload [Feitelson, 2002]. Generating a workload using a mathematical model is useful because the properties of the mathematical model can be easily changed, which results in tuning the characteristics of the generated workload [Singh and Segall, 1982].

Therefore, a workflow-based workload can be generated using mathematical models due to the following reasons:
1. This method is flexible insofar as it allows the generation of a workflow-based workload having:
   a. a diverse characteristics of the workload (e.g., arrival times of workflows in a grid environment, the number of processors requested by the tasks of a workflow, the memory requirement of the tasks of a workflows etc.),
   b. a diverse structure of workflow as discussed in subsection 4.6.3.
2. Such a workload model can be used in a simulation-based experiment for evaluating the grid workflow scheduling algorithms.
3. Such a workload model can be easily tuned in order to study the effects of a grid workflow scheduling algorithm.

The following sections describe a workflow-based workload model.

### 5.3. Workflow-based Workload Modeling

As discussed in the previous section, there is no workflow-based workload available publicly, which can be used for the evaluation of the grid workflow scheduling algorithms. Therefore a workflow-based workload is required to be synthesized such that its characteristics resemble the characteristics of the workload traces from real grid environments. This section proposes a workflow-based workload model by modeling the characteristics of a workload consisting of workflows in order to address the requirements of workflow-based workload modeling (Requirements R1-R4).

This section is organized as follows: Subsection 5.3.1 describes the characteristics of a workflow-based workload and it culminates by listing the three steps for modeling these characteristics. Subsections 5.3.2, 5.3.3, and 5.3.4 follow these steps for modeling the characteristics of a workflow-based workload. Finally, subsection 5.3.5 summarizes the workflow-based workload modeling.

#### 5.3.1. Characteristics of a workflow-based workload

The goal of a synthetic workload model is to build mathematical models for generating the characteristics of the workload which resemble the characteristics exhibited in the workload traces of real grid environments [Hirales-Carbajal et al., 2010]. Iosup and Epema analyze the
important characteristics of the workload traces from the real grid environments spanning over seven years (i.e., from 2003-2010) [Iosup and Epema, 2011]. The authors categorize the characteristics of the workload traces into two main categories, namely: the *general workload characteristics* and the *general job characteristics*. The general workload characteristics include the system utilization, the workload size, and the job submission pattern. The general job characteristics include type of jobs, job runtime, memory requirements, input output requirements and the network requirements. The analysis of [Iosup and Epema, 2011] considers various types of jobs which exist in the workload traces from the real grid environments, including bags of tasks, workflows, and pilots. However, since this thesis aims at generating a workflow-based workload (Requirements R1–R4), an analysis of the workflow-related characteristics is required. Ostermann et al. analyze the characteristics of a workflow-based workload and divide them into two categories, i.e., intrinsic workflow characteristics (i.e., the workflow structure) and grid environment-related characteristics (including the makespan of the workflows) [Ostermann et al., 2008].

This subsection categorizes the characteristics of a workflow-based workload into two characteristics, namely the static characteristics and the dynamic characteristics. The static characteristics do not change over the time but the dynamic characteristics may change because of the dynamicity of the grid environment. Both these categories are explained next.

**Static characteristics**

The static characteristics of a workflow-based workload are dependent on the structure and the requirements of the workflows contained in the workload. Figure 5-4 presents an overview of the static characteristics.

As shown in Figure 5-4, the *Static Characteristics* of a workflow-based workload are classified into the *Workflows’ Requirements* and the *Workflows’ Structure*. The requirements of a workflow contained in a workflow-based workload can be defined in terms of:

- the *Number of Processors* required by the tasks of the workflow,
- the *Memory Requirement* of the tasks of the workflow, and
• the Resource Architecture (e.g., its OS and the hardware architecture) required to execute the tasks of the workflow.

As discussed subsection 2.1.3, the structure of a workflow contained in a workflow-based workload can be defined in terms of:

• the Number of Tasks in the workflow,

• the Computational Weights of the tasks,

• the Number of Edges between the tasks of the workflow, and
Dynamic characteristics

The dynamic characteristics of a workflow-based workload are dependent upon the processing capabilities and network characteristics of the grid resources in a grid environment. Since the grid resources’ capabilities and network characteristics vary over the time, the dynamic characteristics of a workflow-based workload may also change accordingly. Figure 5-5 presents an overview of the dynamic characteristics of a workflow-based workload, namely: Arrival Time, Computation Time, Data Movement Time, and Makespan of the workflows contained in the workload.

The dynamic characteristics of a workflow-based workload are described next:

- **Arrival time** of a workflow is defined as the time when a workflow enters in a grid environment. The arrival time of the workflows in a grid environment represents the users’ submission patterns. E.g., more workflows may be submitted on workdays than on the weekends or public holidays.

- **Computation Time** of a workflow is defined as the sum of the computation time of the tasks, where the computation time of a task is defined as the time spent in its execution on a grid resource.

- **Data Movement Time** of a workflow is defined as the sum of the data movement time of the tasks of the workflow, whereas the data movement time of a child task of
a workflow is defined in terms of the communication time between it and all its parent tasks.

- **Makespan** of a workflow is defined as the time elapsed between the start of the execution of the first task of the workflow and the end of the execution of the last task of the workflow. Makespan of a workflow is a function of its computation time and data movement time.

The previous discussion analyzed the characteristics of a workflow-based workload. The steps for modeling these characteristics are listed below:

1. Since a workflow-based workload must exhibit the users’ submission pattern as shown in the workload traces in the real grid environments, the workload size and the arrival time of workflows need to be modeled (as suggested by the Requirement R3). Subsection 5.3.2 presents the modeling of the arrival time of workflows.

2. The static characteristics are required to be modeled for each workflow in a workflow-based workload (as suggested by the Requirement R2). The workflows in a workflow-based workload must have a diverse structure (Requirement R1). Therefore, the HC-test bench is used as a basis for modeling the workflows. Subsection 5.3.3 models the static characteristics.

3. The dynamic characteristics of a workflow-based workload are required to be modeled. The computation time and the data movement time of a workflow are used for modeling the makespan of the workflow (Requirement R4). Subsection 5.3.4 models the makespan of the workflows.

**5.3.2. Modeling the arrival time of the workflows**

Jobs enter following a certain pattern in a real grid environment [Baghban and Rahmani, 2008; Ostermann et al., 2008; Iosup and Epema, 2011]. The arrival pattern of workflows can be generated in a synthetic workload by using different distributions [Feitelson, 2002]. Workflows may exhibit an arrival rate for an interval, where the length of the interval, denoted as $T'$, is defined as the length of the time period for which a workflow-based workload is being modeled. The arrival rate of the workflows in an interval of length $T$ can be used to model the arrival time of each workflow (Requirement R3). The arrival time of a
5.3 Workflow-based Workload Modeling

The workflow is referred as ‘A’. The time elapsed between the two consecutive workflows entering in a grid environment is known as inter-arrival time and it is denoted as ‘IA’. The IA time is also an important characteristic of a workflow-based workload because it shows the frequency of workflows entering in a grid environment.

According to [Li et al., 2007] the arrival of job in grid system can be modeled as a point process. This can be done by using a mathematical construct that represents individual events as random points at time $t_1, ... t_n$ in an interval of length $T$ as shown in Figure 5-6. The inter-arrival time $IA$, is the time difference between the arrival times of two consecutive jobs.

The arrival time of the workflows in a workload can be modeled by using a distribution (e.g., Normal distribution, Exponential distribution, and Poisson distribution) [Feitelson, 2002]. The Poisson process is selected for modeling the arrival times of workflows in an interval of length $T$ in the workflow–based workload model due to the following reasons.

1. The number of workflows arriving in an interval is always finite. The number of events in a Poisson process is also finite [Ross, 2006].

2. The arrival of the next workflow is independent from the arrival of previous workflows in an interval in the workflow-based workload. The next occurrence of an event in an interval is also independent of the current and past occurrence in a memory-less Poisson process [Ross, 2006].

3. Since the inter-occurrence times of the events occurring in a Poisson process are always exponentially distributed [Ross, 2006; Ng and Boon-Hee, 2008], modeling the
arrivals of workflows as the Poisson Process automatically leads to modeling the inter-arrival times as exponentially distributed.

As defined by [Ross, 2006], a counting process \( \{N(t), t > 0\} \) with a rate \( \lambda \), where \( N(t) \) is the number of events up to time \( t \) and \( \lambda > 0 \) is said to be Poisson process if:

1. \( N(0) = 0 \),
2. \( N(t) \) has stationary and independent increments,
3. \( P\{N(h) = 1\} = \lambda h + o(h) \), and
4. \( P\{N(h) \geq 2\} = o(h) \).

A function \( f \) is said to be \( o(h) \) if \( \lim_{h \to 0} \frac{f(h)}{h} = 0 \).

Alternatively, A stochastic process is said to be a Poisson process with a certain rate \( \lambda \) [Ross, 2006], where \( \lambda > 0 \) if:

1. \( N(0) = 0 \),
2. \( N(t) \) has independent increments, and
3. The number of events in any interval of length \( t \) is Poisson distribution with a rate of \( \lambda t \), also \( \forall s, t \geq 0 \) and is given as Equation 5-1.

\[
P\{N(t + s) - N(t) = n\} = \frac{e^{-\lambda t}(\lambda t)^n}{n!}
\]

Equation 5-1 gives the Probability Density Function (PDF) of the Poisson distribution, where the notations are explained below:

<table>
<thead>
<tr>
<th>Notations</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e )</td>
<td>Base of natural logarithm</td>
</tr>
<tr>
<td>( n )</td>
<td>Number of events in the interval ([0,t])</td>
</tr>
<tr>
<td>( n! )</td>
<td>( n! ) is the factorial of ( n )</td>
</tr>
</tbody>
</table>
5.3 Workflow-based Workload Modeling

\( \lambda \quad \text{A positive real number considered as the rate of events in an interval} \)

The proof that the above definitions are equivalent can be found in [Ross, 2006]. These definitions describe a homogeneous Poisson process with a fixed arrival rate of \( \lambda \).

Since the arrival rate of workflows in a grid is not fixed, the peak time arrival rate of workflows is different from the off time arrival rate. So the homogeneous Poisson process is unable to model the arrival of workflows. The systems where the event rate changes over the time can be modeled using a compound or non-homogeneous Poisson process [Rubinstein and Kroese, 2008]. A counting process is said to be a non-homogeneous Poisson process when:

1. \( N(0) = 0 \).
2. \( N(t) \) has independent increments.
3. The event rate in an interval \((t_1, t_2)\) is a function given in Equation 5-2.

\[
\lambda_{t_1,t_2} = \int_{t_1}^{t_2} \lambda(t)dt
\]  

Equation 5-2

4. The number of events in the interval \((t_1, t_2)\) has a Poisson distribution with a rate of \( \lambda_{t_1,t_2} \) are given in Equation 5-3.

\[
P\{N(t_2) - N(t_1) = n\} = \frac{e^{-\lambda_{t_1,t_2}}(\lambda_{t_1,t_2})^n}{n!}
\]  

Equation 5-3

The occurrence times of events in an interval \((t_1, t_2)\) in a non-homogeneous Poisson process is shown in Figure 5-7. In a non-homogeneous Poisson process the numbers of events in an interval are dependent on the rate \( \lambda \) as well as the time \( t \) as shown in Figure 5-7. Here the occurrence times of events are same as a homogeneous Poisson process with a rate of \( \lambda \) and the positions of occurrences are uniform in the interval \([0, \lambda]\). This suggests that each occurrence time of an event of a homogeneous Poisson process is rejected with a probability \( p \), where \( p \) is given in Equation 5-4.
In Equation 5-4 $t_n$ is the occurrence time of nth event. The occurrence times except the rejected occurrence times are desired occurrence times of the events, shown as white circle in Figure 5-7. The rejected occurrence times are shown as black circles in Figure 5-7.

Algorithm 5.1 models the arrival time of workflows for an interval as a non-homogeneous Poisson process. This means that the occurrence time of an event is the arrival time of a workflow.
As shown in Algorithm 5.1, the \texttt{poissonSimulation} algorithm takes two input parameters, namely: the interval length $T$ and the arrival rate $\lambda$ of the workflows in the interval. The required variables $t, n$, and $i$ are initialized first (Algorithm 5.1: line 2). The \texttt{WHILE} loop (lines 3-15) iterates until the value of $t$ is less than the interval length $T$. The loop generates the arrival time of the next workflow $t_n$. Next, the counter $i$ representing a pass of the \texttt{WHILE} loop is incremented (line 4). An independent random variable $U_i \approx U(0,1)$ is generated using the Uniform distribution in the interval $(0,1)$ (line 5). The next arrival time is calculated and assigned to the variable $t$ (line 6). The \texttt{IF} condition (lines 7-9) checks the value of the variable $t$ to determine whether or not it is still less than the interval length $T$. If the value of the variable $t$ exceeds the interval length $T$, the \texttt{WHILE} loop (of step number 3) is broken. Next, another independent random variable $V_i \approx U(0,1)$ is generated using the Uniform distribution in the interval $(0,1)$ (line 10). The \texttt{IF} condition (lines 11-14) rejects the arrival time stored in the variable $t$ when the value of the random variable $V_i$ is less than $\frac{\lambda(t)}{\lambda}$, this where the value of $\lambda(t)$ can be calculated by Equation 5-2, i.e., each arrival time is rejected with a probability $p$ given in Equation 5-4. The value of $n$ is incremented by 1 (line 12) and an arrival time $t_n$ is generated at (line 13), $n$ represent the number of event and $t_n$ is the time of occurrence of nth event. The series $\{ t_1, t_2, t_3 ... t_n \}$ represents the time of arrival of workflows.
In some situations, a certain number of workflows need to be tested to evaluate the effectiveness of a grid workflow scheduling algorithm. In this situation, the algorithm 5.2 can be used for the generation of arrival time of the workflows.

As shown in Algorithm 5.2, the poissonSimulation_2 algorithm takes two input parameters, namely: the numberOfWorkflows and the arrival rate $\lambda$ of the workflows. The required variables $t$, $n$, and $i$ are initialized first (Algorithm 5.2: line 2). The WHILE loop (lines 3-12) iterates until the value of variable $n$ is less than the numberOfWorkflows. The loop generates the arrival time of the next workflow $t_n$ . Next, the counter $i$ representing a pass of the WHILE loop is incremented (line 4). An independent random variable $U_i$ is generated using the Uniform distribution in the interval $(0,1)$ (line 5). The next arrival time is calculated and assigned to the variable $t$ (line 6). Next, another independent random variable $V_i$ is generated using the Uniform distribution in the interval $(0,1)$ (line 7). The IF condition (lines 8-12) rejects the arrival time stored in the variable $t$ when the value of the random variable $V_i$ is less than $\frac{\lambda(t)}{\lambda}$, this where the value of $\lambda(t)$ can be calculated by Equation 5-2, i.e., each arrival time is rejected with a probability $p$ given in Equation 5-4 . The value of $n$ is incremented by 1 (line 9) and an arrival time $t_n$ is generated at (line 10), $n$ represent the number of event and $t_n$ is the time of occurrence of nth event. Also the series $\{ t_1, t_2, t_3 \ldots t_n \}$ represent the arrival times of workflows.
5.3.3. Modeling the static characteristics

The static characteristics of each workflow contained in workflow-based workload are modeled in this section. The modeled characteristics include workflow number, workflow structure (i.e., number of tasks in the workflow, number of edges in the workflow, computational weights of the tasks of the workflow and the communication weights of the interdependent tasks of the workflows), and the requirement of each workflow (i.e., number of processors required by the tasks of the workflow and memory requirement of each task of the workflow). There can be many other static characteristics associated with a workflow like deadline and budget etc., but as the scope of thesis is limited to community grids therefore such characteristics are not considered here.

**Workflow Number** ($W$)

It is a natural number and act as a counter starting from 1. Workflow number is always incremented by 1 when a new workflow is added in the workflow-based workload. Workflow number is required to be serialized the jobs (which are workflows in this case) it is denoted by $W$.

**Workflow Path** ($WP$)

The purpose of providing a workflow-based workload is to use it for the evaluation of grid workflow scheduling algorithms. In order to avoid the biasness in the evaluation the workload model must contain the diverse structure workflows according to the requirement $R1$. To provide the diverse structure the HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007] is selected as it provides a verity of workflows classes based on the structure of the workflows as explained in section 4.6. Therefore, workflow-based workload utilizes the HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007]. Selection of the HC-test bench does not fulfill the requirement $R1$. In order to fulfill the requirement $R1$ i.e. the workload must contain the diverse structure, the workflow-based workload must contain the diverse structure’ workflows. Therefore for all arrival times of an interval the workflows are selected using a normal distribution. On this way, each workflow from 9600 workflows of the HC-test bench have an equal probability of being selected for an interval, hence the requirement $R1$ is full filled by providing the diverse structure workflows in a workflow-based workload. In the workflow-based workload only the path of the workflow is given, which is denoted as
The path of the workflow is the location where a workflow of the HC-test bench is actually saved e.g.

```
...\HC-Testbench\GS6-250\Consistent\NoSHigh\EL_Long\HNodeHEdge\uh0605021576.etg```

The above path of the workflow shows that the workflow named as “uh0605021576.etg” may have the number of tasks in the range of 6-250. Moreover, in this workflow the NoS is high, EL is Long and computation and communication weight is High.

In order to fulfill the requirement R2 the static characteristics of the workflow-based workload are modeled. Next the modeling of the static characteristics of the workflow-based workload is discussed.

**Number of tasks (N)**

As explained in section 5.1 the first row of each workflow in ETG format contains the information about the number of tasks in a workflow. E.g., in case of Figure 5-3 the number of tasks in the workflow is 7. The number of tasks in a workflow is denoted as ‘N’.

**Processor requested (P)**

The number of processors, which can be requested by the tasks of a workflow are denoted as ‘P’. For each workflow the number of processes which can be requested is selected randomly between the 1 and 4. The workflow scheduling algorithms studying the effect of tunable processor requirements can model the processor requirements accordingly.

**Resource Architecture (RA)**

A workflow may demand a specific architecture for the execution of the tasks of the workflow i.e., a specific OS, and hardware architecture of the grid resource, and grid resource scheduling policy. Resource architecture of in the workflow-based workload is denoted as ‘RA’. Currently the workflow-based workload specifies the different values for the different OS requirement and the demand of specific hardware is not modeled. The values \{1, 2, 3, 4, 5\} represent that the tasks of a workflow requires \{Linux, Unix, Windows, Google Chrome, does not matter\} Oss (i.e., if 1 then a workflow demands the Linux OS, if 2 the tasks of the workflow demands Unix and so on if the OS value is 5 this mean that the
workflows tasks can execute on any available resource regardless of its OS). Currently all the tasks of a workflow can demand only one type of OS.

**Number of Edges \( (E) \)**

As shown in Figure 5-3, last value in task row describes the number of dependencies of the task. E.g., in Figure 5-3 task number 0 has no dependency. The next row is regarding task 1. After the four entries last entry states that the number of parents of task 1 so it has one parent. All the last entries of the tasks rows are summed up and this value is considered as the number of edges denoted as \( E \). Figure 5-3 shows that \( E = 7 \).

**Computational weights \( (CW) \)**

Each task of the workflow has four computational weights. As the purpose of developing a workflow-based workload model is to use it for the testing of the grid workflow scheduling algorithms. Moreover, GridSim simulator is selected to test the workflow scheduling algorithms. The computational weight of a task in GridSim simulating environment can be defined as Millions of Instructions (MI). So from the four weights of each task the highest value is selected as the computational weight of the task e.g., in case of task 0 the computational weight is 18480854 MI. The computational weight of a task of a workflow is represented as \( CW \).

**Communication weights \( (CmW) \)**

The tasks of a workflow may depend on one or more than one other tasks of the workflows. The dependent tasks are said to be child tasks and tasks on which a task is dependent are said to be parent tasks. The communication weight between a parent task and a child task of a workflow is defined in Mega Bytes (MB). E.g., in case of task 1 it has 15201MB communication weight with task 0. The communication weight of a task is represented as \( CmW \).

**Memory required \( (Mm) \)**

The sum of the communication weights of the edges of a task in a workflow is defined as the memory required by the task. The memory required by a task of the workflow is denoted by \( Mm \).
5.3.4. Modeling the makespan of the workflows

The modeling of arrival times of workflows is already explained in subsection 5.3.2, the following subsection model the dynamic characteristics of a workflow-based workload. The computation time, data movement time and makespan of a workflow on a grid system can only be calculated while having the information about the processing capacities and network characteristics of the grid resources. The makespan which is required to be modeled according to requirement R4 cannot be modeled without having the information about the grid resources. Moreover as makespan of a workflow is a function of the computation time of the workflow and the data movement time of the workflow, therefore the computation time and data movement time also need to be modeled. The grid workflow scheduling is not a simple task this includes:

1. **Pre-execution steps**: The workflow tasks mapping to resources with certain objective optimization, data required by a task to start its execution is transferred to the resource where the task is mapped, dispatching the task to the mapped resource etc., are done in pre execution steps.

2. **Task execution**: The processors are assigned to the tasks of workflows for their execution.

3. **Post execution steps**: The post execution steps involve saving the data generated by the task for its dependent task, notifying all the dependent tasks for their parent completion, result gathering and sending to the corresponding user.

The objective need to be optimized in the pre execution step refers to makespan and the computation time of a workflow is referred as task execution and the post execution steps involves the data movement of a workflow. The makespan of a workflow is dependent upon the computation time of the tasks of a workflow as well as the data movement time of the workflow. Finding the optimal makespan of a workflow with given set of grid resources is well known N-P complete problem as explained in 2.1.4. Finding a solution to the N-P complete problems is reasonable time often referred as polynomial time is important. Therefore a makespan of a workflow which lies in the range of the lower bound of makespan and the upper bound of makespan can be referred as the optimal solution of the N-P complete problem found in the reasonable time subjected that both the lower bound of the
makespan and the upper bound of the makespan measureable and are finite real numbers. In some contemporary researches the researchers consider the lower bound of the makespan of a workflow as execution time of the longest path (i.e., critical path) of the workflow. However, in a workflow there may exists more than one critical path so for a workflow-based workload model the lower bound of the makespan is defined considering the computation time of the tasks of the workflow and the data movement time of the workflow. For reference points the lower and upper bound of the makespan are required therefore the lower and upper bound of the computation time of a workflow and data movement time of a workflow are also need to be modeled, as makespan is a function of computation time and data movement time of a workflow.

Grid Resource Configurations

Modeling the makespan of a workflow requires the configurations of the grid resources i.e. their processing capabilities and network characteristics. Let’s assume that there are total $M$ resources in a grid. The processing capacity of each resource is measured in Millions of Instruction per Second (MIPS) according to the Standard Performance Evaluation Corporation (SPEC) CPU (INT) 2000 [SPEC, 2012] benchmarks. It is assumed that the processing capacity of the grid resources is known. The processing capacity of a resource $R_k$ is denoted as $PC(R_k)$. The mean processing capacity the grid is given in Equation 5-5.

$$MPC = \frac{\sum_{k=1}^{M} PC(R_k)}{M}$$

\forall 1 \leq k \leq M \tag{Equation 5-5}

Also, the communication links’ capacities of resources can be calculated using the bandwidth and the latency of the links between the resources, it is assumed that all the grid resources are interconnected. If all the grid resources are interconnected, there are possibly $M \times M$ links. The $CC(R_k, R_l)$ shows the communication capacity of the link between the grid resources $R_k$ and $R_l$. The mean communication capacity of the grid can be calculated according to Equation 5-6.

$$MCC = \frac{\sum_{k=1}^{M} (\sum_{l=1}^{M} CC(R_k, R_l))}{M^2}$$

\forall 1 \leq k, l \leq M \text{ and } k \neq l \tag{Equation 5-6}
The slowest resource processing capacity can be referred as the minimum processing capacity of the grid. It is denoted as \( \text{MinPC} \) given in Equation 5-7.

\[
\text{MinPC} = \text{Min}(PC(R_k)) \quad \forall 1 \leq k \leq M
\]  

Equation 5-7

The sum of the processing powers of the grid resources is known to be the maximum processing capacity of the grid and it is denoted as \( \text{MaxPC} \) and is given in Equation 5-8.

\[
\text{MaxPC} = \sum_{k=1}^{M} PC(R_k) \quad \forall 1 \leq k \leq M
\]  

Equation 5-8

Similarly, the slowest link between the resources of the grid is referred as the minimum communication capacity and sum of all the links’ communication capacities is referred as the maximum communication capacity. The slowest communication capacity is denoted as \( \text{MinCC} \) and maximum communication capacity is referred as \( \text{MaxCC} \), these are given in Equation 5-9 and Equation 5-10 respectively.

\[
\text{MinCC} = \text{Min}(CC(R_k, R_l)) \quad \forall 1 \leq k, l \leq M \text{ and } k \neq l
\]  

Equation 5-9

\[
\text{MaxCC} = \sum_{k=1}^{M} \left( \sum_{l=1}^{M} CC(R_k, R_l) \right) \quad \forall 1 \leq k, l \leq M \text{ and } k \neq l
\]  

Equation 5-10

**Computation time of a workflow**

The computational weight of the workflow is the sum of all the computational weights of the tasks of the workflow. The mean, lower and upper bound of the computation time of a workflow can be obtained by dividing the weight of the workflow by the mean, maximum and minimum processing capacities of the grid. In a workflow having \( N \) number of tasks, \( \text{CW}(t_i) \) is the computation weight of the task \( t_i \) of the workflow. The computation weight of the workflow is denoted as \( \text{CWW} \) and is given in Equation 5-11.
The Lower Bound of the Computational time of a Workflow is LBCW and it is given in Equation 5-12.

\[
LBCW = \frac{CWW}{MaxPC}
\]  
Equation 5-12

The Mean Computation time of a Workflow is denoted as MCW and can be calculated according to the Equation 5-13.

\[
MCW = \frac{CWW}{MPC}
\]  
Equation 5-13

The Upper Bound of the Computational time of a Workflow is denoted as UBCW and is given in Equation 5-14.

\[
UBCW = \frac{CWW}{MinPC}
\]  
Equation 5-14

**Data movement time of a workflow**

Similarly, the communication weight of the workflow is the sum of all the communication weights of all the edges of the workflow. The mean, lower bound and upper bound of the data movement time of a workflow can be obtained by dividing the communication weights of the workflow by the mean, maximum and minimum communication capacities of the grid. A workflow having E number of edges and \(CmW(t_i, t_j)\) is the communication weight of the task \(t_i\) and \(t_j\) of the workflow. The communication weight of the workflow is denoted as CCWW and is given in Equation 5-15.
Chapter 5  Workflow-based Workload Modeling

\[ CCW = \sum_{i=1}^{N} CCW(t_i, t_j) \]  \hspace{1cm} \text{Equation 5-15}

\[ \forall 1 \leq i, j \leq N \]

The lower bound of the data movement time of a workflow is denoted as \( LBDMW \) and it is given in Equation 5-16.

\[ LBDMW = \frac{CCW}{MaxCC} \]  \hspace{1cm} \text{Equation 5-16}

The mean data movement time of a workflow is denoted as \( MDMW \) and can be calculated according to the Equation 5-17.

\[ MDMW = \frac{CCW}{MCC} \]  \hspace{1cm} \text{Equation 5-17}

The upper bound of the data movement time of a workflow is denoted as \( UBDMW \) and is given in Equation 5-18.

\[ UBDMW = \frac{CCW}{MinCC} \]  \hspace{1cm} \text{Equation 5-18}

**Makespan of a workflow**

Makespan of a workflow can be calculated as a function of computational time and the data movement time of the workflow. The lower bound of the makespan is the sum of lower bounds of computation time and lower bound of data movement time of the workflow. This is given in Equation 5-19.

\[ LBMSW = LBDMW + LBCW \]  \hspace{1cm} \text{Equation 5-19}

Like the LBMSW the upper bound of the makespan and the mean makespan of the workflow is calculated by using Equation 5-20 and Equation 5-21 respectively.

\[ UBMS = UBDMW + UBCW \]  \hspace{1cm} \text{Equation 5-20}
In a real grid environment, there are several issues affecting the makespan of a workflow e.g., the workflows tasks may wait in the grid resources queues, the workflow at a grid scheduler may wait its turn etc. Also, the in cases of workflow scheduling the tasks need to wait for their parents to complete, which adds to the makespan of the workflow. In the workflow-based workload modeling such issues are not considered. Therefore, the actual makespan may differ from the calculations.

5.3.5. Summary of workflow-based workload modeling

The subsection 5.3.1 analyzed the characteristics of a workflow-based workload and listed three steps of modeling these characteristics. Subsections 5.3.2-5.3.4 modeled the different characteristics of a workflow-based workload. Table 5.1 summarizes the characteristics modeled in the previous subsections.

\[ MMS = MCCW + MCW \]

---

<table>
<thead>
<tr>
<th>Explanation</th>
<th>Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow number</td>
<td>( W )</td>
</tr>
<tr>
<td>Arrival Time</td>
<td>( A )</td>
</tr>
<tr>
<td>Workflow’s path</td>
<td>( W )</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>( N )</td>
</tr>
<tr>
<td>Number of edges</td>
<td>( E )</td>
</tr>
<tr>
<td>Resource architecture</td>
<td>( RA )</td>
</tr>
<tr>
<td>Processor requested</td>
<td>( P )</td>
</tr>
<tr>
<td>Computation weight</td>
<td>( CW )</td>
</tr>
<tr>
<td>Communication weight</td>
<td>( CmW )</td>
</tr>
<tr>
<td>Memory required</td>
<td>( Mm )</td>
</tr>
<tr>
<td>Lower bound of Computational time of a workflow</td>
<td>( LBCW )</td>
</tr>
<tr>
<td>Lower bound of data movement time of a workflow</td>
<td>( LBDMW )</td>
</tr>
<tr>
<td>Lower bound of makespan of a workflow</td>
<td>( LBDMW )</td>
</tr>
<tr>
<td>Mean computational time of a workflow</td>
<td>( MCW )</td>
</tr>
<tr>
<td>Mean data movement time of a workflow</td>
<td>( MDMW )</td>
</tr>
<tr>
<td>Mean makespan of a workflow</td>
<td>( MMW )</td>
</tr>
<tr>
<td>Upper bound of computational time of a workflow</td>
<td>( UBCW )</td>
</tr>
<tr>
<td>Upper bound of data movement time of a workflow</td>
<td>( UBDMW )</td>
</tr>
<tr>
<td>Upper bound of makespan of a workflow</td>
<td>( UBMW )</td>
</tr>
</tbody>
</table>
5.4. Generating the Workflow-based Workload Model

The requirement R1 suggests the need of a workflow-based workload model containing the diverse workflows with respect to their structure in order to avoid the biasness while evaluating grid workflow scheduling algorithms. To provide the diverse structure the HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007] is selected since it provides a variety of workflows classes based on the structure of the workflows as explained in section 5.1. Therefore, the HC-test bench is used as a basis for the workflow-based workload modeling.

Conceptual overview

Figure 5-8 shows the conceptual overview of the workflow-based workload modeling. The Workflow-based Workload Generator utilizes the HC-test bench for generating the required Workflow-based Workload Model containing the static as well as dynamic characteristics of a workflow-based workload. Since the dynamic characteristics are dependent on a grid environment, the processing capabilities and the network characteristics of the grid resources need to be specified for the generating the dynamic characteristics.
As shown in Figure 5-8, the Workflow-based Workload Generator requires four configuration files, namely:

1. The Workflow-based Workload Configurations file is the main configuration file. The format of this file is described with an example in Appendix A.1. This file contains the interval length (in time units) of the workload or the number of workflows in the interval, the arrival rate of the workflows in the interval (workflows/unit of time), the path of the root directory of the HC-test bench, and the paths of the other three configuration files. The time unit is defined in terms of seconds here.

2. The Grid Resources’ Processing Capabilities file specifies the processing capabilities of the grid resources, which are required for generating the dynamic characteristics of the workflow-based workload. The format of this file is described with an example in Appendix A.2. This file contains the number of grid resources in the grid environment, and the following parameters for each grid resource:
   a. the number of processors in the grid resource,
   b. whether the grid resource is homogenous or heterogeneous,
   c. the processing speed of each PE (in MIPS) in the grid resource (in case of a heterogeneous grid resource),
   d. the processing speed of a PE (in MIPS) in the grid resource (in case of a homogenous grid resource), and
   e. the storage capacity of (in Terabytes)

3. The Grid Resources Links’ Bandwidth file specifies the number of resources and considering that each grid resource is connected to each other grid resource the bandwidth of the links (in Mb/S). The format of this file is described with an example in Appendix A.3

4. The Grid Resources Links’ Latency file specifies the number of resources and considering that each grid resource is connected to each other grid resource the latency of the links (in seconds). The format of this file is described with an example in Appendix A.3
The paths of all the required inputs are listed in a configuration file named as *Workflow-based Workload Configurations*. The *Workflow-based Workload Generator* first provided the path of this main configuration file and it parses the configurations’ file gets the paths all required files. Using this information the workflow-based workload model generates a workflow-based workload model and saves it. Different workflow-based workload model can be generated using different configurations of grid resources, the network characteristics arrival rate and interval length. A workflow-based workload model is generated for the *European (EU) Data Grid configurations as specified in* [Sulistio et al., 2008]. The generated workflow-based workload model contains forty-eight thousand workflows. The steps involved in workflow-based workload model generation include:

**Generating the arrival time of the workflows**

From the main configuration file the workflow-based workload generator get the values of interval length and arrival rate for the interval. Using the Algorithm 5.1 the arrival time $A$ of the workflows in the interval is modeled.

**Generating the static characteristics of the workflows**

From the main configuration file the directory path of the HC-test bench is obtained. For each arrival of a workflow in an interval a workflow is selected from the HC-test bench. The static characteristics of each workflow, including workflow number $W$, workflow path $WP$, the number of tasks in the workflow $N$, the number of edges in the workflow $E$, resource architecture $RA$, the number of processor required by the tasks of a workflow $P$, computational weight of the tasks of a workflow $CW$, communication weight of the edges of the workflow $CmW$, and memory required by each task of the workflow $Mm$.

**Calculating and saving the makespan of the workflows**

From the main configuration file the paths of files of *Grid Resources’ Processing Capabilities*, links bandwidths Grid Resources Links’ Bandwidth and *Grid Resources Links’ Latency are obtained*. By parsing these files the parameters like number of processors in a grid resource processing speed of each processor in MIPS the bandwidth of the links between the grid resources and the latency of the links between the grid resources is obtained. Using these parameters the makespan is calculated. Since a workflow’s makespan calculation requires
the data movement time as well the computation time of the workflow, therefore the data movement time and computation time are also calculate according to the models provided in subsection 5.3.4. As the makespan calculations require the lower bound mean and average of the makespan so is the case of data movement time and computation time. The values of lower bound of computation time of a workflow $LBCW$, lower bound of data movement time of a workflow $LBDMW$, lower bound of makespan $LBMW$, mean computation time of a workflow $MCW$, mean data movement time of a workflow $MDMW$, mean makespan $MMW$, upper bound of computation time $UBCW$, upper bound of data movement time $UBDMW$, and upper bound of makespan $UBMW$ for each workflow of the workflow-based workload model is generated.

**Format of the workflow-based workload**

The characteristics of a synthetic workload must bear a resemblance to the characteristics of the real grid workload [Hirales-Carbajal et al., 2010]. To provide such resemblance the workflow-based workload is modeled by following the Grid Workload Format (GWF) [GWF, 2012]. The following principles of the SWF are followed during the modeling of workflow-based workload.

1. The workload is stored in an ASCII file.

2. Each workflow is represented by a single row in the workload's file.

3. Rows contain a predefined number of fields and each field represents a specific characteristic of the workflow either static or dynamic.

4. Comments in a workflow-based workload are allowed begins and ends with `;`.

5. In a workflow-based workload each workflow is assigned a number and it is referred as workflow *Number*.

6. Each line of the workflow-based workload represents a workflow and each field of a line shows a static or dynamic characteristic of workload. The fields are separated by white spaces.
The fields’ names and their specified abbreviations of the workflow-based workload is provided in Table 5-1 and the schema of the workflow-based workload is shown Figure 5-9.

5.5. GridSim

As described in section 4.5, the GridSim simulator is selected to form as a basis for simulating a dynamic grid environment (Requirement R5) in this thesis. The GridSim is an open source simulation toolkit written in the Java programming language [Buyya and Murshed, 2002]. The GridSim toolkit allows modeling and simulation of the entities involved in a grid environment like users, grid resources, and resource schedulers (see subsection 4.3.5 for a detailed description of the GridSim entities). Figure 5-10 shows the layered architecture of the GridSim. These layers are described from bottom to top next.

As shown in Figure 5-10, the bottom layer of the GridSim’s layered architecture is the SimJava Simulation Kernel, which is an event driven simulation package implemented in Java programming language [Howell and Mcnab, 1998]. The SimJava Simulation Kernel layer manages the communication among the GridSim entities by message passing operations defined in SimJava2. The second layer models the core elements of the GridSim, including the grid Resource, Network, and Traffic Generator etc. The GridSim allows modeling a wide range of grid resources including single-processor or multi-processors with shared or distributed memory. The processing capacity of a grid resource is represented as Millions of Instructions per Second (MIPS). The Network is an important core element, since it provides a communication medium for other core elements. The Traffic Generator is another core element, which can be used for generating the traffic over the network in order to mimic the behavior of shared network links in a real grid environment spanning across the globe.
The next two higher layers are responsible for the simulation and management of the entities related to the Computational Grid and the Data Grid. These layers have some common services like the Grid Information Service (GIS) and the Job Management. The jobs require grid resources (i.e., compute resources, storage resources). The information about the available grid resources is maintained in the GIS repository. The Job Management service is responsible for the job submission, job monitoring and result gathering activities. The Computational Grid-specific entities include the Reservation, Resource Allocation services, and the Workload Traces entity. The computational grid resources are managed by the resource-level resource allocation policies, which are time or space shared [Buyya and Murshed, 2002]. Later on the Advance Reservation-based resource allocation policies were also included as services related to the Computational Grid [Buyya and Sulistio, 2008]. The Workload Traces are used in a trace-based simulation. Such traces can originate from a real grid or can be generated synthetically. Currently, the GridSim allows modeling of the Workload Traces related to a single grid resource\(^2\). The Data Grid-specific entities include:

\(^2\) Workload (GridSim 5.0 API Specification),
the *Data Set*, the *Replica Catalogue*, and the *Replica Manager*. The Data Set entity provides abstractions for modeling the data files. The Replica Catalogue (RC) is a repository, which maintains the information the availability of the Data Sets and the location of their replicas. The Replica Manager uses the RC for providing replica management services for jobs of the Data Grid.

The top most layers are used for defining *User Code* including the *Grid Scenario*, *User Requirements*, and *Application Configuration*, etc. The user can define the overall configuration of the grid environment in the Grid Scenario. The user can also define their own applications and requirements using the Application Configuration and User Requirements. The *Grid Scheduler* maps the users’ jobs to the appropriate grid resources according to the implemented grid scheduling algorithm. Next the essential steps to create a simulation environment in GridSim are described.

**Steps of creating a simulation in GridSim**

This section describes the creation of a grid simulation environment using GridSim. The focus of this simulation environment is to show the steps followed for the creation of a grid environment.

**GridSim initialization**

The GridSim simulation kernel, which is based on the SimJava2, is needed to be initialized, before creating any GridSim entity. The simulation kernel needs the information like number of users, calendar etc., for the initialization.

**Network topology creation**

The network resources (routers, switches etc.) and links’ capacities (baud rate, propagation delay) etc., are needed to be created in a network topology creation. The Network topology in GridSim can be built in two ways, during simulation or by a file. Further details can be found in [Sulistio et al., 2007].

**Grid resource creation**

Each resource in GridSim is associated with a resource characteristic object at its creation. The resource characteristics objet stores the properties of a grid resource like architecture,
OS, list of PEs, internal scheduling policy, time zone and its price. These properties help the grid scheduler to take a scheduling decision. Each resource contains a calendar object as well. This calendar object tells that whether a resource is available in holidays or not and its peak time and off times etc. Each grid resource may have some processors or memory units or both. The processing elements (PEs) of a grid resource may have same processing speeds or different. The processing speed of each processor is measured in Millions of Instructions per Second (MIPS). The link of the resource is created which connect the resource to one of the existing routers or to another grid resource as specified in the topology. Each grid resource is registered with the GIS.

Grid User

The grid users submit their jobs to the grid scheduler. A grid user receives the results of executed job from grid scheduler. When a grid user has no more jobs to execute it notifies the GIS. When all the users finish the execution of their jobs GIS notifies all the registered entities to shut down.

Grid Scheduler

The grid scheduler receives the jobs from the grid users. A grid scheduler discovers the grid resources with the help of GIS and shortlist the discovered resources according to the grid users’ requirement. The grid scheduler submits the jobs to the grid resources according to implemented scheduling algorithm. When a grid resource finishes the execution of a job, it sends the result to the grid scheduler. The grid scheduler sends the results of jobs to the corresponding grid users.

Start simulation

The last step building an experiment in GridSim is to invoke the simulation start method of GridSim. This is done through GridSim.startSimulation method.

GIS

GIS is a repository providing the information like the number of available grid resources, the number of processors in the grid resource, number of tasks on a grid resource etc., The GIS periodically updates this information about the registered resources with it. In GridSim the
GIS sends the finish simulation signal to all entities (i.e., grid resources, grid scheduler, routers etc.) when all the grid users finishes the execution of their workload. The shutdown command of GIS finishes the simulation.

5.6. Layered Architecture of a Simulated Grid Environment

Figure 5-11 shows a layered architecture of a simulated grid environment using the GridSim simulator. Figure 5-11 shows the SimJava Simulation Kernel Layer, GridSim is built on the top of SimJava Simulation Kernel as explained in section 5.5. The GridSim layer represents the core entities, which are necessary for the grid simulation environment for workflow scheduling. These entities include Network, Resources, GIS and Data Locator (DL). The workflow scheduling requires various types of resources e.g., computational resources to execute the tasks of the workflows, data resources to save the data generated by the parent tasks of a workflow for the child tasks. The Network is required as a medium of communication between different resources and other grid entities like GIS and DL. Moreover workflow scheduling in a simulated grid environment required the Network abstractions because the data (which is generated by the parent tasks of a workflow for their child tasks) are transferred between the grid resources. In Figure 5-11 GIS is a repository which provides the grid resources information to the grid workflow scheduler as explained in section 5.5. The Data Locator entity is used to check that at which resource the files (generated by the parent tasks of a workflow for their child tasks) are saved, and the availability of the memory space at the grid resources. The GridSim provides the RC entity to save the data files generated during a simulation experiment, but the RC creates the replicas of the files at different locations. The replicas are created so that the files can be used later on, however the RC and RC management policies are not the focus of this thesis. Therefor a Data Locator entity is created which only gives the information of the storage space in a grid resource. Data Locator is used to save the data file only at one grid resource (i.e., without replication). The Data Locator is also used to find the location (i.e., grid resource) of a saved data file.
5.6 Layered Architecture of a Simulated Grid Environment

Grid Scheduler Layer shows grid users submit the workflows to the Grid Workflow Scheduler and Grid Workflow Scheduler schedules the tasks of workflows according to implemented scheduling algorithm. Different workflow scheduling algorithms can be plugged into Grid Workflow Scheduler. The design of the Workflow Scheduler is discussed in the upcoming chapter.

GridSim does not provide a specific application model [Sulistio et al., 2008]. A workflow-based application model is required to test the workflow scheduling algorithms in the simulated grid environment. The unavailability of a workflow-based application model in GridSim requires the development of this model and its integration in GridSim. The design of the workflow-based application model is discussed in this section. The Workflow-based Application Model Layer shows that in a simulated grid environment the Grid User utilizes the Application Model to read the Workflow-based Workload Model and creates an Application Model consisting of workflows. The features of a Workflow-based Application Model are discussed in detail in upcoming subsection.

Class diagram of Workflow-based application model for GridSim

In this section the class diagram from Unified Modeling Language (UML) is used to explain the workflow-based application model for GridSim. Figure 5-12, depicts the class diagram of
the workflow-based application model and its interaction with the GridUser class. In GridSim a GridUser is considered as owner of the application, the GridUser shown in Figure 5-12 has an association link with the ApplicationModel class. The ApplicationModel class reads the workflows from the Workflow-based Workload Model and creates an ApplicationModel composed of Workflow. The GridUser later submits these workflows to the grid scheduler. The grid scheduler schedules the workflows according to implemented workflow scheduling algorithm and sends the results of a completed workflow to the GridUser. Next each class shown in Figure 5-12 is explained one by one.

Figure 5-12: Class diagram of the workflow-based application model for GridSim
5.6 Layered Architecture of a Simulated Grid Environment

**Edge Class**

An edge in a workflow shows the dependency constraint among two tasks of the workflow, where one of them is parent task and the dependent task is said to be child task. In the Edge class there are two attributes namely `parentGridletID` and `childGridletID`.

**Gridlet class**

GriSim provides the Gridlet class the object of Gridlet class can be used as an independent task having a specific length defined in MIs and a file or list of files as their input parameters and a file or a list of file as output.

**Workflow class**

The workflow class has the attributes as specified in the workflow-based workload model. This class provides the following methods to create a workflow object. Next the methods of the Workflow class are explained. The constructor of the class Workflow takes the `workflowNumber`, `numberOfGridlets`, a list of gridlets, and edges of the workflow. The `workflowNumber` is the number associated with each workflow in the workflow-based workload model. The `numberOfGridlets` is the number of tasks of the workflow. The `gridlets` is the list of Tasks of the workflow. The gridlet in GridSim represents an independent task, having a definite task length represented in Millions of Instructions (MI). The `communicationWeightOnEdge` attribute in the workflow class mentions a key-value Map, where the key is an Edge object presenting an edge and value is a long object representing the communication weight on the Edge. The workflow class has the getter methods for its private attributes. The `getEdges` method returns a list of key values of the Map `edges`. The method `getCommunicationWeightOnEdge` is parameterized by an Edge object and returns the communication weight on the edge if the Edge object exists in `edges` attribute of workflow class.

**ApplicationModel class**

The ApplicationModel class has a containment relationship with the workflow class this containment is shown as attribute workflows in the ApplicationModel class. The `workflows` attribute in the ApplicationModel class mentions a key-value Map, where the key is a `workflowNumber` a long value and a workflow object mapped against the `workflowNumber`. 
The method of `ReadWorkload` of the class `ApplicationModel` is assigned a path of a workload as input parameter. The `ReadWorkload` parses the workflow-based workload and saves the workflows. The method `getWorkflows` returns the `workflows` attribute of the class.

**GridUser class**

The `GridUser` class is inherited from the `GridSim` class. Every class inherited by `GridSim` has to override the method `body` which defines the behavior of the entity during the simulation. The `GridUser` class has an association relationship with the `ApplicationModel` class. The workflow can be submitted to a Grid Scheduler through the `submitWorkflow` method. The workflow results from `Grid Scheduler` are received by the `GridUser` through the `receiveResults` method.

### 5.7. Validating the Requirements of Workflow-based Workload Modeling

In this section, the results of the workflow-based workload model are presented and the requirements of the workflow-based workload model are validated in the light of results obtained. The results analysis consists of analyzing the following parameters of the workflow-based workload model:

- number of tasks of the workflows in the workflow-based workload model,
- number of edges of the workflows in the workflow-based workload model,
- percentage of the different classes of workflows from HC-test bench in the workflow-based workload,
- lower bound of the computation time, data movement time and makespan of the workflows in the workflow-based workload model,
- mean values of the computation time, data movement time and makespan of the workflows in the workflow-based workload model, and
- upper bound of the computation time, data movement time and makespan of the workflows in the workflow-based workload.
The workflow-based workload contains forty-eight thousands workflows. These workflows belong to the different classes of the workflows of the HC-test bench. The contained workflows in the workflow-based workload model are categorized according to the number of tasks in the workflows. The classes of workflows are named as follows:

1. The workflows belonging to Large ($L$) class have greater than 200 tasks,

2. The workflows belonging to Medium ($M$) class have the number of tasks greater than 100 but less than 200, and

3. The workflows belonging to Small ($S$) class have the number of tasks less than 100.

Figure 5-13 shows that that 12% of workflows in workload model contain more than 200 tasks and belongs to the large class of the workflows, 21% belong to the class of the medium class of workflows, and 67% of workflows belong to the small class of workflows containing 100 or fewer tasks. Therefore the workflow-based workload can said to be mixture of workflows large, medium and small.

![Figure 5-13: Workflow classes' percentage in Workflow-based workload model](image-url)
Figure 5-14 shows the Cumulative Distribution Function (CDF) of the number of tasks of the workflows contained in the workflow-based workload. In the workflow-based workload model 33% of workflows contain more than 100 tasks. As compared to the workload model of [Ostermann et al., 2008], where more than 75% workflows compose of less than 40 tasks, the workflow-based workload model has a high number of tasks in workflows.

There can be $N \times N$ number of edges in a workflow, where $N$ is the number of tasks in a workflow. The workflow-based workload model when analyzed in terms of number of edges in a workflow, it is concluded that more than 50% of workflows belong to a high number of edges (NoS High) class of the HC-test bench. The CDF of the number of edges of workflow-based workload model is shown in Figure 5-15.
In the workflow-based workload model 43% of workflows belong to heavy computational weigh and 41.24% belong to heavy communication weights class. The percentage of different classes of HC-test bench workflows in workload model is shown in Table 5-2.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>% of total workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher number of dependencies</td>
<td>50%</td>
</tr>
<tr>
<td>Heavy computational weight</td>
<td>43%</td>
</tr>
<tr>
<td>Heavy communication weight</td>
<td>41.24%</td>
</tr>
</tbody>
</table>

The previous discussion showed that in the workflow-based workload model the number of tasks of the containing workflows varies from 6-250 and the number of edges the workflows also belong to different classes of the HC-test bench. The workflows on the basis of computation and communication weights also belong to different categories of the HC-test bench. Therefore, the workflow-based workload model fulfills the requirement (Requirement R1), i.e., the workflows in the workflow-based workload model possess a diverse structure.
The characteristics of the workflow-based workload model are categorized into static characteristics and dynamic characteristics as explained in subsection 5.3.1. The modeling of static characteristic is explained in subsection 5.3.3, the analysis of the number of tasks in the workflow and the number of edges in the workflows of a workflow-based workload is presented above. So the requirement (Requirement R2) i.e., to provide the modeling of static characteristic is fulfilled.

The arrival time of workflows in workflow-based workload model is achieved by Poisson simulation. In Figure 5-16 the arrival rate of hourly interval is shown, here the maximum 128 workflows per minute and minimum 85 workflows per minute. The mean workflows are 102. In order to reduce the arrival rate the mean of Poisson can be adjusted to achieve the lower or higher number of arrivals. The workflow-based workload model contains the 48000 workflows and the arrival time of the workflows is modeled according to algorithm 5.2, therefore the requirement (Requirement R3) is fulfilled by the workflow-based workload model.

Figure 5-16: Workflow arrival rate of an hourly interval
The workflows can also be categorized according to lower bound, upper bound and mean values of computational time, data movement time and makespan. Since these are dependent on the grid resources characteristics, therefore these can be calculated only having the information about the system configurations as discussed in section 5.3.4. The computation time, data movement time and makespan of the workflows in the workflow-based workload model are calculated for EU data grid configurations, which can be found in [Sulistio et al., 2008]. The average values of lower bound, upper bound and mean of computation time, data movement time and makespan of each class (L, M, S) of workflows are calculated and are shown in Table 5-3, Table 5-4 and Table 5-5 respectively. Hence the (Requirement R4) i.e. modeling of makespan of the workflow is fulfilled. The time unit of all these characteristics is hour. The lower bound of the makespan of S class workflows is calculated by the modeling method provided in section 5.3.4. In order to achieve the mean value of the lower bound of makespan for S class workflows the sum of the lower bound of the makespan of all S class’s workflows are divided by the number of S class workflows in the workflow-based workload model. Similarly the mean lower bound of the computation time and data movement time are calculated for each class of workflows. The average value for the upper bound and mean are also calculated by the same method.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Large Size Workflows</th>
<th>Medium Size Workflows</th>
<th>Small Size Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower Bound Of Computation Time</td>
<td>$1.967 \times 10^{-4}$</td>
<td>$1.24 \times 10^{-4}$</td>
<td>$1.6 \times 10^{-5}$</td>
</tr>
<tr>
<td>Lower Bound Of Data Movement Time</td>
<td>0.0106</td>
<td>0.0045</td>
<td>$2.4 \times 10^{-4}$</td>
</tr>
<tr>
<td>Lower Bound Of Makespan</td>
<td>0.0108</td>
<td>0.00470</td>
<td>$2.5 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Large Size Workflows</th>
<th>Medium Size Workflows</th>
<th>Small Size Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper Bound Of Computation Time</td>
<td>0.006655</td>
<td>0.00285</td>
<td>$1.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>Upper Bound Of Data Movement Time</td>
<td>15.160343</td>
<td>6.4920</td>
<td>0.34</td>
</tr>
<tr>
<td>Upper Bound Of Makespan</td>
<td>15.1669</td>
<td>6.4949</td>
<td>$1.03 \times 10^{-5}$</td>
</tr>
</tbody>
</table>
Table 5-5: Mean value of computation, communication time and makespan

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Large Size Workflows</th>
<th>Medium Size Workflows</th>
<th>Small Size Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Computation Time</td>
<td>$8.122 \times 10^{-6}$</td>
<td>$7.9086 \times 10^{-5}$</td>
<td>$8.3 \times 10^{-8}$</td>
</tr>
<tr>
<td>Mean Data Movement Time</td>
<td>$3.9022 \times 10^{-4}$</td>
<td>$2.722 \times 10^{-6}$</td>
<td>$3.7 \times 10^{-7}$</td>
</tr>
<tr>
<td>Mean Makespan</td>
<td>$3.98346 \times 10^{-4}$</td>
<td>$0.1880 \times 10^{-5}$</td>
<td>$4.5 \times 10^{-7}$</td>
</tr>
</tbody>
</table>

It is observed that in all most all cases, the data movement time is greater than computation time of the workflows. So while designing a workflow scheduling algorithm the communication among the workflow’s task must be handle intelligently for a greater cut down from the makespan.

5.8. Summary

This chapter presents a workflow-based workload model. The characteristics of the workflow-based workload are discussed by categorizing them into the static characteristics and the dynamic characteristics. Both the static and dynamic characteristics are mathematically modelled for generating the workflow-based workload model (Requirements R1-R4). Next, the layered architecture of the GridSim simulator is discussed. Using the workflow-based workload model, a workflow-based application model is designed and integrated into the GridSim simulator. Next, the steps of creating a simulated grid environment in the GridSim simulator are presented (Requirement R5). A simulation experiment is created using the GridSim simulator and the workflow-based application model by following the steps. The characteristics of the workflow-based workload model studied in the experiment include: the number of tasks and number of dependencies of the workflows contained in the workflow-based workload model, the computational weights and the communication weights of the tasks and the edges respectively. The workflows contained in the workflow-based workload model are categorized with respect to their number of tasks into three classes namely small, medium and large. The computation time, the communication time and the makespan of each workflow is calculated and the mean value of the computation time, the communication time and the makespan for each class is presented.
6. Design of Prediction-based Grid Workflow Scheduler

Section 5.6 discussed the layered architecture of a simulated grid environment (cf. Figure 5-11). This chapter presents the design specification of a decentralized grid workflow scheduler with an aim to address the requirement (Requirement R6 cf. section 2.3). This chapter is organized as follows: Section 6.1 provides a brief overview of the prediction process in the context of grid workflow scheduling. Section 6.2 presents the architecture of a decentralized grid workflow scheduler. Section 6.3 presents the proposed workflow scheduler architecture using the UML class diagrams. Finally the section 6.4 summarizes this chapter.

6.1. Prediction Process and Grid Workflow Scheduling

A general prediction process is shown in Figure 6-1.

![Figure 6-1: A general prediction process](image-url)
The prediction process begins by information gathering in the Observation process. The gathered information is used to build some Assumptions during the prediction process. The Prediction Model applies a prediction technique on the gathered information and the Assumptions and forecasts the future state of the system, i.e., Prediction Parameters. An Experiment is conducted to evaluate the Prediction Parameters. Usually the Experiment is conducted using simulation. The Prediction Accuracy is checked by comparing the values of the Prediction Parameters and the actual values obtained from the Experiment. For future predictions, the Assumptions are adapted according to the Prediction Accuracy.

Figure 6-2 shows a cycle of grid scheduling based on a Prediction Model. A Grid scheduler utilizes the Prediction Model to schedule the tasks to grid resources by taking the Prediction Parameters provided by the Prediction Model into account. The examples of such Prediction Parameters include the grid resources’ load, the grid resources’ failure times etc.

As described in the general prediction process (cf. Figure 6-1) the Prediction Model component generates the Prediction Parameters and assigns them to the Grid Scheduler component. The Grid Scheduler component uses the Prediction Parameters and generates a schedule, which consists of the task to resource mappings and the Expected Results (i.e., the
expected completion time of the tasks on the grid resources where these tasks are mapped). The Tasks are assigned to the Grid Resources by the Grid Scheduler as specified in the schedule. The Grid Resources execute the Tasks and send the Results back to the Grid Scheduler. The Grid Scheduler sends these Results as well as the Expected Results to the Prediction Accuracy component. The Prediction Accuracy component calculates the difference between the Expected Results (specified in the schedule) and the actual Results. In the future the Prediction Parameters are adapted according to Prediction Accuracy of the previous predictions.

Usually, the Prediction Model generates the Prediction Parameters (e.g., resource load at a time, failure time of a particular resource etc.) by using a mathematical model and/or the historical data analysis. The Grid Scheduler utilizes the Predicted Parameters for making the decisions regarding tasks to resource mappings. The mathematical model and historical data analysis methods used to generate the prediction parameters are discussed next.

**Mathematical Model**

As described in the general prediction process (cf. Figure 6-1), the input parameters of a prediction model are calculated in the observation process. In the context of grid workflow scheduling, the input parameters to a prediction model may include the number of tasks on a grid resource for a certain period of time, the average task length of the tasks at a grid resource etc. These input parameters are forwarded to the prediction model and a mathematical model contained in the prediction model defines the relationships among the input parameters as functions. These functions are the output of the prediction model. The sequence of steps followed in order to construct the mathematical model for grid workflow scheduling is listed below.

1. Finding the non-changeable parameters: In case of grid workflow scheduling, these non-changeable parameters may be a workflow specification such as number of tasks, number of edges in a workflow, the number of processors in a grid resource, the architecture of a grid resource etc.

2. Fixing the Assumptions e.g., the average task length of a grid resource for a particular time period, the number of tasks remained at a grid resource during a specific time period etc.
3. Setting the Prediction Parameter: i.e., finding the answer of the question “what to predict?” E.g., a grid scheduler may submit the tasks of the workflows to a grid resource with a minimum LUs’ load. In this case, the grid scheduler needs to predict the LUs’ load at the grid resources as a prediction parameter.


The prediction models can be categorized into two classes, namely: Continuous Prediction Model and Discrete Prediction Model. In a Continuous Prediction Model the contained mathematical model defines the relationships among the input parameters using the Differential and/or Partial Differential equations. However, in a Discrete Prediction Model the contained mathematical model defines the relationships among the input parameters using the algebraic equations.

**Historical Data Analysis**

The prediction of a future state of the grid environment can be done by the analysis of the historical data. The historical data can be obtained through experiments. The experiments can be conducted either by simulating a grid scenario, or executing a grid scenario on a real grid environment and maintaining a history log. The future state of the grid is then predicted by observing the historical data obtained by the experiments.

The prediction model provides the prediction parameters as output. Next, the prediction parameters are described.

**Prediction Parameters**

The prediction model applies the prediction method to forecast the prediction parameters. These prediction parameters are the output of the prediction model. The grid workflow scheduler uses the predicted parameters (e.g., grid resources LUs’ loads, queue wait time of the grid resource etc.) for taking the task to resource mapping decisions for the workflows’ tasks. There are three major classes of the prediction parameters for grid workflow scheduling, namely: the prediction parameters about the grid resources, the workflows and the grid environment. Next, each of these classes of the prediction parameters is described.
Grid resources: The prediction parameters about the grid resources include: the availability of a grid resource at a particular time, the probability of the failure of a grid resource, the failure rate of a grid resource, the load at a grid resource, queue wait time of a grid resource etc. The focus of this thesis is limited to the prediction of the loads of the grid resources and queue wait time of the grid resources as stated in requirements (Requirements R7-R9).

Workflows: The prediction parameters about the workflows include: the completion time of a task of a workflow on a grid resource, the data movement time between the tasks of a workflow, the tasks’ wait time at different grid resources and the makespan of a workflow. This thesis considers all of these prediction parameters.

Grid environment: Since the grid environments are highly dynamic, the resources may join and leave the grid environment at any time. This affects the performance of the grid environment. E.g. a grid resource leaving the grid environment necessitates the rescheduling of all the tasks assigned to it and the rescheduling time is added to the makespan of the related workflows. The prediction parameters related to a grid environment may include: the number of tasks completed per unit time, the number of failure-free resources. This thesis assumes that neither a grid resource fails nor a new grid resource is added during the simulation of an experiment i.e., the number of grid resources remain fixed. Therefore, the prediction parameters related to the grid environment are beyond the scope of this thesis and are not considered. However, the dynamicity of grid environment is defined in terms of the fluctuation of the load at the grid resources due to various competing schedulers and the LUs of the grid resources.

6.2. Architecture of a Prediction-based Decentralized Grid workflow Scheduler

This section presents the architecture of a prediction-based decentralized grid workflow scheduler. Subsection 6.2.1 describes the prediction in the context of the decentralized grid workflow scheduler and subsection 6.2.2 explains the architecture of the proposed prediction-based grid workflow scheduler.
6.2.1. Decentralized grid workflow scheduler and prediction

According to the requirement **R6**, a decentralized architecture is required for handling multiple workflows in the grid environment at a time. In order to address the requirement (Requirement **R6**), this subsection provides a decentralized grid workflow scheduler. Figure 6-3 shows a conceptual view of a decentralized grid workflow scheduler, where more than one grid schedulers are present and work parallel to each other.

![Figure 6-3: Conceptual view of decentralized grid workflow scheduler](image)

As shown in Figure 6-3, the *Grid Resources* register themselves with the *Grid Services* and receive the unique identification number (ID) in response (shown as a solid two way line). A decentralized *Grid Workflow Scheduler* receives the workflows from the *Grid User* (shown as a dotted line towards the Grid Workflow Scheduler). The Grid Workflow Scheduler contacts the Grid Services for the available grid resources (shown as a solid two way line). A Grid Workflow Scheduler submits the tasks of the workflow to the available grid resources according to the schedule generated by the implemented workflow scheduling algorithm (shown as a dotted one way line from the Grid Workflow Scheduler). The grid resources receive tasks (shown as a dotted one way line towards the grid resources), execute the tasks and send the results back to the corresponding Grid Workflow Scheduler (shown as a dotted one way line from the Grid Resources). The *Grid Workflow Scheduler* sends the results of a
completed workflow to the corresponding Grid User (shown as a dotted one way line from the Grid Workflow Scheduler to the Grid User).

The role of a Grid Workflow Scheduler is to provide a transparent access to the grid resources for the Grid Users. However, the Grid Users’ involvement in the scheduling process (i.e., selection of the grid resources) is recommended in order to help optimize the user-centric objectives. Such involvement is defined in terms of the requirements specified by the Grid Users. The Grid Users’ requirement may include the minimum number of processors required to execute the tasks of their workflows, the memory required by the tasks of a workflow etc. The Grid User’s requirements for a workflow are associated with the workflow at the time of workflow submission to a Grid Workflow Scheduler.

A Grid Workflow Scheduler discovers the Grid Resources by contacting the Grid Services (specifically the GIS). The resource discovery is a process to get the information about the available grid resources. However, the monitoring process includes the retrieval of dynamic information about the state of the grid resources (like number of free processors at a grid resource, number of tasks at the grid resource etc.). The resource monitoring can either be performed by using one of the following two approaches:

1. The Grid Workflow Scheduler periodically queries the local information service of each grid resource, if provided. In case where the local information services are present on the Grid Resources, the GIS provides a mechanism to contact the local information services of the grid resources.

2. The second approach is to monitor the grid resources through the GIS. The GIS periodically queries the grid resources about their state. The grid workflow scheduler can contact the GIS and get this information. In this way, the number of queries to the grid resources can be reduced. This results in saving the expensive bandwidth thereby making the bandwidth available for the data migration for the tasks of the workflows.

Due to the advantages of the second approach, it is used for the resource monitoring in this thesis, i.e., the GIS is used for the resource discovery as well as the resource monitoring. Since the number of grid schedulers and the grid resources may be large in a grid environment, the GIS must be scalable and sensitive to changeability. Scalability means that
a GIS must handle a large number of grid schedulers. Changeability means the frequency of
the changing state of the grid resources. When the GIS is sensitive to changeability, it has to
provide up-to-date information regarding the grid resources. Special purpose GIS can also be
used to monitor a specified type of the grid resources, example is Data Locator which is used
to keep the track of storage units available at each grid resource and the available space on
them. The Data Locator can be used to reduce the load at the single available GIS. Moreover,
a Grid Workflow Scheduler contacting with Data Locator will receive only the information
about the storage. This way, a Grid Workflow Scheduler limits itself to the retrieval of
relevant information. The GIS and Data Locator are called the Grid Services in this thesis.

The GIS and the Data Locator periodically update the information about the state of the grid
resources (e.g., the number of free processors in a grid resource, the number of tasks at the
grid resource and the available storage space at a grid resource etc.). Due to the dynamicity
of the grid resources, the grid resource monitoring is complicated. It is also a fact that
information about the grid resources provided by Grid Services may be incomplete due to
the dynamic nature of grid resources. E.g., the number of free processors of a grid resource
saved in GIS repository may not actually be free due to its LUs’ tasks arrived after it was
queried about this information. So, the list of available grid resources and the information
about the state of the grid resources maintained by the GIS cannot be fully trusted.
Therefore, the information about the future state of the grid resources needs to be
predicted in order to ensure the availability of the accurate information required by the Grid
Workflow Scheduler. The next subsection discusses the architecture of one of the
prediction-based grid workflow schedulers present in the decentralized grid workflow
Schedulers.

6.2.2. Architecture of a prediction-based grid workflow scheduler

The grid workflow scheduler maps the tasks of a workflow to the grid resources in order to
achieve the scheduling objective; mostly the objective considered by the grid workflow
schedulers is the achievement of low makespan of the workflows. As discussed in subsection
5.3.4, the makespan of a workflow is dependent upon the pre-execution, execution and
post-execution steps of the workflow. A grid workflow scheduler consists of various
components responsible for performing the pre-execution, execution and post-execution
steps. Figure 6-4 depicts the important components of the grid workflow scheduler, which
include: the Workflow Queue, the Workflow Manager, the Scheduling Algorithm, the Predictor, the Ready Tasks List, the Dispatcher and the Result Gatherer. Note that the links from the Grid Resources towards the Grid Services are not numbered in Figure 6-4 because these links describe the registration of grid resources and the information gathering. A detailed account of these steps is discussed section 5.5. Next, the functionalities of the components of the Grid Workflow Scheduler are described.

**Workflow Queue**

The Workflow Queue is a queue data structure where the received workflows are kept in the same order as they were received. The workflows always enter from the one end of the Workflow Queue (referred as the Rare of the queue) and always leave the Workflow Queue from the other end (referred as Front) as shown in Figure 6-4.
Workflow Manager

When a workflow reaches at the front of the Workflow Queue, it is sent to the Workflow Manager. The Workflow Manager checks the dependencies among the tasks of a workflow and maintains a Ready Tasks List. The Ready Tasks List contains the tasks of a workflow, which are no more dependent on any other tasks of the workflow and the data locations of the tasks are known.

Predictor

The Predictor gets the information about the state of the grid resources (e.g., the number of tasks at a grid resource) from the GIS. The assumptions are made which are passed to the prediction model of the Predictor. The prediction model generates the prediction parameters like the average number of tasks arriving at a grid resource from its LUs and the tasks lengths of tasks from the LUs of a grid resource, or queue wait time of a workflow’s task on a grid resource. When the results of the executed tasks of a workflow are received from the grid resources at the Grid Workflow Scheduler, these results are passed to the Predictor, so that the prediction accuracy can be measured and the consequent prediction parameters are adapted accordingly.

Data Movement Controller

The Data Movement Controller gets the information like the number of storage units in a grid resource, the storage capacity of each storage unit, the available free storage space at each storage unit in a grid resource from the Data Locator. The Data Movement Controller calculates the data movement time of the tasks of the workflow on different grid resources. Also, it saves the data files generated by the tasks of the workflow on the available storage space of the grid resources. The Data Movement Controller implements a data saving algorithm so that the time spent on data movement can be reduced.

Scheduling Algorithm

The Scheduling Algorithm uses the information gathered from the Predictor and the Data Movement Controller and generates a schedule according to the implemented algorithm. Note that the scheduling algorithms and Predictor component are discussed in detail in the upcoming chapters. The schedule of the tasks of a workflow consists of the tasks, the grid
resources (where the tasks are mapped) and the required data file locations of the tasks. E.g., \((t_i, R_k, FL(dataFile_1, r), \ldots, FL(dataFile_p, m))\) means that the task \(t_i\) is scheduled to the grid resource \(R_k\), the \(dataFile_1\), generated for the task \(t_i\) is saved on the grid resource \(R_r\) and so on.

**Dispatcher**

The *Dispatcher* receives a schedule and dispatches the tasks to the grid resources as specified in the schedule.

**Result Gatherer**

The Result Gatherer component receives the results of the completed tasks of a workflow and updates the Workflow Manager, the Predictor and the Data Movement Controller. When all the tasks of a workflow complete their execution, the Result Gatherer sends the results of the workflow to the corresponding Grid User.

In Figure 6-4 the architecture of a grid workflow scheduler and a workflow scheduling scenario are shown. The numbers on the directed edges shown in Figure 6-4 depict the flow of execution of a workflow. This flow begins by the submission of a workflow by a Grid User to a *Grid Workflow Scheduler* and finishes when the Grid User receives the results of the executed workflow from the *Grid Workflow Scheduler*. The description of each step shown as a numbered directed edge in Figure 6-4 is provided next.

1. A *Grid User* submits a workflow to a *Grid Workflow Scheduler*. The arrival of a workflow from a Grid User is shown as a directed edge towards the Workflow Queue. The workflows are received at the *Grid Workflow Scheduler* and are placed in the *Workflow Queue* at the *Grid Workflow Scheduler*. The workflows are executed in the same order as they were received, i.e., the *Workflow Queue* uses the First Come First Serve (FCFS) mechanism.

2. When a workflow reaches at the front of the Workflow Queue, it is assigned to the Workflow Manager. The Workflow Manager generates the specifications of the tasks and the dependencies of the workflow. The Workflow Manager maintains a Ready Tasks List, which is used to save the tasks having no dependency constraints on the other tasks of the workflow.
3. The two components involved in this step are the Predictor and the Data Movement Controller. The Predictor and the Data Movement Controller get the information from the Grid Services as explained above. The respective communication between the Predictor, the GIS and Data Movement Controller and Data Locator are shown as directed edges in Figure 6-4 and are explained next.

   a. The *Predictor* gets the information like the number of tasks at the grid resources from the GIS.

   b. The *Data Movement Controller* gets the information regarding available storage space at different grid resources from *Data Locator* and saves the data files generated by the workflows’ tasks on different resources subject to the availability of storage space. The *Data Locator* maintains a repository of the generated data files during the course of execution of the workflows.

4. The Workflow Manager, Predictor and Data Movement Controller are involved in this step. The communications of the involved components are explained below.

   a. The *Workflow Manager* assigns the *Ready Tasks List* to the *Scheduling Algorithm*.

   b. The Predictor generates and forwards the predicted parameters (e.g., the average number of tasks arriving at a Grid Resource from its LUs, average task length of the tasks from the LUs’ at the Grid Resource etc.) to the Scheduling Algorithm.

   c. The Data Movement Controller calculates the data movement time of the tasks of a workflow on different *Grid Resources* and forwards it to the Scheduling Algorithm.

5. The Scheduling Algorithm generates a schedule and assigns it to the Dispatcher.

6. The Dispatcher submits the tasks of the workflow to the grid resources as specified in the schedule.
7. The Grid Resources send the results of the completed tasks of the workflow to the Grid Workflow Scheduler. The results sent by the Grid Resources are received by the Result Gatherer contained in the Grid Workflow Scheduler.

8. The Result Gatherer forwards the gathered results to the Data Movement Controller, the Workflow Manager and the Predictor. The Predictor calculates the difference between the predicted and the actual results and updates the prediction parameters for the future. The Workflow Manager removes the dependency constraints of the child tasks of the parent tasks of the workflow, which are completed (see step 7).

9. The Data Movement Controller consults the Data Locator for deleting the data files used by the completed tasks and for saving the data files generated by the completed tasks of a workflow. The steps 3-9 are performed until all the tasks of a workflow complete the execution and the results are received by the Result Gatherer.

10. When all the tasks of a workflow complete their execution, the results of the workflow are sent to the corresponding Grid User.

6.3. UML Modeling of Prediction-based Grid Workflow Scheduler

The previous section described the design of a prediction–based grid workflow scheduler. This section elaborates the design by presenting the class diagrams of the components of the prediction-based grid workflow scheduler using the Unified Modeling Language (UML). Note that an abstract design specification is presented in this section. This design specification is extended in chapters 7 and 8 for presenting the realization of the prediction-based grid workflow schedulers.

Since the prediction-based grid workflow scheduler uses the prediction to schedule the tasks of a workflow to the grid resources, the classes related to prediction are introduced as shown in Figure 6-5.
Chapter 6 Design of Prediction-based Grid Workflow Scheduler

Figure 6-5: Class diagram of prediction classes

PredictionParameters class

As discussed in section 6.1, the prediction model generates the prediction parameters. Examples of a prediction parameter include the LUs' load at a grid resource and the queue wait time at a grid resource. The abstract class PredictionParameters (Figure 6-5) is introduced as a data container for maintaining the prediction parameters. The realizations of a prediction model extend the PredictionParameters abstract class for providing the concrete implementation of a prediction parameter.

ActualResults class

As discussed in section 6.1, the predicted parameters are compared with the actual results in order to measure the accuracy of the prediction model. The abstract class ActualResults is introduced as a data container for maintaining the actual results obtained through a simulated experiment. The realizations of a prediction model extend the ActualResults abstract class for providing the concrete implementation of the actual results.

Predictor class

The Predictor class (Figure 6-5) is introduced for realizing a prediction model. It provides the predict method for generating the prediction parameters resulting from the realized prediction method.

+predict(): List<PredictionParameters>

The predict method generates and returns a list of PredictionParameters. The predict method implements the prediction model. Two prediction models proposed in this thesis are LUs’ Load Prediction (LULP) model (see Chapter 7) and the Queue Wait time Prediction
(QWP) model (see Chapter 8). Both these prediction models provide an implementation of the predict method of the Predictor class.

The update method of the Predictor class is introduced for updating the assumptions and measuring the accuracy of the prediction.

```java
+update(in actualResults : List<ActualResults>) : void
```

The update method takes a list of ActualResults instances in the actualResults input parameter. The prediction accuracy is measured and the assumptions of the prediction model are updated accordingly. This method returns nothing.

Since data movement is an important factor to be handled in grid workflow scheduling, the classes related to data movement (Figure 6-6) are described next.

**DataMovementController**

- resourceIDs : List<int>
- fileLocationsOfProducedFiles : Map<int, List<FileLocation>>
- fileLocationsOfConsumedFiles : Map<int, List<FileLocation>>
- dataSavingAlgorithm : IDataSavingAlgorithm

+DataMovementController(in dataSavingAlgorithm : IDataSavingAlgorithm, in resourceIDs : List<int>)
++deleteFileConsumedByTask(in taskNumber : int) : boolean
++saveFilesProducedByTask(in taskNumber : int, in files : List<DataFile>) : List<FileLocation>
++getFileLocationsProducedForTask(in taskNumber : int) : List<FileLocation>
++calculateDataMovementTime(in taskNumber : int, in resourceId : int) : double

**DataLocator**

- resourcesTotalStorage : Map<int, List<Storage>>
- resourcesAvailableStorage : Map<int, List<Storage>>

+addFile(in file : DataFile, in resourceId : int) : boolean
+deleteFile(in file : DataFile, in resourceId : int) : boolean
+locateFile(in fileName : String) : int
+getAvailableStorage() : Map<int, List<Storage>>

**FileLocation class**

Since the data files produced by the parent tasks are required by the child tasks, these data files need to be saved on a grid resource and the location of these data files need to be stored. The FileLocator class (Figure 6-6) is introduced for maintaining the location of a data file on a grid resource. The class models the file attribute for maintaining the data file
as a `gridsim.datagrid.DataFile` instance and the `storingResourceId` attribute for maintaining the ID of the grid resource, where the data file is saved. The `FileLocator` class provides the setter and the getter methods for setting the value of the attributes and retrieving them respectively. However, the setter and getter methods are not shown in Figure 6-6 due to readability reasons.

**IDataSavingAlgorithm Interface**

Since the data files produced by the tasks of a workflow need to be saved, the `IDataSavingAlgorithm` interface (Figure 6-6) is introduced for this purpose. The `IDataSavingAlgorithm` interface has to be implemented by a class realizing a data saving algorithm. The `IDataSavingAlgorithm` interface provides the `saveFiles` method for saving the data files on the grid resources.

```
+saveFiles(in files : List<DataFile>, in availableStorage : Map<int, List<Storage>>) : List<FileLocation>
```

The `saveFiles` method takes a list of data files in the `files` input parameter and a map of available storage on the grid resources in the `availableStorage` as input parameter. The key in the `availableStorage` map is the ID of a grid resource and the value is the list of storage units available on the grid resource, which is modelled as a `gridsim.datagrid.Storage` instance. The method saves the data files to the grid resources and returns a `List<FileLocation>` instance specifying the location of the saved data files.

**DataLocator class**

The data files produced by the parent tasks in a workflow are saved on the grid resources until their child tasks requiring those data files get ready. A task of a workflow get ready when all of its parents tasks complete their execution and the locations of the data files produced by the parent tasks are known. The locations of the data files required by a task are needed for its schedule generation. Therefore, the `DataLocator` class is introduced for maintaining the location of the data files saved on the grid resources.

The `DataLocator` class (Figure 6-6) models the `resourceTotalStorage` attribute for maintaining the storage capacities of the grid resources as a map. The key in the
resourceTotalStorage map is the ID of a grid resource and the value is the total storage on the grid resource, which is modelled as a list of the gridsim.datagrid.Storage instances. The DataLocator class models the resourceAvailableStorage attribute for maintaining the available storage on the grid resources as a map. The key in the resourceAvailableStorage map is the ID of a grid resource and the value is the available storage on the grid resource, which is modelled as a list of the gridsim.datagrid.Storage instances.

The class DataLocator provides the addFile method for saving a data file on a grid resource.

+addFile(in file : DataFile, in resourceID: int): boolean

The addFile (Figure 6-6) method takes a data file in the file input parameter and the resource’s ID in the resourceID input parameter. The method saves the file on the resource identified by the resourceID input parameter. The addFile method returns a boolean instance specifying whether or not the data file was successfully saved.

The class DataLocator provides the deleteFile method for deleting a data file from a grid resource.

+deleteFile(in file : DataFile, in resourceID: int): boolean

The deleteFile (Figure 6-6) method takes a data file in the file input parameter and the resource’s ID in the resourceID input parameter. The method deletes the file on the resource identified by the resourceID input parameter. The deleteFile method returns a boolean instance specifying whether or not the data file was successfully deleted.

The class DataLocator provides the locateFile method for locating a data file.

+locateFile(in fileName : String): int

The locateFile (Figure 6-6) method takes a data file’s name in the fileName input parameter. The method searches and locates the file on a grid resource and returns an int instance specifying the ID of the resource where the file is stored. If the data file is not found on any of the grid resource than locateFile method returns 0.
The class `DataLocator` provides the `getAvailableStorage` method for querying the available storage units in the grid resources.

```
+getAvailableStorage(): Map<int,List<Storage>>
```

The `getAvailableStorage` (Figure 6-6) method returns the available storage units on all the grid resources as a map. The key in the map is the ID of a grid resource and the value is the list of available storage units on the grid resource.

**DataMovementController class**

Since the data files produced by the tasks of a workflow are later consumed by some other tasks of the workflow, the data files produced by the tasks of a workflow need to be saved. Each task of a workflow may consume a list of data files and produce a list of data files.

A task of a grid workflow when scheduled to a grid resource the data movement time of the task on the resource must be taken into account. The class `DataMovementController` (Figure 6-6) is introduced to provide a repository of the data files produced and consumed during the execution of a workflow. It also calculates the data files movement time of a task of a workflow at a grid resource, where the date files produced for the task of the workflow are saved on various resources of the grid.

The `DataMovementController` class models a list of `resourceIDs` as a list of `int` instances. The `resourceIDs` contains the unique IDs of the grid resources, which do not change throughout a simulation experiment. The `DataMovementController` class models the `filelocationsOfProducedFiles` attribute as a map to store the locations of the data files produced by the tasks of a workflow. The key in the `filelocationsOfProducedFiles` map is the task number of a task of a workflow and the value is the list of the locations of the data files produced by the task. The `DataMovementController` class models the `filelocationsOfConsumedFiles` attribute as a map to store the locations of the data files consumed by the tasks of a workflow. The key in the `filelocationsOfConsumedFiles` map is the task number of a task and the value is the list of the data files’ locations consumed by the task. The `DataMovementController` class models the `dataSavingAlgorithm` attribute modeled as an instance of the interface `IDataSavingAlgorithm` for providing the functionality of data saving.
The class provides a constructor method for creating its instance.

```csharp
+DataMovementController(in dataSavingAlgorithm : IDataMovementController,
in resourceIDs : List<int>)
```

The method `DataMovementController` takes two input parameters, i.e., the `dataSavingAlgorithm` and the `resourceIDs`. The method creates an instance by assigning the input parameters to the corresponding attributes.

The data files saved on different grid resources are needed to be deleted when these files are no more required. The deletion of the data files is important in order to make the storage units free for more data files. The method `deleteFilesConsumedByTask` is provided to delete the data files consumed by a task.

```csharp
+deleteFilesConsumedByTask (in taskNumber : int): boolean
```

The method `deleteFilesConsumedByTask` takes an instance of `int` as an input parameter `taskNumber` specifying the task number of a task. The method deletes all the files consumed by the task specified by the `taskNumber`.

The method `saveFilesProducedByTask` is provided to save the data files produced by a task.

```csharp
+saveFilesProducedByTask (in taskNumber : int, in files : List<DataFile>):
List<FileLocation>
```

The method `saveFilesProducedByTask` takes an instance of `int` as an input parameter `taskNumber` specifying the task number of a task. The second input parameter, i.e., `files`, is a list of datafiles. The method utilizes the input parameters for saving the data files and returns a list of `FileLocation` instances defining the locations of the data files.

When a task is scheduled on a grid resource, the data files produced by the parent tasks of the task are needed to be located. The method `getFileLocationsProducedForTask` is provided for locating the data files produced for a task of a workflow.

```csharp
+getFileLocationsProducedForTask (in taskNumber : int): List<FileLocation>
```

The method `getFileLocationsProducedForTask` takes an instance of `int` in the `taskNumber` input parameter identifying a task. The method locates the data files produced
for the task identified by the taskNumber and returns a list of FileLocation instances specifying the locations of the data files.

The data movement time of a task of a workflow is an important factor which must be taken into account while taking the tasks to resource mapping decisions. The method calculateDataMovementTime (Figure 6-6) of the class DataMovementController provides the realization of data movement time calculation of a task on a grid resource.

+ calculateDataMovementTime(in taskNumber : int, in resourceID : int): double

The method takes two input parameters taskNumber and resourceID, both are the instance of int. The method calculates the data movement time of the task using the data file movement time of the task specified as taskNumber on the resource specified by the resourceID and returns the data movement time of the task on the resource.

The grid workflow scheduler maps the tasks of a workflow to the grid resources with an aim to optimize certain objective, mostly makespan. Figure 6-7 depicts the classes involved in the schedule generation of a workflow.
ScheduleItem Class

The ScheduleItem class (Figure 6-7) is introduced for maintaining the schedule of a task of a workflow. The class models the task attribute for maintaining the task of a workflow as a gridsim.datagrid.DataGridlet instance and the resourceId attribute for maintaining the ID of the grid resource, where the task is scheduled. The requiredFiles attribute of the class provides a list of the file locations where the data files produced for the task are saved. The class ScheduleItem provides setter and getter methods for setting the value of the attributes and retrieving them respectively. However, the setter and getter methods are not shown in Figure 6-7 due to readability reasons.

TaskToResourceMapping Class

The TaskToResourceMapping class (Figure 6-7) is introduced for maintaining the mapping of a task of a workflow to a grid resource. The class models the taskNumber attribute for maintaining the task number of a workflow task instance and the resourceId attribute for maintaining the ID of the grid resource, where the task is mapped. The class TaskToResourceMapping provides setter and getter methods for setting the value of the attributes and retrieving them respectively. However, the setter and getter methods are not shown in Figure 6-7 due to readability reasons.

WorkflowManager Class

The instance of a WorkflowManager class is used to maintain the ready task list of a workflow and the workflow itself. The attribute workflow of the WorkflowManager class models a workflow, which is already explained in section 5.6. The readyTasksList attribute maintains the ready tasks of a workflow, modeled as an instance of gridsim.datagrid.DataGridletList. The map scheduled is modeled in workflow manager to check that whether all the tasks of a workflow are scheduled or not. The key in the map of scheduled is the task number of a task and the value is an instance of boolean showing that a task is scheduled or not. If the boolean value shows that a task is scheduled then it is not added in the ready tasks list.
The workflow attribute of the class WorkflowManager needs to be initialized. The setworkflow method provides the realization for the initialization of the workflow attribute.

+ setWorkflow(in workflow : workflow): void

The method setworkflow takes an instance of Workflow as an input parameter the method returns nothing.

+ getreadyTasksList():DataGridletList

The method getreadyTasksList returns a DataGridletList of containing the ready tasks of the workflow saved in the attribute workflow.

+ removeDependencyConstraints(in taskNumber : in): void

The method removeDependencyConstraints (Figure 6-7) takes a taskNumber as input parameter and removes the dependency constraints of all the child tasks of the task specified by the taskNumber. This method returns nothing.

ISchedulingAlgorithm Interface

Since the tasks of a workflow are mapped to the grid resources with a certain objective, the ISchedulingAlgorithm interface (Figure 6-7) is introduced for this purpose. The ISchedulingAlgorithm interface has to be implemented by a class for realizing a scheduling algorithm. The ISchedulingAlgorithm interface provides the generateSchedule method for actually generating the mapping of the tasks of a workflow to the available grid resources.

+generateSchedule(in readyTasksLists : List<DataGridletList>, in readyTimeOfResources : Map<int, double>, in MIPSRatingsOfPEs Map<int, long>, in predictionParameters : Map<int, PredictionParameters>, dataMovementTimes : Map<TaskToResourceMapping, double>) : List<TaskToResourceMapping>

The method generateSchedule (Figure 6-7) takes a list of the instances of DataGridletLists in the input parameter readyTasksList. An instance of map is passed as an input parameter in readyTimeOfResources. The key in the map of readyTimeOfResources is the ID of a grid resource and the value is the ready time of the
grid resource. An instance of map is passed as an input parameter in MIPS\text{RatingsOfPEs}. The key in the map of MIPS\text{RatingsOfPEs} is the ID of a grid resource and the value is the MIPS rating of the grid resource. An instance of map is passed as an input parameter in Prediction\text{Parameters}. The key in the map of Prediction\text{Parameters} is the ID of a grid resource and the value is a list of prediction parameters of the grid resource. An instance of map is passed as an input parameter in data\text{MovementTimes}. The key in the map of data\text{MovementTimes} is an object of TaskToResource\text{Mapping} and the value in the map is the data movement time of the task in TaskToResource\text{Mapping} at the grid resource specified in TaskToResource\text{Mapping} object. The method generate\text{Schedule} generates the schedule of the tasks of the workflow, which consist of the task to resource mapping and returns a list of TaskToResource\text{Mapping}.

**Dispatcher Class**

The tasks of a workflow are need to be scheduled according to the generated schedule therefore the class Dispatcher is introduced. Figure 6-8 shows the Dispatcher class.

```
+ dispatch(in scheduleItems : List<ScheduleItem>): void
```

**Figure 6-8: Dispatcher class**

The method dispatch of the Dispatcher class takes a list of schedule items as an input parameter and dispatch the tasks according to the schedule specified in Schedule\text{Items} object of the list Schedule\text{Item}. The dispatch method returns nothing.

The grid resources produce the results after executing the tasks of a workflow. These results need to be gathered and sent to the corresponding grid workflow scheduler. Figure 6-9 shows the result gathering related classes.
IResultListener Interface

The IResultListener interface (Figure 6-9) is introduced for informing the grid workflow scheduler about a completed task of a workflow. The class providing the functionality of the grid workflow scheduler implements this interface for receiving the results. The interface provides the receiveFinishedTask callback method for notifying the results to the implementing classes.

++ receiveFinishedTask(in task : DataGridlet): void

The receiveFinishedTask callback method takes a completed task as a gridsim.datagrid.DataGridlet instance in the task input parameter. The method returns nothing.

ResultGatherer class

The ResultGatherer class (Figure 6-9) is introduced for gathering the results produced by the tasks of a workflow from the grid resources where the tasks have completed their execution. The class follows the Observer design pattern [Gamma et al., 1995] for sending the results to the grid workflow scheduler. The class models the listeners attribute for storing a list of listeners interested in receiving the results of a completed task, where a listener implements the IResultListener interface. Each of these listeners is notified by invoking the receiveFinishedTask callback method as soon as the results of a completed task of a workflow are gathered. The ResultGatherer class provides the addListener method for adding a listener to the listeners attribute.
The `addListener` method takes an instance of a class implementing the `IResultListener` interface in the `listener` input parameter. The method adds the `listener` input parameter to the list maintained by the `listeners` attribute. The method returns nothing.

A grid workflow scheduler receives the workflows from the grid users, who are considered as an owner of workflow application. The grid workflow scheduler schedules the tasks of a workflow to the available grid resources and gets the results of the tasks completed by the grid resources. When all the tasks of a workflow complete their execution the results are sent back to the corresponding grid user. Figure 6-10 depicts a class diagram of the `GridWorkflowScheduler` and the implemented interface of the class.

**GridWorkflowScheduler class**

The `GridWorkflowScheduler` (Figure 6-10) class is introduced to provide the functionality of a grid workflow scheduler. In the grid workflow scheduler a queue of workflows is maintained in the attribute `workflows`. The `workflowManager` attribute `GridWorkflowScheduler` is required to govern the scheduling process by providing a ready
task list of the workflow, removing the dependency constraint of the completed tasks of the workflow and to check that whether all the tasks of a workflow are completed or not. The attributes readyTasksList and finishedTasksList are modeled as instances of the gridsim.datagrid.DataGridletList. Both work as container to keep the ready and completed tasks of a workflow. The predictor attribute is modeled as an instance of Predictor; the attribute is required to get the prediction parameters list during the scheduling process. The attributes dataSavingAlgorithm and schedulingAlgorithm are modeled in GridWorkflowScheduler class to provide the realizations of the function defined in the said interfaces. DataMovementController attribute is modeled in GridWorkflowScheduler class to provide the functionalities like saving the data files produced by the tasks of a workflow, deleting the data files which are no more required, and calculating the data movement time of the task on a grid resource. The dispatcher attributes are required to dispatch the tasks of a workflow to the grid resources. It is modeled in GridWorkflowScheduler class as an instance of Dispatcher. The resultGatherer attributes are required to get the results of the tasks completed by the grid resources. It is modeled in GridWorkflowScheduler class as an instance of ResultGatherer.

The grid user submits the tasks to a grid workflow scheduler, which are in the GridWorkflowScheduler in receiveWorkflow method.

+receiveWorkflow(in workflow : Workflow): boolean

The method receiveWorkflow provides the functionality receiving a workflow from a grid user. The method receiveWorkflow gets an instance of Workflow in the attribute workflow as an input parameter. The method returns an instance of boolean, the value of the instance shows that receiving a workflow was successful or not. Every workflow received in the grid workflow scheduler is placed in the queue; the workflows are placed in the same order as they are received. A workflow is only considered for scheduling if it reached to the front of queue and the previous workflow has been completed.

When a grid resource finishes the execution of a task it sends the results of the completed tasks to the corresponding grid scheduler. An algorithm representing the procedure followed in receiving a completed task of a workflow is shown in Algorithm 6-1.
Algorithm 6-1: Receiving a task of a workflow method

```java
receiveFinishedTask(task)
{
    1. Add task to the finishedTasksList
    2. Call data movement controller to delete the consumed file by the task
    3. Call data movement controller to save the data files produced by the task
    4. Update the predictor
    5. Update the workflow manager to remove the dependency constraints of the task
    6. readyTasksList<-workflowManager.getReadyTasksList
    7. Send the results of a workflow to the corresponding user if all the tasks of a workflow have been completed the execution
}
```

Algorithm 6-1 depicts that a task is received in the GridworkflowScheduler, the task is placed in the finishedTasksList. The dataMovementController functions are called to save the data files produced by the task and delete the data files consumed by the task. The results are passed to Predictor to update the prediction parameters for future. The readyTasksList in the GridworkflowScheduler is assigned the ready tasks of the workflow by the workflowManager. This method works in a thread and the thread remains alive until it receives a finish simulation event from the GIS.

Since every class inherited from gridsim.GridSim must have a body function providing the behavior of the class during the simulation, body function is provided in the class GridWorkflowScheduler.

+body(): void

The body method of the class GridWorkflowScheduler receives no input parameter and returns nothing. In the body method of the class GridWorkflowScheduler the workflows from the grid users are received, scheduled to the grid resources and the result of a completed workflow is sent back to the corresponding grid user. Next an algorithm representing the behavior of a grid workflow scheduler is represented.
The received workflow when reaches at the front of the workflows queue it is assigned to the workflowManager. The workflowManager creates a readytaskslist of the workflow. The GridWorkflowScheduler receives this ready tasks list from the workflowManager by the method getReadyTasksList. The GridWorkflowScheduler gets a list of predictionparameters from the predictor by the predict method. The list of available resources is inquired from GIS. The GridWorkflowScheduler uses the dataMovementController to calculate the data movement time of each task on each available resource by the method calculateDataMovementTime. A list of task to resource mapping is obtained from the schedulingAlgorithm and an instance of List containing the objects of ScheduledItems is created according to task to resource mapping list. The workflowManager is asked to mark the tasks in the scheduled item list as scheduled. The list of scheduled items is passed to the dispatcher. The dispatcher sends the tasks to the grid resources as specified in the list of schedule item. This method works in a thread and the
thread remains alive until it receives a finish simulation event from the GIS or there are unscheduled workflows in the queue workflows.

6.4. Summary

This chapter introduces the general prediction process. Next, a prediction-based grid workflow scheduler is proposed (Requirement R6). The system architecture of the prediction-based grid workflow scheduler is presented by describing its components as well as the communication among the components. A design specification of the components of the prediction-based grid workflow scheduler is presented with the help of class diagrams in the UML. The subsequent chapters extend the design specification for providing the implementation of prediction-based grid workflow scheduling algorithms.
7. Grid Workflow Scheduling Using Resources Local Users’ Loads Prediction

This chapter presents the solutions for the requirements (Requirements R7, R8, R10 and R12). Requirement R7 suggests the prediction of the number of tasks at a grid resource submitted by the LUs, whereas Requirement R8 suggests the prediction of task’s length of a task of the LUs at a grid resource. Requirement R10 suggests the design and development of a grid workflow scheduling algorithm utilizing the predicted LUs’ load, while requirement R12 suggests that the design of a data saving algorithm for saving the data files generated during the execution of the workflows’ tasks.

This chapter is organized as follows: Section 7.1 discusses the conventional workflow scheduling algorithms focusing on their oversimplified assumption that the grid resources are dedicated for the workflows’. This assumption does not hold for a real grid environment because of the presence of the LUs’ load at the grid resources. Therefore, the need of a LUs’ load prediction-based workflow scheduling algorithm is highlighted. Section 7.2 presents a conceptual view of the LUs’ load prediction-based grid workflow scheduling. Section 7.3 describes prediction model for predicting the LUs’ load at the grid resources. Section 7.4 presents a data saving algorithm to save the data files produced during the execution of the workflows in order to reduce the data movement time of the tasks of the workflows. Section 7.5 proposes the grid workflow scheduling algorithms, which utilize the predicted load of the LUs at the grid resources as well as the data saving algorithm. The proposed scheduling algorithms include: the LUs’ load prediction-based Min-Min, Max-Min and the Sufferage algorithms. Section 7.6 describes the experiments conducted for evaluating the LUs’ load prediction-based scheduling algorithms. Finally the section 7.7 summarizes this chapter.

7.1. Conventional Scheduling Algorithms

The scheduling algorithm implemented in a grid workflow scheduler maps the tasks of a workflow to the appropriate grid resources in order to achieve a scheduling objective (e.g., to minimize the makespan of workflows, to maximize the resource utilization etc.).
Various scheduling algorithms have been presented to address the problem of grid workflow scheduling, however most of them are static scheduling algorithms as discussed in subsection 2.1.6. The static scheduling algorithms (see e.g., [Yu and Buyya, 2006; Yu, 2007; Wieczorek et al., 2007] for the classification) consider the following assumptions:

1. The computation time of the tasks of a workflow and the communication time between the tasks of the workflow with respect to a grid environment are known in advance and these computation time and communication time of the workflow are precise and accurate.

2. Most previous studies (see e.g., [Sakellariou and Zhao, 2004; Li, 2008]) assume the grid resources to be dedicated, i.e., without any LUs and hence the grid resources are always available with their full processing capacity to the tasks of workflows.

However, the dynamic grid environment contradicts with these assumptions. The accurate and precise estimation of the computation time and communication time of the tasks of a workflow is difficult because of the varying processing capabilities of the grid resources. Moreover, the grid resources are owned by different organizations and these organizations may have site-specific policies, e.g., the organizations may specify different resource sharing mechanisms for their grid resources as discussed in 2.1.2. It is therefore evident that conventional grid workflow scheduling approaches oversimplify the problem by taking unrealistic assumptions, and there is a need to develop the workflow scheduling algorithms for dynamic grid environments. Dynamic workflow scheduling algorithms are required in order to cope with the dynamicity of a grid environment some. Examples of such dynamic grid workflow scheduling algorithms are [Sakellariou and Zhao, 2004; Yu and Shi, 2007]. Prediction-based scheduling algorithms are the special case of the dynamic scheduling, which consider the current state and/or the historical states of a grid environment and predict the future state of a grid environment. One of the possible prediction parameters is the grid resources’ LUs’ load. The LUs’ load at a Grid Resource is defined as the product of the number of tasks at a Grid Resource from its LUs and average task length of a task from the LUs. A prediction-based grid workflow scheduling algorithm, which predicts the LUs’ load at the grid resources, can map the tasks of the workflows on the non-dedicated grid resources where the LUs’ load at the grid resources varies over the time. Therefore, a LUs’ load prediction-based grid workflow scheduling algorithm not only addresses the non-
dedicated nature of the grid resource but also considers the fluctuating load at the grid resources due to their LUs. In this chapter the grid workflow scheduling algorithms based on the LUs’ load prediction are presented. Next, the conceptual view of the LUs’ load prediction-based grid workflow scheduling is presented.

7.2. Conceptual View of Local User Load Prediction-based Grid Workflow Scheduling

Since the grid resources are subjected to the site-specific sharing policies, they are not always available with their full processing capabilities. In this situation a grid workflow scheduling algorithm need a prediction model to predict the load at the grid resources due to their LUs. Figure 7-1 depicts a conceptual view of a LUs’ load prediction-based grid workflow scheduling. The numbers on the arrows show the sequence of steps and these are explained next.

![Figure 7-1: Conceptual view of local user load prediction-based scheduling](image)

1. The activities performed in this step include:
   a. The Grid Resources register themselves with the GIS.
   b. The GIS assigns unique IDs to the Grid Resources.
   c. A Grid User submits a workflow to the Prediction-based Grid Workflow Scheduler.
d. The Local Users (LUs) at the Grid Resources may also submit their tasks to the Grid Resources.

The LUs’ Load Prediction-based Grid Workflow Scheduler (LULP-Grid Workflow Scheduler) consists of many components (i.e., the Scheduling Algorithm, the Dispatcher, and the Result Gatherer etc.) as explained in subsection 6.2.3. All these components except the Predictor are named as Grid Workflow Scheduler in Figure 7-1. The Predictor component is named as the LUs’ Load Predictor (LULP), which shows the implemented prediction model.

2. The activities involved in this step include gathering the Grid Resources’ information from the GIS and sending the results of the completed tasks of the LUs to the corresponding LUs. Following steps are performed for these activities:

   a. The LULP-Grid Workflow Scheduler requires the information like the available Grid Resources, the number of processors in each available Grid Resource, the number of tasks at the Grid Resources at a time etc. LULP-Grid Workflow Scheduler gathers this information from the GIS.

   b. A Grid Resource, after executing the tasks of its LUs, sends the results back to the corresponding LUs.

3. The information gathered in the step (2a) is used for generating the prediction parameters, which are the LUs’ loads (as defined in section 7.1) at the Grid Resources in this case.

4. The LULP-Grid Workflow Scheduler shortlists the Grid Resources according to the requirements of a workflow. These requirements include the operating system of the Grid Resource, the available free memory space at the Grid Resource, and number of free processors in a Grid Resource. The Grid Workflow Scheduler generates a schedule.

5. The Grid Workflow Scheduler sends the tasks of the workflow to the grid resources and receives the results of the completed tasks from the Grid Resources.

6. Following activities are performed in this step:
a. The results of the workflow’s tasks are sent to the LULP as well as to the other components of the LULP-Grid Workflow Scheduler. The LULP checks the accuracy of prediction parameter (i.e., the LUs’ load) and updates itself for the consequence predictions. The data files produced by the completed task of a workflow are saved on the grid resources so that these data files can be used later by some other tasks of the workflow.

b. The Grid Workflow Scheduler checks whether or not all the tasks of a workflow have been completed. The results of the workflow are sent to the corresponding Grid User when all the tasks of the workflow are completed.

The LULP component of a LULP-Grid Workflow Scheduler is explained in the next section.

7.3. Local Users’ Load Prediction (LULP)

A grid workflow scheduling algorithm tries to achieve a scheduling objective. The grid workflow scheduling begins with the discovery of available grid resources set. Then a subset of the grid resources fulfilling the requirements of a workflow (e.g., the number of processors, the memory space and the operating system required to execute the tasks of a workflow) is selected. The requirements of a workflow are associated with it at the time of its submission by the grid user. The tasks of the workflow are mapped on the selected resources by the workflow scheduling algorithm in order to achieve a certain objective, which is to achieve lower makespan in this case. The workflows’ tasks and the LUs’ tasks compete with each other to access a grid resource for their execution. The data files produced by the tasks of a workflow are needed to be saved on the grid resources so that the child tasks can use them. Achieving a scheduling objective, a lower makespan, is dependent on two factors, namely: mapping the tasks of the workflows on the grid resource having less competition by the tasks of the LUs as well as lowering the data movement time of the tasks of the workflows. Therefore, the scheduling objective, i.e., achieving a lower makespan of the workflows, can only be achieved when the grid workflow scheduler is able to predict the LUs’ loads of the grid resources and map the tasks of workflows to the grid resources in the presence of the LUs’ at grid resources and take the data movement time of the tasks into account. So that the load between resources is balanced and each grid resource finishes its execution at roughly the same time, thereby minimizing the execution
time. This type of distribution of workflows’ tasks to the grid resources can be done by assigning more tasks of workflows to more powerful grid resources. Such distribution of the workflows’ tasks may yield in high makespan as it does not take into account the LUs’ loads of the grid resources.

In this chapter the prediction-based workflow scheduling algorithms are presented, which takes into account the variation of the grid resources’ load due to their LUs. According to requirement R7 and requirement R8, a grid workflow scheduler must be able to predict the number of tasks and the task length of the tasks submitted to the grid resources. Therefore, the LULP provides the predicted LUs’ load at the grid resources to the grid workflow scheduler. The predicted LUs’ load is then used by the scheduling algorithm to map the tasks of a workflow to the grid resources.

In the prediction-based scheduling, more tasks of a workflow are assigned to a grid resource where these tasks are expected to complete in minimum time i.e., to by facing less competition due to LUs’ load. The prediction of the LUs’ load at a grid resource is required to adjust the amount of workload (the tasks of workflows) assignment to the grid resource. Therefore, the prediction-based grid workflow scheduling involves the following steps:

1. predicting the LUs’ load at the grid resources and
2. using the predicted LUs’ load to guide the workflow scheduling algorithm to find the suitable task to resource mappings for the tasks of the workflow such that the overall makespan of the workflow is low.

A solution for obtaining a lower makespan of a workflow is that the grid workflow scheduler may use a Service Level Agreement (SLA) so that the tasks may not suffer from the time-varying processing capabilities of grid resources due to their LUs’ load. Another solution is to use the historical data of the grid resources to predict the future processing capabilities of the grid resources. In this thesis the second solution is used for the prediction of the LUs’ loads at the grid resources.

Next, the important models required for grid workflow scheduling and the LUs’ load prediction are discussed. Table 7-1 provides a list of notations and their description, which are used in LULP.
### Table 7-1: List of notations

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>Workflow</td>
</tr>
<tr>
<td>$T$</td>
<td>Set of tasks in $G$</td>
</tr>
<tr>
<td>$t_i$</td>
<td>Task number $i$ of a workflow $G$</td>
</tr>
<tr>
<td>$E$</td>
<td>Set of Edges in $G$</td>
</tr>
<tr>
<td>$e_{i,j}$</td>
<td>Edge joining the tasks number $i$ and $j$</td>
</tr>
<tr>
<td>$W$</td>
<td>Set of weight of the task of $G$</td>
</tr>
<tr>
<td>$C$</td>
<td>Set of weight of the edges of $G$</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of total resources in Grid</td>
</tr>
<tr>
<td>$R$</td>
<td>Set of resources</td>
</tr>
<tr>
<td>$R_r$</td>
<td>Resource number $r$ of the Grid</td>
</tr>
<tr>
<td>$\gamma_r$</td>
<td>The number of processors in resource $r$</td>
</tr>
<tr>
<td>$\text{Speed}(P_r)$</td>
<td>Processing speed of a single processor of resource number $r$</td>
</tr>
<tr>
<td>$PC(R_r)$</td>
<td>Processing capacity of the grid resource number $r$</td>
</tr>
<tr>
<td>$\alpha_r$</td>
<td>Number of memory units in the grid resource number $r$</td>
</tr>
<tr>
<td>$\text{MemorySpace}(MS_{k,r})$</td>
<td>Memory capacity of the $k$th memory unit of the grid resource number $r$</td>
</tr>
<tr>
<td>$MC(R_r)$</td>
<td>Memory capacity of the grid resource number $r$</td>
</tr>
<tr>
<td>$TT_{k,r}$</td>
<td>Unit data transfer rate between the grid resource $k$ and $r$</td>
</tr>
<tr>
<td>$\text{Exe}(t_i, R_r)$</td>
<td>Execution time of task $t_i$ on grid resource $R_r$</td>
</tr>
<tr>
<td>$\text{DFMT}(t_{j,i}, R_{r,k})$</td>
<td>Data File movement time of the file generated by task $t_j$ for task $t_i$, which is saved on grid resource $r$ and is required on resource $r$</td>
</tr>
<tr>
<td>$\text{RT}(t_i, R_r)$</td>
<td>Ready time of the task $t_i$ on the grid resource $R_r$</td>
</tr>
<tr>
<td>$\text{RT}(R_r)$</td>
<td>Ready time of the grid resource $R_r$</td>
</tr>
<tr>
<td>$\Gamma$</td>
<td>Time interval</td>
</tr>
<tr>
<td>$\lambda_{t_i}$</td>
<td>The number of local tasks at a grid resource $R_r$ in interval $\Gamma$ (predicted)</td>
</tr>
<tr>
<td>$\mu_{t_i}$</td>
<td>Task length of a local task at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{PLL}(R_r)_{\Gamma}$</td>
<td>Predicted LUs' load at the grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{ECT}(t_i, R_r)_{\Gamma}$</td>
<td>Expected completion time of task $t_i$ on a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{ACT}(t_i, R_r)_{\Gamma}$</td>
<td>Actual completion time of the task $t_i$ on a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{LULP_RER}(t_i, R_r)_{\Gamma}$</td>
<td>LU load prediction relative error rate of task $t_i$ on a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{PAECT}(t_i, R_r)_{\Gamma}$</td>
<td>Percentage accuracy of expected completion time of task $t_i$ on a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{DCT}(t_i, R_r)_{\Gamma}$</td>
<td>Difference between the actual completion and expected completion time of task $t_i$ on a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
</tbody>
</table>
### 7.3 Local Users' Load Prediction (LULP)

**Actual LUs' load**
- at the grid resource $R_r$ in interval $\Gamma$
- number of local tasks at a grid resource $R_r$ in interval $\Gamma$
- task length of a local task at a grid resource $R_r$ in interval $\Gamma$

**Mean**
- number of local tasks at a grid resource $R_r$ in interval $\Gamma$
- task length of a local task at a grid resource $R_r$ in interval $\Gamma$

**Adaptation degree**
- for number of tasks from LUs
- for task length of a task from LU

**Change in direction**
- from increasing to decreasing or from decreasing to increasing in the number of task after the interval $\Gamma$
- in the task length of LUs' task after the interval $\Gamma$

**Increment (from predicted to actual)**
- number of tasks at a grid resource $R_r$ in interval $\Gamma$
- task length at a grid resource $R_r$ in interval $\Gamma$

**Decrement (from actual to the predicted)**
- number of tasks at a grid resource $R_r$ in interval $\Gamma$
- task length at a grid resource $R_r$ in interval $\Gamma$

---

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ALL(R_r)_{\Gamma}$</td>
<td>Actual LUs' load at the grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>actual$<em>{\lambda</em>{\Gamma, r}}$</td>
<td>Actual number of local tasks at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>actual$<em>{\mu</em>{\Gamma, r}}$</td>
<td>Actual task length of a local task at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$S$</td>
<td>Window size=5 in this case</td>
</tr>
<tr>
<td>Mean$<em>{\lambda</em>{\Gamma, r}}$</td>
<td>Mean number of local tasks at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>Mean$<em>{\mu</em>{\Gamma, r}}$</td>
<td>Mean task length of a local task at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$adaptDegree_{\lambda}$</td>
<td>Adaptation degree for number of tasks from LUs</td>
</tr>
<tr>
<td>$adaptDegree_{\mu}$</td>
<td>Adaptation degree for task length of a task from LU</td>
</tr>
<tr>
<td>$changeinDirection(\lambda_{\Gamma, r})$</td>
<td>Change in direction from increasing to decreasing or from decreasing to increasing in the number of task after the interval $\Gamma$</td>
</tr>
<tr>
<td>$changeinDirection(\mu_{\Gamma, r})$</td>
<td>Change in direction from increasing to decreasing or from decreasing to increasing in the task length of LUs' task after the interval $\Gamma$</td>
</tr>
<tr>
<td>$incrementValue(\lambda_{\Gamma, r}, \text{actual}<em>{\lambda</em>{\Gamma, r}})$</td>
<td>Increment from predicted to actual number task at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{realIncrementValue}(\text{actual}<em>{\lambda</em>{\Gamma, r}}, \text{actual}<em>{\lambda</em>{\Gamma-1, r}})$</td>
<td>Increment in the actual number task at a grid resource $R_r$ in interval $\Gamma$ from the interval $\Gamma - 1$</td>
</tr>
<tr>
<td>$incrementValue(\lambda_{\Gamma+1, r})$</td>
<td>Increment value in the number of task at the grid resource $R_r$ for the interval $\Gamma + 1$</td>
</tr>
<tr>
<td>$\text{changePointIncrement}(\lambda_{\Gamma, r})$</td>
<td>Change point increment value for the number of tasks at a grid resource $R_r$ after the interval $\Gamma$</td>
</tr>
<tr>
<td>$incrementValue(\mu_{\Gamma, r}, \text{actual}<em>{\mu</em>{\Gamma, r}})$</td>
<td>Increment from predicted to actual value of task length at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{realIncrementValue}(\text{actual}<em>{\mu</em>{\Gamma, r}}, \text{actual}<em>{\mu</em>{\Gamma-1, r}})$</td>
<td>Increment value of actual task length at a grid resource $R_r$ in interval $\Gamma$ from the interval $\Gamma - 1$</td>
</tr>
<tr>
<td>$incrementValue(\mu_{\Gamma+1, r})$</td>
<td>Increment value in the task length at the grid resource $R_r$ for the interval $\Gamma + 1$</td>
</tr>
<tr>
<td>$\text{changePointIncrement}(\mu_{\Gamma, r})$</td>
<td>Change point increment value for the task length of local task at a grid resource $R_r$ after the interval $\Gamma$</td>
</tr>
<tr>
<td>$decrementValue(\lambda_{\Gamma, r}, \text{actual}<em>{\lambda</em>{\Gamma, r}})$</td>
<td>Decrement from actual value to the predicted value in the number task at a grid resource $R_r$ in interval $\Gamma$</td>
</tr>
<tr>
<td>$\text{realDecrementValue}(\text{actual}<em>{\lambda</em>{\Gamma, r}}, \text{actual}<em>{\lambda</em>{\Gamma-1, r}})$</td>
<td>Decrement value in actual number task at a grid resource $R_r$ in interval $\Gamma$ from the interval $\Gamma - 1$</td>
</tr>
<tr>
<td>$decrementValue(\lambda_{\Gamma+1, r})$</td>
<td>Decrement value in the number of task at the grid resource $R_r$ for the interval $\Gamma + 1$</td>
</tr>
<tr>
<td>$decrementValue(\mu_{\Gamma, r}, \text{actual}<em>{\mu</em>{\Gamma, r}})$</td>
<td>Decrement from actual value to the</td>
</tr>
</tbody>
</table>
7.3.1. Workflow model

It is assumed that each workflow in the LULP-Grid Workflow Scheduler is represented as a directed acyclic workflow $G (T, E, W, C)$.

1. There is total $N$ number of tasks in a workflow. ‘$T$’ is the set of tasks and $t_i$ represents the $i^{th}$ task of set $T$ and $1 < i \leq N$.

2. $E$ is the set of edges. The directed edge $e_{i,j}$ joins tasks $t_i$ and $t_j$ and $1 < i, j \leq N$, where task $t_i$ is called the parent task and task $t_j$ is called the child task. This also implies that $t_j$ cannot start until task $t_i$ is completed and sends its data to the resource where $t_j$ has to start, $t_j \in \text{child}(t_i)$ where $\text{child}(t_i)$ is the set of all immediate child of $t_i$. Similarly, $t_i \in \text{parent}(t_j)$, whereas $\text{parent}(t_j)$ is the set of all immediate parents of $t_j$.

3. $W$ is the set of computations weights of the tasks of workflow, where $w_i$ is the computation weight of $t_i$.

4. $C$ is the set of communications weights of the tasks of workflow, where $c_{i,j}$ is the computation weight $e_{i,j}$. The cardinality of the set $C$ is denoted as $D$, where $D$ is a positive integer value.

7.3.2. Resource model

It is assumed that there are total ‘$M$’ resources available in the grid, denoted by the set ‘$R$’. ‘$R_r$’ represent the $r^{th}$ resource of the Grid where $1 \leq r \leq M$. 

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>realdecrementValue</td>
<td>predicted value of task length at a grid resource $R_r$</td>
</tr>
<tr>
<td>decrementValue</td>
<td>Decrement value of actual task length at a grid resource $R_r$ in interval $\Gamma$ from the interval $\Gamma - 1$</td>
</tr>
<tr>
<td>changePointdecrement</td>
<td>Change point decrement value for the number of tasks at a grid resource $R_r$ after the interval $\Gamma$</td>
</tr>
<tr>
<td>changePointdecrement</td>
<td>Change point decrement value for the task length of local task at a grid resource $R_r$ after the interval $\Gamma$</td>
</tr>
</tbody>
</table>
7.3 Local Users’ Load Prediction (LULP)

1. Each resource is a cluster.

2. Each resource $R_r$ has a definite number of processors denoted by $\gamma_r$.

3. All processors of a resource have equal processing speed.

4. The processing speed of a processor $P_r$ is denoted by $\text{Speed}(P_r)$, where $r$ is the resource number.

5. The processing capacity of the resource $R_r$ is given in Equation 7-1.

\[
PC(R_r) = \text{Speed}(P_r) \times \gamma_r
\]  \hspace{1cm} \text{Equation 7-1}

6. Each resource $R_r$ has a definite number of memory units present in it and are denoted by $\alpha_r$. The Memory space of $k$th memory unit of $r$th resource is denoted as $\text{MemorySpace}(MS_{k,r})$ where $1 \leq k \leq \alpha_r$.

7. The memory capacity of a resource $R_r$ is given in Equation 7-2.

\[
MC(R_r) = \sum_{k=1}^{\alpha_r} \text{MemorySpace}(MS_{k,r})
\]  \hspace{1cm} \text{Equation 7-2}

8. It is assumed that the grid resources have two types of network connections, local network connections within a cluster and internet connections. Communications through internet across the different resources is considered and the communication within a cluster is assumed negligible.

9. The latencies and the bandwidths across the grid resources are denoted as ‘$L$’, ‘$B$’ respectively and both are $M \times M$ matrices. It is assumed that the values of $L$ and $B$ are known. TT is also a $M \times M$ matrix. The value of $TT_{k,r}$ is calculated using $B_{k,r}$ and it represents the unit data transfer rate between the resources $R_k$ and $R_r$. The value of $TT_{k,r} = 0$ when $k = r$.

7.3.3. Workflow task execution time and data file movement time

Each task of a workflow is allowed to execute at only one resource and at least one free processor of the resource is required to execute it. The execution time of a task $t_i$ on a resource $R_r$ is denoted as $\text{Exe}(t_i, R_r)$ and it can be obtained by dividing its computational weight on the processing speed of a single processor of the resource $R_r$. It is given in Equation 7-3.
\[ Exe(t_i, R_r) = \frac{w_i}{\text{Speed}(P_r)} \]  

\[ \text{Equation 7-3} \]

The data file movement time between a parent task \( t_j \) and a child task \( t_i \), when the data file generated by \( t_j \) is saved on resource \( R_r \) and \( t_i \) is scheduled to the resource \( R_k \) is represented as \( DFMT(t_{ji}, R_{rk}) \) it is calculated by Equation 7-4.

\[ DFMT(t_{ji}, R_{rk}) = \begin{cases} 0 & \text{when } k = r \\ L_{kr} + (c_{ji} \times TT_{kr}) & \text{otherwise} \end{cases} \]  

\[ \text{Equation 7-4} \]

### 7.3.4. Ready time of a task and resource

Ready time of a task \( t_i \) on the resource \( R_r \) is represented as \( RT(t_i, R_r) \) and it is zero when \( t_i \) is an entry task (a task having no parent), otherwise it is the maximum of the data file movement times of all of its parents. E.g., if a task \( t_i \) has \( 'p' \) parents and the task \( t_i \) is scheduled to resource \( R_r \) in this case the ready time of the task is given in Equation 7-5. Here \( 1 \leq k, q, m, r \leq M \) and \( p \leq N \). Data file movement time is calculated according to Equation 7-4.

\[ RT(t_i, R_r) = \begin{cases} 0 & \text{when } t_i \text{ is an entry task} \\ \text{Max}(DFMT(t_{1i}, R_{kr}), ..., DFMT(t_{pi}, R_{kr}), ..., DFMT(t_{pi}, R_{rm})) & \text{otherwise} \end{cases} \]  

\[ \text{Equation 7-5} \]

Ready time of a resource is zero when a resource can provide at least one processor to the task of a workflow application. If all the processors of a resource are busy then it is the free time of the processor, which earliest finishes the task currently in execution. The ready time of resource is represented as ‘RT (\( R_r \)’ and it is given in Equation 7-6. Here \( Exe_1, Exe_2 \) show the execution times of tasks at processors 1 and 2 respectively.

\[ RT(R_r) = \begin{cases} 0 & \text{when } R_r \text{ has atleast one processor free} \\ \text{Min}(Exe_1, Exe_2 \ldots Exe_{r_r}) & \text{otherwise} \end{cases} \]  

\[ \text{Equation 7-6} \]

### 7.3.5. LUs’ load prediction at the grid resource

For the LUs’ loads, it is assumed that each resource \( R_r \) of the grid has a number of local task which is a random number generated by Poisson distribution. Let us consider the LUs’ tasks
7.3 Local Users’ Load Prediction (LULP)

arrival at a grid resource as a count point process. The \( x_{\Gamma-1,r}, x_{\Gamma,r} \) and \( x_{\Gamma+1,r} \) shows the random number of tasks in the interval \( \Gamma - 1, \Gamma \) and \( \Gamma + 1 \) on the grid resource \( R_r \) and \( \lambda_{\Gamma,r} \) is the number of local tasks at the grid resource \( R_r \) in the interval \( \Gamma \). Equation 7-7 gives the probability that the value of \( x_{\Gamma,r} = x \). The value of \( x_{\Gamma,r} \) is then generated according to Algorithm 7-1.

\[
f(x_{\Gamma,r} = x) = \frac{(\lambda_{\Gamma,r})^x e^{-\lambda_{\Gamma,r}}}{x!} \quad \text{Equation 7-7}
\]

Algorithm 7-1: Poisson variant generation

1. PROCEDURE poissonSimulation(\( \lambda_{\Gamma,r} \)) {
2. \hspace{1em} SET \( p := 1 \), \( L := e^{-\lambda_{\Gamma,r}} \), \( k := 0 \)
3. \hspace{1em} DO {
4. \hspace{2em} Increase \( k \) by 1
5. \hspace{2em} Generate an independent random variable \( U_k \approx U(0,1) \)
6. \hspace{2em} SET \( p := p \times U_k \)
7. \hspace{1em} } WHILE \( (p > L) \)
8. \hspace{1em} SET \( x := k - 1 \)
9. \hspace{1em} RETURN \( x \)
10. }

Similarly it is assumed that the average task length of a task from the LUs is measured in Millions of Instruction (MI) and follows a Poisson distribution. The average task length of the tasks from the LUs at grid resource \( R_r \) in the interval \( \Gamma \) is considered as a variant of \( \mu_{\Gamma,r} \). The number \( y_{\Gamma-1,r}, y_{\Gamma,r} \) and \( y_{\Gamma+1,r} \) shows the random number representing the task length of a task from the LUs on the grid resource \( R_r \) in the interval \( \Gamma - 1, \Gamma \) and \( \Gamma + 1 \). The value of \( y_{\Gamma,r} \) can also be calculated by using the Algorithm 7-1. The predicted LUs’ load at a grid resource \( R_r \) in an interval \( \Gamma \) is denoted as \( PLL(R_r)_\Gamma \) and it is a product of \( x_{\Gamma,r} \) and \( y_{\Gamma,r} \). \( PLL(R_r)_\Gamma \) can be calculated according to Equation 7-8.

\[
PLL(R_r)_\Gamma = x_{\Gamma,r} \times y_{\Gamma,r} \quad \text{Equation 7-8}
\]

The predictor requires to be executed for a certain number of times as a training period so that the observations can be fixed. In this case, the values of \( \lambda_{\Gamma,r} \) and \( \mu_{\Gamma,r} \) are required as a startup point for LULP which are obtained from the training period. A predictor may be trained at runtime i.e., during the simulation of an experiment but in this case the extra
overhead of training time is incurred. Therefore the proposed LULP was executed such that for each grid resource 100 history measurements are available. Using this history information the LUs’ load at the grid resource is predicted and it is then used to guide the grid workflow scheduling algorithm for mapping the tasks of the workflows to the grid resource.

The LUs’ load at the grid resource is predicted by using the time series prediction method. The time series-based prediction models are used by various scientific communities examples are earth and ocean science, biomedical, signal processing and networking. The characteristics of the time series used in a prediction model classify a prediction model into linear or non-linear prediction model [Box and Jenkins, 1990; Mitsa, 2010]. The linear prediction models such as Moving Average (MA) model, Auto Regression (AR) model and Auto Regression Moving Average (ARMA) model are widely used due to their computational efficiency [Box and Jenkins, 1990]. Such, linear models are used for predicting a future measurement by using the fixed number of immediate historical data measurements, which is referred as window size [Box and Jenkins, 1990; Mitsa, 2010]. Usually the method of learning through historical information is used to predict the prediction parameter. The time period for which the historical information is analyzed plays important role in prediction. The history data of a long period of time longer time gives the good estimates (i.e., less error rate) as well as the trend followed by of a prediction parameter in the series can be accurately projected.

In order to predict the LUs’ load at a grid resource the MA-based model is selected because, it is simple and it is computational efficient and it is placed in the group of prediction methods which have nearly no overhead [Herbst et al., 2013]. However instead of treating all the historical measurements same as the conventional MA-based model does the proposed MA-based LUs’ load prediction model (i.e., LULP) gives more weights to the recent history. Giving more weight to the more recent history is important as more far-off a history measurement lesser is its impact on future state. The traditional MA model are not able to measure the tendency of a future measurement, however the proposed MA-based prediction model can also be used to predict the future tendency of the LUs’ load at the grid resources (i.e., the future load will be increased or decreased at a grid resource). The predicted LUs’ load is then used to guide the grid workflow scheduling algorithm to calculate
the expected completion time of the task on the grid resources. The weighting of the most recent history and the adaptation process in case of deviation of actual values of the load factor are discussed in detail in subsection 7.3.8.

7.3.6. Expected completion time of a task

The expected completion time of a task $t_i$ on a resource $R_r$ in the time interval $\Gamma$ is denoted as $ECT(t_i, R_r)_{\Gamma}$. It is the sum of its execution time and the execution time of the LUs’ load at a grid resource. If a task $t_i$ of a workflow is mapped on a resource $R_r$, its expected completion time is calculated according to Equation 7-9.

$$ECT(t_i, R_r)_{\Gamma} = \max\{RT(t_i, R_r), RT(R_r)\} + Exe(t_i, R_r) + \frac{PLL(R_r)_{\Gamma}}{PC(R_r)}$$  \hspace{1cm} Equation 7-9

7.3.7. Checking error rate and measuring accuracy

A workflow task $t_i$ which was scheduled to a resource $R_r$ in the time interval $\Gamma$ completes its execution and the results of the task are received by the LULP-Grid Workflow Scheduler. The results are used to measure the relative error rate of the expected completion time of the task. Let the actual completion time is denoted as $ACT(t_i, R_r)_{\Gamma}$. The LULP-relative error rate is denoted as $LULP_{-RER}$. The error rate is calculated by finding the difference between the actual completion time of a task and the expected completion time of the task and dividing it on the actual completion time of the task as shown in Equation 7-10.

$$LULP_{-RER}(t_i, R_r)_{\Gamma} = \frac{abs(ACT(t_i, R_r)_{\Gamma}) - ECT(t_i, R_r)_{\Gamma}}{ACT(t_i, R_r)_{\Gamma}}$$ \hspace{1cm} Equation 7-10

This relative error rate is then used to measure the percentage of accuracy of the expected completion time of the task according to Equation 7-11.

$$PAECT(t_i, R_r)_{\Gamma} = (1 - LULP_{-RER}ACT(t_i, R_r)_{\Gamma}) \times 100$$ \hspace{1cm} Equation 7-11

7.3.8. Adaptation process of the LULP

The task assigned to a grid resource when completes its execution, the grid resource sends the results of the task to the corresponding LULP-Grid Workflow Scheduler. The actual completion time of the task $t_i$ of a workflow which has completed its execution on the grid
resource \( R_{r} \) in the interval \( \Gamma \) and it is denoted as \( ACT(t_{i}, R_{r})_{\Gamma} \). The difference between the expected completion time and the actual completion time of the task is denoted as \( DCT(t_{i}, R_{r})_{\Gamma} \) and it can be calculated according to Equation 7-12.

\[
DCT(t_{i}, R_{r})_{\Gamma} = ACT(t_{i}, R_{r})_{\Gamma} - ECT(t_{i}, R_{r})_{\Gamma}
\]  

Equation 7-12

It is assumed that if the task completes in less than its expected completion time, the execution time of LUs’ tasks was less on the resource. Conversely, if the task has taken more time than the expected completion time this means, that the execution time of the LUs’ tasks was more than the expected execution time of the LUs’ tasks on the resource. The sum of the \( DCT \) values of all tasks assigned to a grid resource \( R_{r} \) in the interval \( \Gamma \) is then used to predict the LUs’ load at the grid resource for the interval \( \Gamma + 1 \). Let the total tasks of the workflows submitted to the grid resource \( R_{r} \) in the interval \( \Gamma \) is a positive integer \( Q \). The sum of the \( DCT \) values for all the tasks submitted to a grid resource in the time interval \( \Gamma \) is the difference between the actual execution time of the local tasks and the predicted execution time of the local tasks at the grid resource \( R_{r} \) in the interval \( \Gamma \). In order to calculate the difference between the actual load and predicted load at the grid resource, the sum of the \( DCT \) value is multiplied by the processing capacity of the grid resource \( R_{r} \). Using the simple algebra the value of the average actual LUs’ load at the grid resource for the interval \( \Gamma \) is obtained and it is given in Equation 7-13.

\[
\left[ \frac{ALL(R_{r})_{\Gamma}}{PC(R_{r})} - \frac{PLL(R_{r})_{\Gamma}}{PC(R_{r})} \right] \times Q = \left[ \sum DCT(t_{i}, R_{r})_{\Gamma} \right]
\]

Where \( Q \) is the total number of tasks submitted to \( R_{r} \) in the interval \( \Gamma \)

\[
ALL(R_{r})_{\Gamma} = \frac{\left[ \sum DCT(t_{i}, R_{r})_{\Gamma} \right] \times PC(R_{r})}{Q} + PLL(R_{r})_{\Gamma}
\]  

Equation 7-13

Similar to the predicted LUs’ load at a grid resource, the actual LUs’ load has also two factors, namely: the actual number of tasks from the LUs, which is denoted by \( \text{actual}_{\lambda_{r}, r} \) and an actual average task length of a task from the LUs, which is denoted by \( \text{actual}_{\mu_{r}, r} \). The difference between \( \text{actual}_{\lambda_{r}, r} \) and \( \lambda_{r, r} \), and the difference between \( \text{actual}_{\mu_{r}, r} \) and \( \mu_{r, r} \) is needed to be calculated and values of \( \lambda_{r+1, r} \) and \( \mu_{r+1, r} \) are updated.
accordingly which are then used to predict the LUs’ load at the grid resource \( R_r \) for the interval \( \Gamma + 1 \).

Since the information about the LUs’ load at the grid resources is not available, it is required to be predicted. The available information about the grid resources includes:

1. The number of tasks at the grid resource at a time.
2. The number of processors in the grid resource.
3. The processing speed of the grid resource.
4. The number of free processors at the grid resource etc.

This information about the grid resource is obtained from the GIS, where GIS periodically updates this information. LULP at a Grid Workflow Scheduler periodically contacts with GIS to get the information about the number of tasks at the grid resources. The GIS gives a value of \( \text{currentTasks}(R_r) \) to a LULP-Grid Workflow Scheduler and the LULP calculates the current local tasks by subtracting the number of tasks of the LULP-Grid Workflow Scheduler currently on the grid resource. LULP then calculated the mean LUs’ tasks at a grid resource at the end of an interval. E.g. the number of tasks from a LULP-Grid Workflow Scheduler at a point of time to a grid resource \( R_r \) is equal to 5 and GIS informs that the number of tasks at the grid resource \( R_r \) is 12. At that point of time the LULP calculates the value of actual local tasks by subtracting the number of tasks of LULP-Grid Workflow Scheduler at the said grid resource from the number of tasks given by the GIS i.e. the local tasks at the grid resource at that point is equal to \( (12-5=7) \). At the end of the interval \( \Gamma \) the average value of the number of tasks in the interval is calculated and it is named as actual_\( \lambda_{\Gamma,r} \). The average actual LUs’ load at the grid resource \( R_r \) in the interval \( \Gamma \) is denoted as \( \text{ALL}(R_r)_\Gamma \) and it is the product of average actual_\( \lambda_{\Gamma,r} \) and actual_\( \mu_{\Gamma,r} \). Substituting the value \( \text{ALL}(R_r)_\Gamma \) in terms of actual_\( \lambda_{\Gamma,r} \) and actual_\( \mu_{\Gamma,r} \) in Equation 7-13, and using the simple algebra the value of actual_\( \mu_{\Gamma,r} \) is then given according to Equation 7-14.

\[
\text{actual}_\mu_{\Gamma,r} = \frac{\left[ \sum DCT(t_i,R_r)_{\Gamma} \right] \times PC(R_r) - Q \times PLL(R_r)_{\Gamma}}{Q \times \text{actual}_\lambda_{\Gamma,r}} \tag{Equation 7-14}
\]

After getting the individual values of average actual_\( \lambda_{\Gamma,r} \) and actual_\( \mu_{\Gamma,r} \). These values are checked against values used for prediction (i.e., \( \lambda_{\Gamma,r} \) and \( \mu_{\Gamma,r} \)), and it is decided that weather
\( \lambda_{\Gamma+1,r} \) and \( \mu_{\Gamma+1,r} \) is need to be increased or decreased. The mean values \( \lambda_{\Gamma,r} \) and \( \mu_{\Gamma,r} \) are denoted as mean_\( \lambda_{\Gamma,r} \) and mean_\( \mu_{\Gamma,r} \) and these can be calculated according to Equation 7-15 and Equation 7-16. Here \( S \) is the window size, and it is the number of previous intervals used to calculate the mean values.

\[
\text{Mean}_\lambda_{\Gamma,r} = \frac{\sum_{l=1-S}^{\Gamma-1} \text{actual}_\lambda_{l,r}}{S} \quad \text{where} \ l \in [\Gamma - S, \Gamma - 1]
\]

\[
\text{Mean}_\mu_{\Gamma,r} = \frac{\sum_{l=1-S}^{\Gamma-1} \text{actual}_\mu_{l,r}}{S} \quad \text{where} \ l \in [\Gamma - S, \Gamma - 1]
\]

In the conventional MA-based prediction model the next predicted value is the mean value of the history measurement of window size. However, the proposed MA-based prediction model uses the mean values of the load factors as a threshold and a most recent history is assigned a weight, the proposed prediction model also determines the trend followed by the series of a load factor. The increment and decrement in an actual value follow the concept defined by [YANG, 2007], however the LULP prediction model is different than that of the [YANG, 2007] as it is used to predict the LUs’ load at the grid resources for an interval, and the actual values of the load factors of 3 consecutive intervals are used to predict the trend followed by the time series of a load factor. The Equation 7-17 and Equation 7-18 show the decreasing trend of the individual LUs’ load factor and Equation 7-19 and Equation 7-20 show the increasing trend of the LUs’ load factors.

\[
\text{IF} \quad ((\text{actual}_\lambda_{\Gamma,r} - \text{actual}_\lambda_{\Gamma-1,r} < 0) \ AND \ (\text{actual}_\lambda_{\Gamma-1,r} - \text{actual}_\lambda_{\Gamma-2,r} < 0) \ AND \ (\text{actual}_\lambda_{\Gamma-2,r} - \text{actual}_\lambda_{\Gamma-3,r} < 0)) \rightarrow \lambda_{\Gamma+1,r} \ at \ R_r \ will \ decrease \ for \ time \ interval \ \Gamma + 1
\]

\[
\text{IF} \quad ((\text{actual}_\mu_{\Gamma,r} - \text{actual}_\mu_{\Gamma-1,r} < 0) \ AND \ (\text{actual}_\mu_{\Gamma-1,r} - \text{actual}_\mu_{\Gamma-2,r} < 0) \ AND \ (\text{actual}_\mu_{\Gamma-2,r} - \text{actual}_\mu_{\Gamma-3,r} < 0)) \rightarrow \mu_{\Gamma+1,r} \ at \ R_r \ will \ decrease \ for \ time \ interval \ \Gamma + 1
\]
In history of the LULP always the increment or decrement values are saved. The increment value calculation procedure for $\lambda_{r,\Gamma}$ is discussed next in this section and $\mu_{r,\Gamma}$ uses the same procedure for increment. An increment value used in the $\lambda_{r,\Gamma}$ (which is a predicted value for the interval $\Gamma$) is represented as $\text{incrementValue}(\lambda_{r,\Gamma}, \text{actual } \lambda_{r,\Gamma})$ it is calculated according to Equation 7-21. The difference between the values $\text{actual } \lambda_{r,\Gamma}$ and $\lambda_{r,\Gamma-1}$ of the grid resource $R_r$ gives the value increased from previous interval it is denoted as $\text{realIncrementValue}(\text{actual } \lambda_{r,\Gamma}, \text{actual } \lambda_{r,\Gamma-1})$ and it can be calculated according to Equation 7-22.

$$\text{incrementValue}(\lambda_{r,\Gamma}, \text{actual } \lambda_{r,\Gamma}) = \lambda_{r,\Gamma} - \text{actual } \lambda_{r,\Gamma}$$  \hspace{1cm} \text{Equation 7-21}

$$\text{realIncrementValue}(\text{actual } \lambda_{r,\Gamma}, \text{actual } \lambda_{r,\Gamma-1}) = \text{actual } \lambda_{r,\Gamma} - \text{actual } \lambda_{r,\Gamma-1}$$  \hspace{1cm} \text{Equation 7-22}

If the value of a LUs’ load factor used to calculate the LUs’ load is less than its actual value, then the value used for prediction must be increased for the upcoming interval. The question that with which extent the increment should be made is needed to be answered. In Equation 7-23 the value of increment for the load factor $\lambda_r$ for a grid resource $R_r$ is given, where $\text{adaptDegree}_\lambda$ is a weight showing the impact of the last observation on the next prediction value. From the training period it is suggested that the value of $\text{adaptDegree}_\lambda$ should be 0.6 and for $\mu_{r,\Gamma}$ the value of $\text{adaptDegree}_\mu$ is equal to 0.5 for all the resources. The value of the adapt degree of the LUs’ load factors can lie in the range of 0.0 and 1. If it is
assigned the value 0.0 this means that the value of the load factor is increased by the difference between the actual and predicted value. If the value of adapt degree is assigned a value equal to 1 this means the load factor is increased by the difference of its actual value of two consecutive intervals. There are two different values of adapt degree as the LUs’ load at a grid resource is considered as a product and moreover, both the factors may increase or decrease independently. The value of adaptDegree is equal to 0.6 because during the training period of LULP it is observed that the number of tasks from the LUs at the grid resources neither increase nor decrease steadily nor the values fluctuated too much. In [Kavitha and Sankaranarayanan, 2013] used three different values of adapt degree, (0.3, 0.6, 0.9), and [YANG, 2007] used a value of 0.5 for adapt degree. In [YANG, 2007] it is proved that the MA-based prediction model using the weighting scheme outperformed the 9 conventional prediction methods including the conventional MA-based prediction method.

\[
\text{incrementValue}(\lambda_r) = \text{incrementValue}(\lambda_{r, \text{actual}}, \lambda_{r, \text{actual}}) + (\text{realIncrementValue}(\lambda_{r, \text{actual}}, \lambda_{r-1, \text{actual}}) - \text{incrementValue}(\lambda_{r, \text{actual}}, \lambda_{r, \text{actual}})) \times \text{adaptDegree}\lambda
\]

Equation 7-23

If there is a huge difference between the actual value and the predicted value of a load factor than there are two possibilities.

1. The trend followed in the previous intervals will be not followed by the upcoming interval and the next interval is a change point in the series.

2. The trend will remain for the upcoming interval and the value of increment should be adjusted such that it does not change the direction of the time series.

The actual value of a LUs’ load factor is checked against a threshold value. The threshold value of a LUs’ load factor is the mean of that factor. The mean values of the LUs’ load factors can be calculated according to according to Equation 7-15 and Equation 7-16. If the actual value of a LUs’ load factor is less than its mean value than it will remain in the same direction as the previous intervals were. Otherwise this may be a change point in the series. The value of a load factor used for predicting LUs’ load for the interval \(\Gamma\) is checked against its mean value of the interval \(\Gamma\) if it is less than its mean value than the increment value will
be same as defined in Equation 7-23. Otherwise in else part it is checked whether the series is changing its direction or it will follow the same trend of increasing as the previous intervals’ values followed. To check that whether the time series is changing its direction after the time interval $\Gamma$ following steps are followed.

1. The product of the difference between the consecutive predicted values and the differences between the consecutive actual values of a LUs’ load factor for all intervals in the window size is assigned a value of 0 and 1 according to the conditions stated in the Else part after the Equation 7-24.

2. All the 0 or 1 are summed up and are divided by the window size.

3. This value is then said to be $\text{changeinDirection}(\lambda_{l,r})$.

4. The value for the change point increment is then calculated by multiplying the increment value for that factor in the interval $\Gamma$.

5. Increment value for the interval $\Gamma + 1$ is assigned a value, which is minimum of $\text{incrementvalue}$ and $\text{changePointIncrement}$ of the factor.

\[
\begin{align*}
\text{IF} (\text{actual}_{\Gamma,r} < \text{Mean}_{\Gamma,r}) & \quad \text{incrementValue}(\lambda_{\Gamma+1,r}) = \text{incrementValue}(\lambda_{\Gamma}) \\
\text{ELSE} & \\
\text{changeinDirection}(\lambda_{\Gamma,r}) & = \sum_{S}^{\Gamma-2} (\lambda_{l,r} - \lambda_{l-1,r})(\text{actual}_{\lambda_{l,r}} - \text{actual}_{\lambda_{l-1,r}}) \\
& \quad \text{where } l \in [\Gamma - S, \Gamma - 1] \text{ and } \text{actual}_{\lambda_{l,r}} > \text{actual}_{\lambda_{l-1,r}} \text{ and} \\
& \quad \text{IF } ((\lambda_{l,r} - \lambda_{l-1,r})(\text{actual}_{\lambda_{l,r}} - \text{actual}_{\lambda_{l-1,r}}) > 0) \\
& \quad \quad (\lambda_{l,r} - \lambda_{l-1,r})(\text{actual}_{\lambda_{l,r}} - \text{actual}_{\lambda_{l-1,r}}) = 1 \\
& \quad \quad \text{ELSE} \\
& \quad \quad (\lambda_{l,r} - \lambda_{l-1,r})(\text{actual}_{\lambda_{l,r}} - \text{actual}_{\lambda_{l-1,r}}) = 0 \\
\text{changePointIncrement}(\lambda_{\Gamma,r}) & = \text{incrementValue}(\lambda_{\Gamma,r}, \text{actual}_{\lambda_{\Gamma,r}}) \\
& \quad \times \text{changeinDirection}(\lambda_{\Gamma,r}) \\
\text{incrementValue}(\lambda_{\Gamma+1,r}) & = \text{Min}(\text{abs}(\text{incrementValue}(\lambda_{\Gamma})), \\
& \quad \text{abs}(\text{changePointIncrement}(\lambda_{\Gamma,r})))
\end{align*}
\]
If the value of a LUs’ load factor used for predicting the LUs’ load at a grid resource is greater than its actual value, it is need to be decreased. The calculations for $\mu_{\Gamma,r}$ are shown for decrement procedure and for $\lambda_{\Gamma,r}$ these are same.

$$\text{decrementValue}(\mu_{\Gamma,r}, \text{actual}_\mu_{\Gamma,r}) = \text{actual}_\mu_{\Gamma,r} - \mu_{\Gamma,r}$$  \hspace{1cm} \text{Equation 7-26}

$$\text{realdecrementValue}(\text{actual}_\mu_{\Gamma,r}, \text{actual}_\mu_{\Gamma-1,r}) = \text{actual}_\mu_{\Gamma-1,r} - \text{actual}_\mu_{\Gamma,r}$$  \hspace{1cm} \text{Equation 7-27}

Like the increment value the decrement value of each factor of the LUs’ load is saved in the history the calculation of decrement value for the load factor $\mu_{\Gamma,r}$ of a grid resource $R_r$ is shown in Equation 7-28, where the $\text{decrementValue}(\mu_{\Gamma,r})$ and $\text{realdecrementValue}(\mu_{\Gamma,r})$ can be calculated according to Equation 7-26 and Equation 7-27 respectively.

$$\text{decrementValue}(\mu_{r}) = \text{decrementValue}(\mu_{\Gamma,r}, \text{actual}_\mu_{\Gamma,r}) + (\text{realdecrementValue}(\text{actual}_\mu_{\Gamma,r}, \text{actual}_\mu_{\Gamma-1,r}) - \text{decrementValue}(\mu_{\Gamma,r}, \text{actual}_\mu_{\Gamma,r})) \times \text{adaptDegree}$$  \hspace{1cm} \text{Equation 7-28}

Like an increasing trend followed by a LUs’ load factor, the decreasing trend of the LUs’ load factor is also preserved by checking the value of a load factor actually incurred for an interval and the mean value of the load factor for the interval. If the actual value of a LUs’ load factor is greater than its mean value the decrement value is same as defined in Equation 7-28. Otherwise it may be a changing point and in the else part all the intervals in the window size are checked and the decrement value is assigned a minimum value between the normal decrement value and change point decrement.
IF \( (\text{actual}_{\mu_{\Gamma,r}} > \text{Mean}_{\mu_{\Gamma,r}}) \)

\[
decrementValue(\mu_{\Gamma+1,r}) = \text{decrementValue}(\mu_{r})
\]

ELSE

\[
\text{changeinDirection}(\mu_{\Gamma,r}) = \left( \sum_{l=\Gamma-S}^{\Gamma-2} (\mu_{l,r} - \mu_{l-1,r}) (\text{actual}_{\mu_{l,r}} - \text{actual}_{\mu_{l-1,r}}) \right) / S
\]

where \( l \in [\Gamma - S, \Gamma - 1] \) and \( \text{actual}_{\mu_{l,r}} < \text{actual}_{\mu_{\Gamma,r}} \) and

IF \((\mu_{l,r} - \mu_{l-1,r})(\text{actual}_{\mu_{l,r}} - \text{actual}_{\mu_{l-1,r}}) > 0)\)

\((\mu_{l,r} - \mu_{l-1,r})(\text{actual}_{\mu_{l,r}} - \text{actual}_{\mu_{l-1,r}}) = 1\)

ELSE

\((\mu_{l,r} - \mu_{l-1,r})(\text{actual}_{\mu_{l,r}} - \text{actual}_{\mu_{l-1,r}}) = 0\)

\[
\text{changePointdecrement}(\mu_{\Gamma,r}) = \text{decrementValue}(\mu_{r}) \times \text{changeinDirection}(R_{r})_{\Gamma}
\]

\[
\text{decrementValue}(\mu_{\Gamma+1,r}) = \min(\text{abs}(\text{decrementValue}(\mu_{r})), \text{abs}(\text{changePointdecrement}(\mu_{\Gamma,r})))
\]

The LUs’ load factors used to predict an upcoming interval is then calculated according to Equation 7-31 and Equation 7-32. The LUs’ load for the upcoming interval is calculated using the Equation 7-8.

\[
\mu_{\Gamma+1,r} = \text{actual}_{\mu_{\Gamma,r}} + \text{incrementValue}(\mu_{\Gamma+1,r}) \quad \text{Or}\]
\[
\mu_{\Gamma+1,r} = \text{actual}_{\mu_{\Gamma,r}} - \text{decrementValue}(\mu_{\Gamma+1,r})
\]

\[
\lambda_{\Gamma+1,r} = \text{actual}_{\lambda_{\Gamma,r}} + \text{incrementValue}(\lambda_{\Gamma+1,r}) \quad \text{Or}\]
\[
\lambda_{\Gamma+1,r} = \text{actual}_{\lambda_{\Gamma,r}} - \text{decrementValue}(\lambda_{\Gamma+1,r})
\]

### 7.4. Data and Task Co-Scheduling

In the static workflow scheduling algorithms it is considered that the data files required by the child task of a workflow are already present on the grid resource where the task has to start its execution. However, a workflow’ task cannot get ready for execution until all of its parent tasks complete their executions. Some of the parent tasks of a workflow’s task may have been completed but some of the parent tasks may be still in execution or even not
scheduled. In this case the data files produced by the parent tasks are needed to be saved on the grid resources so that these data files can be used later by the child task.

The grid workflow scheduling algorithm taking a decision to map a task of a workflow to a grid resource must take into account the data movement time as it affects the expected completion time of the task as shown in Equation 7-9. When a task completes its execution the data files produced by the task are saved and the data files consumed by the task are deleted. The data files consumed by a task must be deleted to make the space for the newly generated data files. In order to address (Requirement R12) a data saving algorithm is proposed to save the data files generated by a task of workflow in upcoming discussion.

This algorithm saves the data files on the grid resources such that the data movement time of the child task is less. This is possible only when most of the child tasks are mapped to the resource where the data files required by the child tasks are available. The proposed algorithm tries to save the data files in such way that a data file having biggest size is saved on the parent task resource. It is assumed that a resource where the parent task has completed its execution is most favorite resource while making a child task to grid resource mapping as the expected completion time of a task on resource is low where most of the required files are available. This concept is known as the data and task co-scheduling. Next the data saving algorithm is presented in Algorithm 7-2, which uses the Algorithm 7-3 to save a given file on a given resource identification number (id).
The data files of the workflow's tasks are categorized into 1. Data files produced and 2. Data files consumed. The list of data files produced by a task $t_i$ for its child tasks are denoted as $DFP(t_i)$ and the data files consumed list of task $t_i$ is represented as $DFC(t_i)$. The size of $DFC(t_i)$ when $t_i$ is entry task is zero and the size of $DFP(t_i)$ is zero if the task $t_i$ is an exit node. In the algorithm 7-2 it is shown that the list of data files produced $DFP(t_i)$ by the task
is rearranged in descending order. Two variables \( s \) and \( \text{not_this_location} \) are created as instances of integer and List respectively. The variable \( s \) is used to identify the number of file in \( DFP(t_i) \) list, which is going to be saved in next step. The list \( \text{not_this_location} \) has the resources IDs, where the data file at index \( s \) cannot be saved due unavailability of the storage space. Algorithm7-3 is called for saving the data file on a grid resource specified by the \( \text{resourceID} \). The Algorithm7-3 returns a TRUE or FALSE depicting that the file saving was successful or unsuccessful. If the data file is successfully saved on the grid resource than the index \( s \) is incremented by 1 and next file saving procedure is carried out. If the data file saving was not successful than the grid resource specified by the \( \text{resourceID} \), then \( \text{resourceID} \) is added into the list \( \text{not_this_location} \). The data file on index \( s \) is tried to save on another resource such that the data movement time is low. This process continues until all the data files produced by the task are saved on the grid resources.

7.5. LUs’ Load Prediction-based Grid workflow Scheduling Algorithms

Almost all the scheduling algorithms somehow depend on prediction e.g., the Min-Min schedules the task of minimum length to a resource where it can complete with minimum time. So tasks’ completion times are predicted on the resources. In order to address the requirement (Requirement \( R10 \)) the scheduling algorithms are proposed which takes into account the fluctuating workload of the grid resources due to their LUs and schedule the task of workflow to the grid resource where it can complete in minimum time. The prediction model described above is used in Min-Min, Max-Min and Sufferage algorithms for the workflow scheduling. In other words minimum task length, maximum task length and highest Sufferage value are used to prioritize the tasks of a workflow in the ready list. As discussed in the 6.3 the scheduling algorithm takes ready tasks list, ready time of the resources, the MIPS rating of the grid resources and a list of prediction parameters and the data file movement time of each task in the ready task list on each available resource. The scheduling algorithm then generates a schedule which consists of task to resource mapping list. The schedule is then assigned to dispatcher which dispatches the tasks of a workflow to the grid resources as specified by the scheduling algorithm. When a task completes its execution the relative error rate is checked, the accuracy is measured and the LUs’ load for the next interval is adapted according to the adaptation process discussed in subsection.
7.3.8. This process continues until all the tasks of a workflow complete their execution. When a workflow completes its execution the result of the workflow is then sent to the corresponding grid user. The following steps are followed in each LULP-based scheduling algorithm.

1. Creating a ready task list and for each task in the ready task list the data file movement time of each file on each resource is calculated according to Equation 7-4.

2. Ready time of each task on each resource is calculated according to Equation 7-5.

3. The LULP model is used to calculate the prediction of LUs' load at the grid resources according to Equation 7-8.

4. The MIPS rating of the grid resources along with above three parameters are assigned to a grid workflow LULP-based scheduling algorithm, only those resources information like the ready time of the tasks on the resources is parameterized to LULP-based scheduler which are eligible to execute the tasks in the ready list.

5. The LULP-based scheduling algorithm then finds the expected task completion time of each task in the ready task list on each resource whose information is provided to it according to Equation 7-9.

6. The task to resource mapping is generated by the LULP-based algorithms according to implemented algorithm, e.g., LULP-Sufferage maps a task from ready list with highest value of Sufferage to a grid resource where the task can complete its execution in minimum time than other resources.

7. The task to resource mapping list is sent to the LULP-grid workflow scheduler, where the schedule item list is created and is assigned to Dispatcher component.

8. The dispatcher then dispatches the tasks to the grid resources as specified by the scheduling algorithm and the task manager component marks the task as scheduled.

9. A grid resource after executing the task sends the results of the task back to the corresponding LULP-based Grid Workflow Scheduler, where the data files consumed by the task are deleted and the data files produced by the task are saved according to Algorithm 7-2.
10. The LULP-based grid workflow scheduler sends the results back to the corresponding grid user if all the tasks of a workflow have been completed. The results of a task are also sent to the LULP component to adapt the LUs’ load value for the grid resource for the next time interval as discussed in subsection 7.3.7.

The Strategy Design Patron [Gamma et al., 1995] is followed for development of scheduling algorithms. The start steps like the creation of a ready task list; calculation of ready time of each task in the ready task list, LULP calculation is same are followed before calling the LULP-based scheduling algorithm Therefore these steps are made as startup procedure. Similarly when a task completes its execution the results are sent to the predictor to check the prediction accuracy, the data files consumed by the task are deleted, the data files produced by the task are saved for later use, and the ready tasks list is updated so that a task whose all parents have been completed can be inserted into the ready tasks list. These common end steps are combined in a procedure and are named as end procedure. In the following subsections, start procedure, end procedure and LULP-based algorithms i.e., LULP-Sufferage, LULP-Min-Min and LULP-Max-Min algorithms are presented. LULP-based algorithm takes the LUs’ load at the grid resources as prediction parameter. Prioritize the tasks according to implemented algorithm and schedules the tasks to the grid resources where they can complete in minimum time.

7.5.1. Start Procedure

The common steps required for each LULP scheduling algorithm are shown in the procedure LULP_Start in Algorithm 7-4. The LULP-based scheduling algorithm runs until there exist no unscheduled task of a workflow this is checked by a WHILE loop shown in line 2 of the Algorithm 7-4. The line 3 of the start procedure shows the creation of a readyTasksList. The readyTasksList contains the tasks whose parents have been completed and the data required by them is available. In line number 4 for each task in the readyTaskList the datamovementtime (the data movement time is referred as the ready time of the task on the resource) is calculated and is saved as a map. In this map the resource ID and task number plays the role of key and the data movement time of the task on the resource is value in this map. Line number 5 shows the calculation of predictionParameters, the prediction parameters in this case is the LUs’ load at the grid resources. In line number 6 a LULP-based scheduling algorithm is called.
7.5 LUs’ Load Prediction-based Grid workflow Scheduling Algorithms

Algorithm 7-4: Start procedure LULP scheduling algorithms

1. PROCEDURE LULP_Start{
2. \hspace{1em} WHILE \exists \text{ unscheduled task from set } T \text{ DO } {
3. \hspace{2em} \text{readyTasksList} <\text{- unscheduled ready tasks (ready tasks whose parents had been completed their execution and the data required to start their execution is available)}
4. \hspace{2em} \text{dataMovementTime} <\text{- data movement time of each task in readyTasksList on each resource is calculated according to Equation 7-5}
5. \hspace{2em} \text{predictionParameters} <\text{- LUs’ load at grid resource is using Equation 7-9}
6. \hspace{2em} \text{mappings} <\text{- PROCEDURE LULP_Algorithm(readyTasksList, dataMovementTime, predictionParameters, RTR, MIFratingofresources)}
7. \hspace{1em} } \hspace{1em} \text{// End While}
8. } \hspace{1em} \text{// end PROCEDURE}

The scheduling algorithm returns the tasks to resource mappings of the tasks in the readyTaskList and the resources whose information is provided to the LULP-based scheduling algorithm. The LULP-based grid workflow scheduler then schedules the tasks according to the mappings provided by the scheduling algorithm. Next the LULP-based scheduling algorithms are presented.

7.5.2. LULP-based Sufferage algorithm (LULP-Sufferage)

The algorithms 7-5 present LUs’ Load Prediction-based Sufferage (LULP-Sufferage) scheduling algorithm where, the tasks are prioritized according to their Sufferage values. Sufferage value of a task is the difference between the second minimum expected completion time of the task and the first minimum expected completion time of the task. A task having the maximum value of Sufferage in the ready tasks lists is scheduled to a resource where it can complete its execution earliest in other words the task having the highest value of Sufferage has the highest priority and so on.
Only those resources' information (like MIPS rating, data movement time of the ready tasks on the resources etc.,) is passed to the scheduling algorithm which can fulfill the requirement of a workflow. In the LULP-Sufferage algorithm first of all the expected completion time of each task in the ready task list on each resource is calculated. The first minimum completion time of the task is calculated and the second minimum completion time of the task is calculated as shown in line number 8 and 9 of the algorithm. The Sufferage value of each task is calculated as shown in line number 10. A task with the highest value of Sufferage is selected and it is mapped to a resource where it can complete in minimum time. The task is marked as mapped in the ready tasks list and the procedure continues until all the tasks in the ready tasks list are mapped to the grid resources. The mappings list is returned to the calling procedure which is start procedure in this case. Since Sufferage value shows the difference between the resources performances in a heterogeneous environment, this algorithm can produce good schedules for the heterogeneous environments where the resources have a considerable performance differences.
7.5.3. LULP–based Min-Min algorithm (LULP-Min-Min)

The algorithms 7-6 present a LULP-Min-Min algorithm which uses the LUs’ load predictions and maps the tasks of workflow to the resources where they have minimum completion times. The tasks lengths are used to prioritize the tasks in the ready tasks list. The highest priority is assigned to a task having a minimum task length and so on. In the LULP-Min-Min algorithm the expected completion time of each task in the ready task list on each resource is calculated. The Min-Task is selected as shown in line number 9. The Min-Task is a task having minimum task length in the readyTasksList. The Min-Task is then mapped to a resource where it can complete in minimum time. The task is marked as mapped in the ready tasks list and the procedure continues until all the tasks in the ready tasks list are mapped to the grid resources. The mappings list is returned to the calling procedure.

<table>
<thead>
<tr>
<th>Algorithm 7-6: LULP-Min-Min Scheduling algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. PROCEDURE LULP_Min_Min(readyTasksList, dataMovementTime, predictionParameters, RTR, MIPsratingofresources)</td>
</tr>
</tbody>
</table>
| 2. | \[
| \text{WHILE } \exists \text{ unmapped task in readyTasksList DO} |
| 3. | \[
| \text{INTEGER } i<0 \text{, } j<-1 |
| 4. | \[
| \text{FOR } i=0; i<\text{size of readyTasksList}; i++ \text{ DO} |
| 5. | \[
| \text{FOR } j=1; j<=K; j++ \text{ DO} |
| 6. | \[
| \text{Compute } ECT(t,R) \text{ according to Equation 7-9} |
| 7. | \[
| \text{//end for of the counter } j |
| 8. | \[
| \text{//end for of the counter } i |
| 9. | \[
| \text{INTEGER Min-Task= get the position in the readyTasksList of the } |
| 10. | \[
| \text{task which has the minimum Task length} |
| 11. | \[
| \text{INTEGER Min-resource= get a resource id on which the task of step } |
| 12. | \[
| 9 \text{ have minimum completion time} |
| 13. | \[
| \text{mappings<- Map Min-Task on Min-resource} |
| 14. | \[
| \text{DELETE Min-Task from readyTasksList} |
| 15. | \[
| \text{UPDATE the RTR of Min-resource} |
| 16. | \[
| \text{//end while} |
| 17. | \[
| \text{RETURN mappings} |
| 18. | \[
| \text{//end PROCEDURE} |

7.5.4. LULP-based Max-Min algorithm (LULP-Max-Min)

Algorithm 7-7 works similar to the Min-Min algorithm. The only difference is the method of prioritizing the tasks of a workflow in the ready list. The task having biggest value of task’s length, gives it the highest chances to be scheduled on the best resource. The Max-Min algorithm may outperform the Min-Min algorithm where there are many short length tasks
and few big lengths tasks. In this way, the tasks with the biggest length may execute parallel with the smaller lengths’ tasks.

In the LULP-Max-Min algorithm first of all the expected completion time of each task in the ready task list on each resource is calculated. The Max-Task is selected as shown in line number 9. The Max-Task is a task having maximum task length in the readyTasksList. The Max-Task is then mapped to a resource where it can complete in minimum time. The task is marked as mapped in the ready tasks list and the procedure continues until all the tasks in the ready tasks list are mapped to the grid resources. The mappings list is returned to the calling procedure.

**Algorithm 7-7: LULP-Max-Min scheduling algorithm**

```plaintext
PROCEDURE LULP_Max_Min(readyTasksList, dataMovementTime, predictionParameters, RTR, MIPsratingofresources)

1. WHILE ∃ unmapped task in readyTasksList DO
2.   INTEGER i<0 , j<1
3.   FOR i =0; i<size of readyTasksList; i++ DO
4.     FOR j =1; j<=K; j++ DO
5.       Compute \( E(t_i, R_j) \) according to Equation 7-9
6.     } //end for of the counter j
7.   } //end for of the counter i
8.   INTEGER Max-Task=get the position in the readyTasksList of the task which has the maximum Task length
9.   INTEGER Min-resource=get a resource id on which the task of step 9 have minimum completion time
10.  SCHEDULE Max-Task on Min-resource
11.  DELETE Max-Task from readyTasksList
12.  UPDATE the RTR of Min-resource
13.  } //end while
14.  RETURN mappings
15. } //end PROCEDURE
```

7.5.5. Receiving a completed task

When a task completes its execution the grid resources returns and its results are returned to the workflow scheduler. Algorithm 7-8 shows the sequence of steps followed when a task is received from a grid resource. The results of the completed task are saved as shown in line number 1. The data file consumed by the task are deleted and the data files produced by the task are saved on the grid resources as shown in line number 3 and 4 of the receive procedure. The LULP is updated to check the accuracy, so that the consequence prediction of LUs’ load can be adapted accordingly. The dependency constraints among the child tasks
of the completed task are removed and a new readyTaskList is created which contains the tasks whose all parents have been completed and the data files required by them are available.

Algorithm 7-8: Receiving a completed task of a workflow

```
1. receive(tᵢ) {
2.     SAVE the results of tᵢ
3.     DELETE the data files consumed by tᵢ
4.     SAVE the data files produced by tᵢ
5.     UPDATE LULP to check the prediction accuracy
6.     UPDATE the dependency constraints of the child tasks of tᵢ
7.     readyTasksList ← get the new readyTasksList
8.     Send the results of a workflow to the corresponding user if all the tasks of a workflow have been completed their execution
9. }
```

7.5.6. Complexity and overhead analysis

The algorithms LULP-Max-Min, LULP-Min-Min and LULP-Sufferage are list-based scheduling algorithms. The tasks of a workflow ready for execution are placed in a list which is named as readyTasksList. The scheduling decision on this way is dependent on many parallel independent tasks of a workflow in the ready tasks list. The tasks in the ready tasks list are prioritized and the highest priority task is assigned to a resource where it can complete in minimum time. The complexity of such algorithms depends on the number of tasks in a workflow, the number of available resources and the size of the ready tasks list. Each list-based scheduling algorithm executes L times, where L is the size of the List. The completion time of each task on the available resource is computed. Only those resources which can fulfill the requirements of the tasks of a workflow are considered. However, if all the resources in a grid can execute the tasks of a workflow then in this case the list-based scheduling algorithm will execute M times where M is the total number of resources in the grid. Since for each task the completion time is calculated and if the number of tasks in the workflow is equal to N, then the list-based algorithm will execute N times. In big O notation the complexity of list-based scheduling algorithms is \( O(LMN) \). The LULP-based algorithms are compared with HEFT algorithm. The complexity of HEFT algorithm is given as \( O(N^2M) \). The decision of scheduling a task is dependent upon the rank of the task in the workflow, i.e., the highest rank of a task is assigned the highest priority and so on.
Figure 7-2 shows the execution process of a workflow. The grid user when submits a workflow to a scheduler this workflow processing request is received by scheduler through internet. So in the first step the workflow suffers from the internet delay. The received workflow is placed in the queue where it waits for its turn. In all the implemented algorithms, the queue uses the First Come First Serve (FCFS) policy. So the receiving workflows are placed in the same order as they were received. When a workflow reaches at the front of the queue, the ready list of the tasks of the workflow is made and it starts its execution. Whereas, in case of HEFT algorithm the tasks ranks of a workflow are calculated and then the critical path is determined. After the construction of ready tasks list it is rearranged on descending order of tasks’ ranks. This time is referred as schedule generation time.

![Workflow execution process](image)

Figure 7-2: Workflow execution process

The schedule generation time of LULP-based algorithms also includes the prediction time. The LULP uses the MA-based prediction model to predict the LUs’ load, which is computationally efficient and is placed in the category of negligible complexity in the prediction technique as compared to other prediction method of time series [Box and Jenkins, 1990; Mitsa, 2010; Herbst et al., 2013]. The makespan of a workflow is referred as the time elapsed between the execution’s start time of the first task of a workflow and the execution’s end time of the last task of a workflow. When all the tasks of a workflow
complete their execution the results are assembled and the results of a workflow are sent back to the corresponding grid user. The result assembling time of the LULP-based algorithms and the HEFT is same.

### 7.5.7. An example scenario

In order to show working of the LULP-based algorithms a grid environment with three resources is considered. The specification of the resources is shown in Table 7-2.

#### Table 7-2: Example resource configurations

<table>
<thead>
<tr>
<th>Resources</th>
<th>Number of PEs</th>
<th>Processing speed of one PE</th>
<th>Storage Units</th>
<th>Storage capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$R_2$</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$R_3$</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

The unit data transfer time and latencies of the links between the grid resources are given in Table 7-3 and Table 7-4 respectively.

#### Table 7-3: Bandwidth of the links among grid resources

<table>
<thead>
<tr>
<th>Transfer Time</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$R_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$R_3$</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Table 7-4: Latencies of the links among grid resources

<table>
<thead>
<tr>
<th>Latency</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$R_2$</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$R_3$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The LUs’ load for the grid resources is shown in Table 7-5. The workflow shown in Figure 7-3 is scheduled on the grid resources described above. The ready tasks list of the workflow is constructed. The ready tasks list in first step has only one task $t_1$. The expected completion time of $t_1$ is calculated on all the three resources assuming that all three resources are eligible to execute this workflow and it given in Table 7-6.
Since the task list contains only one task, it is scheduled on the resource where it will complete in minimum time i.e., on resource \( R_1 \). In the next step when the task \( t_1 \) finishes its execution the data generated by it is saved on the resource where it has completed its execution. The following sequences of steps are followed to save the data files.

1. The data files are rearranged in descending order of their sizes.

2. The biggest data file and the resource capacity where the producing task has completed its execution are checked if possible to save the biggest data file then it is saved on the resources.

3. Otherwise the data file is saved to another resource such that the data movement time is low.
4. This process is continued until all the data files are saved.

When \( t_1 \) completes and the result is received by the scheduler then the prediction accuracy is performed. Now in case of \( t_1 \) the two files are generated having weights \( c_{1,2} \) and \( c_{1,3} \). The data files having weight \( c_{1,3} \) is saved on \( R_1 \). After that, the remaining capacity is checked by the data locator and if possible then the remaining files are saved on resource \( R_1 \). Let it is possible to save \( c_{1,2} \) on resource \( R_1 \). After saving the data files the data files locations are saved by the LULP-grid workflow scheduler as shown in Table 7-7. The ready tasks list will have two tasks \( t_2 \) and \( t_3 \). The data file movement time and data locations are shown in Table 7-7 and Table 7-8 respectively.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_{1,2} )</td>
<td>( c_{1,2} )</td>
<td>( R_1 )</td>
</tr>
<tr>
<td>( e_{1,3} )</td>
<td>( c_{1,3} )</td>
<td>( R_1 )</td>
</tr>
</tbody>
</table>

Table 7-8: Data Transfer Time of \( t_2 \) and \( t_3 \)

<table>
<thead>
<tr>
<th>Task</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1.5</td>
<td>2</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>0</td>
<td>1.5</td>
<td>2</td>
</tr>
</tbody>
</table>

The task completion times of tasks \( t_2 \) and \( t_3 \) are given in Table 7-9. In this example the working of LULP-Max-Min is explained as the task \( t_2 \) has a priority over task \( t_3 \), due to its longer task length therefore \( t_2 \) is scheduled first and it is assigned to the grid resource \( R_1 \). Then ready time of grid resource is updated and task \( t_3 \) is assigned to the grid resource \( R_3 \).

<table>
<thead>
<tr>
<th>Task</th>
<th>( R_1 )</th>
<th>( R_2 )</th>
<th>( R_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_2 )</td>
<td>( 6 + \text{max}(0,0) + \frac{10}{4} + \frac{6}{2} = 11.5 )</td>
<td>( 6 + \text{max}(0,1.5) + \frac{7.0}{4} + \frac{6}{2} = 15.5 )</td>
<td>( 6 + \text{max}(0,1.5) + \frac{5}{3} + \frac{6}{1} = 15.1 )</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>( 11.8 + \text{max}(0,0) + \frac{10}{4} + \frac{3}{2} = 15.8 )</td>
<td>( 6 + \text{max}(0.2) + \frac{20}{4} + \frac{3}{2} = 14.5 )</td>
<td>( 6 + \text{max}(0.3) + \frac{5}{3} + \frac{3}{1} = 13.6 )</td>
</tr>
</tbody>
</table>

The scheduler receives the result of task \( t_2 \) and fetches the task \( t_4 \) into the ready list. The task \( t_2 \) produced two data files these are \( c_{2,4} \) and \( c_{2,5} \) as shown in Table 7-10. The data file saving algorithm first try to save the biggest file \( c_{2,4} \) on the resource \( R_1 \). Let the attempt of
saving the data file $c_{2,4}$ on $R_1$ fails. Then, it is saved on resource $R_2$. However, the data file $e_{2,5}$ finds a place at resource $R_1$.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Weight</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{2,4}$</td>
<td>$c_{2,4}$</td>
<td>$R_2$</td>
</tr>
<tr>
<td>$e_{2,5}$</td>
<td>$c_{2,5}$</td>
<td>$R_1$</td>
</tr>
</tbody>
</table>

The task $t_4$ is the only ready task and its completion time on the three resources is given in Table 7-11.

<table>
<thead>
<tr>
<th>Task</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_4$</td>
<td>$11.5 + \max(0.2,5) + \frac{10}{4} + \frac{4}{2} = 18.5$</td>
<td>$11.5 + \max(0,0) + \frac{20}{4} + \frac{4}{2} = 19.5$</td>
<td>$11.5 + \max(13.6,2.5) + \frac{5}{3} + \frac{4}{1} = 30.76$</td>
</tr>
</tbody>
</table>

After the completion of task $t_3$ the data file of dependency $e_{3,5}$ is saved on resource $R_1$ and the task completion time is calculated for task $t_5$, it is shown in Table 7-12.

<table>
<thead>
<tr>
<th>Task</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_5$</td>
<td>$13.6 + \max(18.5,0) + \frac{10}{4} + \frac{5}{2} = 37.1$</td>
<td>$13.6 + \max(0,2.0) + \frac{20}{4} + \frac{5}{2} = 23.1$</td>
<td>$13.6 + \max(0,2.0) + \frac{5}{3} + \frac{5}{1} = 22.2$</td>
</tr>
</tbody>
</table>

The $\checkmark$ symbol shows that the task is assigned to the resource in the header of the column. In this case the makespan is 22.2 time unit. In comparison with the HEFT algorithm the data file generated by the parent task does apply any algorithm to save the data files generated during the execution of a workflow therefore, in HEFT the data files are saved on randomly selected grid resources. HEFT algorithm is unaware of the LUs’ load present on the grid resources. Therefore, HEFT may generate a worse makespan due to ignorance of LUs’ load at the grid resource.
7.6. Experiment Design and Result Evaluation

7.6.1. Algorithms to evaluate

In order to evaluate the effectiveness of \texttt{LULP-Sufferage}, \texttt{LULP-Min-Min} and \texttt{LULP-Max-Min} scheduling algorithms these algorithms are compared with \texttt{HEFT} which is the most popular workflow scheduling algorithm. The details of \texttt{LULP-based} scheduling algorithms are given in section 7.5. The details of \texttt{HEFT} can be found in [Topcuouglu et al., 2002]. As the name suggests the \texttt{LULP-Sufferage} scheduling algorithms gets the predicted LUs’ load at each available resource in the grid and then picks a task from the ready tasks list with the highest value of Sufferage and maps it to the resource where it can complete in minimum time. In case of \texttt{LULP-Min-Min} the load prediction process is same but the priority is assigned to a task having smallest task length. In case of \texttt{LULP-Max-Min} the longest task length task has the highest priority. Once a task is selected from the ready list, \texttt{LULP-based} algorithms assign the task to a resource where it can complete in minimum time. In case of \texttt{HEFT} algorithm, the scheduler is unaware of the LUs’ load at the grid resources and there is no mechanism to save the data files generated by the parent tasks for their child tasks. So, in case of \texttt{HEFT} the data files are saved on a randomly selected resource.

7.6.2. Workload generation

A variant of HC-test bench [Hönig and Schiffmann, 2004; Hönig, 2007] is used to generate the workload for this simulation experiments. The workflows in the original test bench contain maximum 24 tasks whereas the test bench used in this study has up to 250 tasks. Also, the HC-test bench consists of 7200 workflows and the variant of HC-test bench contains 9600 workflows. Detail about HC-test bench structure can be found in section 5.1. The goal of this study is to find the behavior of proposed LULP-based workflow scheduling algorithms under different sizes of workflows. So, all the workflows in the test bench have an equal probability of being selected for this experiment. Therefore, the workflow size in terms of number of tasks in this experiment varies from 6 to 250. Along with the workflow properties described in [Hönig and Schiffmann, 2004; Hönig, 2007] like number of edges, edge and task weight, edge length etc., to model a dynamic workload following properties are modeled.
Number of competing workflows: The decentralized architecture of the scheduler allows more than one workflow execution concurrently. In these experiments, the concurrent execution of 5, 10, 15, 20 and 25 workflows is simulated.

Arrival rate: The arrival of workflow is simulated using the Poisson distribution, with a mean arrival rate of 10, 15, 20, and 25 per interval.

LUs’ workload: Beside the grid workload (workflows), the grid resources may have LUs present on them. These LUs submit the tasks to the grid resources according to DAS2 (fs0-fs4) workload archive [DAS-2, 2012]. DAS2 (fs0-fs4) is grid workload log recoded from 5 clusters located in the five different universities in Netherlands. DAS2 stands for “Distributed ASCI Supercomputer-2”, where ASCI is the abbreviation of the “Advance School for Computing and Imaging” in Netherlands.

7.6.3. Resource model

The resources are simulated according to a real grid resources’ configurations i.e., EU data grid. The specifications of resources are given in Table 7-13. The resources’ processing capacity is modeled in Millions of Instructions per Second (MIPS) as per standard performance evaluation corporation (SPEC) CPU (INT) 2000 [SPEC, 2012] benchmarks. The network link capacity of resources varies between 45 Mb/S to 10GB/S. The detail about the network topology of the EU data grid can be found in [Sulistio et al., 2008].

<table>
<thead>
<tr>
<th>Resources</th>
<th>Number of PES</th>
<th>MIPS of one PE</th>
<th>Storage (TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>41</td>
<td>49000</td>
<td>2.75</td>
</tr>
<tr>
<td>$R_2$</td>
<td>52</td>
<td>62000</td>
<td>1.80</td>
</tr>
<tr>
<td>$R_3$</td>
<td>17</td>
<td>20000</td>
<td>1.0</td>
</tr>
<tr>
<td>$R_4$</td>
<td>18</td>
<td>21000</td>
<td>0.50</td>
</tr>
<tr>
<td>$R_5$</td>
<td>12</td>
<td>14000</td>
<td>1.35</td>
</tr>
<tr>
<td>$R_6$</td>
<td>59</td>
<td>70000</td>
<td>2.50</td>
</tr>
<tr>
<td>$R_7$</td>
<td>5</td>
<td>7000</td>
<td>0.35</td>
</tr>
<tr>
<td>$R_8$</td>
<td>2</td>
<td>3000</td>
<td>0.10</td>
</tr>
<tr>
<td>$R_9$</td>
<td>5</td>
<td>6000</td>
<td>0.25</td>
</tr>
<tr>
<td>$R_{10}$</td>
<td>1</td>
<td>1000</td>
<td>0.05</td>
</tr>
<tr>
<td>$R_{11}$</td>
<td>67</td>
<td>80000</td>
<td>5.0</td>
</tr>
</tbody>
</table>
7.6.4. Performance metrics

Since the objective of proposed algorithms was to achieve low makespan of workflows in the presence of the LUs of the grid resources and effectively handling the data movement time between the tasks of a workflow. For comparing the LULP-based algorithms and HEFT scheduling algorithm the following metrics are considered.

1. **Average makespan**: Makespan is defined as the time difference between the minimum submission time of the task of a workflow and the maximum end time of the task of a workflow. This time measures the performance of the scheduling algorithm.

2. **Average makespan difference ratio**: The makespan difference ratio is calculated by using the HEFT algorithm as base algorithm and the LULP-based algorithms are compared with it. Thus the average makespan difference ratio of HEFT scheduling algorithm is always 0. The formula for the makespan difference ratio for LULP-Max-Min is given in Equation 7-33.

   \[
   \frac{\text{Makespan Difference Ratio}(LULP - \text{Max} - \text{Min})}{\text{Makespan of HEFT} - \text{Makespan of LULP} - \text{Max} \text{ Min}} = \frac{\text{Makespan of HEFT}}{\text{Makespan of HEFT}} \quad \text{Equation 7-33}
   \]

3. **Data movement time**: The average data movement time a workflow is calculated by dividing the sum of data movement times of all the tasks of a workflow on the number of tasks in the workflow. The average data movement time in an interval is calculated by dividing the average data movement time of all workflows executed in the interval by total number of workflows executed in the interval. An average data movement time of an experiment is calculated by dividing the average data movement time of all intervals by the total number of intervals in the experiment. At last, the mean data movement time is calculated which is referred simply as data movement time here by dividing the average data movement time of the all the experiments of a scenario by the 5. As every experiment is repeated 5 times.

4. **Data movement time difference ratio**: The data movement time difference ratio is calculated by using the HEFT algorithm as base algorithm and the LULP-algorithms
are compared with it. Thus the average data movement time ratio of HEFT is always 0. The formula same as for the makespan difference ratio, however here the data movement time is used instead of makespan.

5. **Resource effective utilization**: The resource effective utilization of a workflow (having \( N \) tasks) \( REU(W) \) is given in Equation 7-34. \( ACT(t_i) \) is the actual completion time of the task \( t_i \).

\[
REU(W) = \frac{\sum_{i=1}^{N} ACT(t_i) \times \text{Processors used}(t_i)}{\text{available processors} \times \text{makespan}(W)}
\]  

Equation 7-34

6. **Prediction Accuracy**: The prediction accuracy (%) of the LULP-based algorithms are determined for all simulated experiments, and it shows the % of correct decisions about the workflows’ tasks placement by the LULP-based algorithms in an experiment as discussed in 7.3.7.

7.6.5. Results

All the experiments were carried on a computer having Intel® dual-core processor having 2.6 Gigahertz (GHz). The GridSim simulator was used for simulation. Every experiment was executed five times to get the mean value. The mean arrival rate of the workflows in an interval and the number of concurrently executing workflows were varied in order to study their impact on the parameters introduced above.

Figure 7-4 shows the algorithms’ performance with different number of concurrent workflows. The results show that the **LULP-Min-Min** and the **LULP-Sufferage** scheduling algorithms have almost identical performance in terms of the makespan. The **LULP-based** scheduling algorithms, i.e., the **LULP-Max-Min**, **LULP-Min-Min** and **LULP-Sufferage** have always outperformed the **HEFT** algorithm in terms of the makespan. The **LULP-based** scheduling algorithms have outperformed the **HEFT** algorithm in terms of the makespan even in the case where, the arrival rate of the workflows is high and more concurrent workflows were in execution.

Figure 7-5 shows the makespan difference ratio (%) in various experiments. As the number of competing workflows and mean arrival rate increases, the makespan of all algorithms increases. The **LULP-Max-Min** scheduling algorithm produced the lowest makespan. The
comparison of HEFT and LULP-Max-Min algorithms shows the improvement in the makespan by 43% and 38% when concurrent executing workflows were 5 and the mean arrival rate per interval was 10 and 25 respectively.

Figure 7-6 shows the algorithms’ performance in terms of the data movement time when different numbers of concurrent workflows were in execution with a varying mean arrival rate of workflows per interval. The results show that the LULP-based scheduling algorithms, i.e., the LULP-Max-Min, LULP-Min-Min and LULP-Sufferage always spent less time on data movement as compared to the HEFT algorithm because of the integrated data saving algorithm.

Figure 7-7 shows the data movement time difference ratio (%) in various simulated scenarios. The data movement time of the LULP-Max-Min scheduling algorithm is very less as compared to HEFT and other LULP-based scheduling algorithms, therefore the data movement time difference ratio of LULP-Max-Min shows a significant improvement. As the mean arrival rate and the number of the concurrently executing workflows increases, the data movement time difference ratio also increases in the LULP-based scheduling algorithms. In case when 5 concurrent workflows were in execution and the mean arrival rate was 10 and 25, the data movement time difference ratio (%) of LULP-Max-Min remained 85% and 89% respectively. In case when 25 concurrent workflows were in execution and the mean arrival rate was 10 and 25, the data movement time difference ratio (%) of LULP-Max-Min remained 89.3% and 89.6%. Thus, a significant improvement was achieved in terms of the makespan and the data movement time by the LULP-based scheduling algorithms.

The grid workflows scheduling algorithms always utilize the available grid resources very poorly due to various reasons, e.g., the workflow’s tasks need to wait for the completion of their parents and the arrival of data from their parents. Therefore, despite the availability the grid resources they cannot be utilized. Figure 7-8 shows the resource effective utilization of the conducted experiments. It is clear that the resource effective utilization of the HEFT algorithm is better than the LULP-based scheduling algorithms. Figure 7-8 only shows the resource effective utilization by the workflows. However, the grid resources were also utilized by their LUs, which was same for all the studied scheduling algorithms. So, it must be admitted that the makespan and the data movement time gain makes the resource
utilization loss less effective. The resource effective utilization of the **HEFT** scheduling algorithm was better than **LULP-based** scheduling algorithms, however the **HEFT** scheduling algorithm spent a huge amount of time on the data movement and the makespan of the workflows produced by the **HEFT** scheduling algorithm was high as compared to the **LULP-based** scheduling algorithms. So it can be argued that the **HEFT** scheduling algorithm achieved the resource effective utilization on the cost of the makespan and additional data movement time. The prediction accuracy (%) of the task placement decisions of the **LULP-Min-Min**, **LULP-Sufferage** and **LULP-Max-Min** algorithms were calculated, which are given in Table 7-14, Table 7-15 and Table 7-16 respectively.
Figure 7.4: Makespan vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 7-5: Makespan difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per Interval
7.6 Experiment Design and Result Evaluation

Figure 7.6: Data movement time vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 7.7: Data movement time difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 7.8: Resource effective Utilization (%) vs. total number of concurrent workflows under different mean arrival rate per interval.
### Table 7-14: Prediction accuracy (%) of LULP-Min-Min

<table>
<thead>
<tr>
<th>Concurrent</th>
<th>Mean 10</th>
<th>Mean 15</th>
<th>Mean 20</th>
<th>Mean 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>63.5</td>
<td>67.2</td>
<td>65.3</td>
<td>67.8</td>
</tr>
<tr>
<td>10</td>
<td>69.3</td>
<td>72.3</td>
<td>73.1</td>
<td>74.2</td>
</tr>
<tr>
<td>15</td>
<td>70.1</td>
<td>70.3</td>
<td>74.0</td>
<td>75.3</td>
</tr>
<tr>
<td>20</td>
<td>72.0</td>
<td>72.6</td>
<td>76.2</td>
<td>76.0</td>
</tr>
<tr>
<td>25</td>
<td>78.3</td>
<td>76.4</td>
<td>73.1</td>
<td>72.2</td>
</tr>
</tbody>
</table>

### Table 7-15: Prediction accuracy (%) of LULP-Sufferage

<table>
<thead>
<tr>
<th>Concurrent</th>
<th>Mean 10</th>
<th>Mean 15</th>
<th>Mean 20</th>
<th>Mean 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>63.3</td>
<td>65.2</td>
<td>63.0</td>
<td>69.0</td>
</tr>
<tr>
<td>10</td>
<td>70.2</td>
<td>71.3</td>
<td>72.5</td>
<td>75.3</td>
</tr>
<tr>
<td>15</td>
<td>73.3</td>
<td>72.1</td>
<td>71.4</td>
<td>76.4</td>
</tr>
<tr>
<td>20</td>
<td>79.0</td>
<td>78.2</td>
<td>76.4</td>
<td>76.2</td>
</tr>
<tr>
<td>25</td>
<td>79.2</td>
<td>78.1</td>
<td>73.5</td>
<td>74.0</td>
</tr>
</tbody>
</table>

### Table 7-16: Prediction accuracy (%) of LULP-Max-Min

<table>
<thead>
<tr>
<th>Concurrent</th>
<th>Mean 10</th>
<th>Mean 15</th>
<th>Mean 20</th>
<th>Mean 25</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>65.4</td>
<td>68.2</td>
<td>65.0</td>
<td>68.1</td>
</tr>
<tr>
<td>10</td>
<td>69.1</td>
<td>73.0</td>
<td>76.5</td>
<td>76.0</td>
</tr>
<tr>
<td>15</td>
<td>73.2</td>
<td>72.6</td>
<td>76.0</td>
<td>75.3</td>
</tr>
<tr>
<td>20</td>
<td>79.8</td>
<td>78.1</td>
<td>79.2</td>
<td>77.1</td>
</tr>
<tr>
<td>25</td>
<td>80.0</td>
<td>79.2</td>
<td>76.1</td>
<td>75.0</td>
</tr>
</tbody>
</table>

### 7.7. Summary

This chapter presents the LULP-based scheduling algorithms addressing the challenges of the dynamic grid workflow scheduling i.e., the dynamically arriving workflows and the execution of workflows’ tasks in the presence of the LUs on the grid resources. The LUs’ load is predicted (Requirements R7 and R8) and utilized by the LULP-based scheduling algorithms (Requirement R10) to achieve a lower makespan of the workflows. Moreover, the data saving algorithm (Requirement R12) presented in this chapter shows a significant improvement in terms of the data movement time. However, it is observed that the grid resources are not utilized effectively. This is because the workflow’s tasks are scheduled to the lightly loaded resource and some resources in the grid remain in idle state.
8. Grid Workflow Scheduling Based on Queuing Theory

This chapter presents the solutions for the requirements (Requirements R9, R11, and R12). Requirement R9 suggests the prediction of queue wait time at the grid resources. Requirement R11 suggests the design and development of the grid workflow scheduling algorithm utilizing the predicted queue wait time and requirement R12 suggests that the design of a data saving algorithm which can be used to save the data files generated during execution of the workflows’ tasks.

This chapter is organized as follows. Section 8.1 discusses the conventional grid workflow scheduling workflow scheduling algorithms and their oversimplified assumption that the tasks of workflow immediately after they arrive at the grid resources.

Section 8.2 presents the related work of grid workflow scheduling in the context of the queue wait time prediction-based grid workflow scheduling. Section 8.3 presents the basic concepts used in queuing theory. Section 8.4 presents a grid resource queue model conceptually and section 8.5 presents a mathematical model of grid resource queue. The section 8.6 provides the prediction of the parameters required for the realization of queue wait time of at a grid resource. Section 8.8 presents the QWP-GWS algorithm and a data saving algorithm. Both Algorithms works together to achieve a relatively small makespan and overall less data movement time throughout the workflow execution. Section 8.9 consists of the experimental design of the experiments conducted to evaluate the QWP-GWS algorithm and the data movement algorithm. Finally the section 8.10 summarizes the chapter.

8.1. Conventional Scheduling Algorithms

As discussed in section 7.1, the conventional grid workflow scheduling algorithms consider that the grid resources are dedicated and are always available for the execution workflows’ tasks. The tasks of a grid workflow when sent to a grid resource immediately starts their execution is another unrealistic assumption taken in conventional grid workflow scheduling algorithms. However the site-specific policies implemented at the
grid resources may be just the opposite of these assumptions because of the following reasons.

1. In a real grid scenario the grid users’ applications (workflows) have to compete with each other and with the grid resources’ LUs’ tasks to get the access to a grid resource.

2. A workflow’s task submitted to a grid resource serving many other users must have to wait in the grid resource queue to access a free processor or a number of free processors for its execution.

3. The data require to start the execution of a workflow task is being transferred to the resource.

It is a fact that a task of a workflow and a workflow may not start their execution just after their arrival motivated the grid researchers to introduce the queuing system at the grid resource level and grid workflow scheduler level. Nowadays all the grid resources and grid workflow scheduler implement the queuing systems. The grid workflow scheduling algorithm which is aware of the queue wait time suffering of a workflow’s tasks due to the LUs of grid resources and other competing workflows’ tasks is still missing and motivated us to present such an algorithm in this chapter.

8.2. Related work

In this chapter a grid workflow scheduling algorithm is presented which utilizes a data saving algorithm and predicted queue wait time at the grid resources and maps the tasks of the workflow to the grid resources such that overall makespan is low. The proposed algorithm finds the different paths in a workflow and schedules each path of the workflow to a suitable grid resource.

Many heuristic and meta-heuristic algorithms have been presented to solve the workflow scheduling problems. Some of them are Min-Min, Myopic, Max-Min, Heterogeneous Earliest Finish Time First (HEFT), greedy randomization, adaptive search, simulated annealing, genetic algorithm, A* algorithm and many more. The related work to the proposed Queue Wait Time Prediction-based Grid Workflow Scheduling (QWP-GWS)
algorithm is presented in following subsections. The related work compares the characteristics of the QWP-GWS algorithm in the following two contexts.

1. Critical path-based grid workflow scheduling algorithms

2. Queue wait time prediction-based grid workflow scheduling algorithms

8.2.1. Critical path-based scheduling algorithms

The Critical Path (CP)-based scheduling algorithms belong to the priority-based scheduling algorithms. The first CP-based scheduling algorithm was presented by khwok and ahmad in [Kwok and Ahmad, 1996]. The authors of [Kwok and Ahmad, 1996] presented a workflow scheduling algorithm namely Dynamic Critical Path (DCP) scheduling algorithm. The DCP algorithm dynamically finds the critical path in a workflow and assigns the highest priorities to the tasks on the critical path. The DCP algorithm was designed to schedule the tasks of a workflow on the homogeneous processor system.

[Topcuouglu et al., 2002] presented a Critical Path on a Processor (CPOP) scheduling algorithm and Heterogeneous Earliest Time First (HEFT) scheduling algorithm. The CPOP algorithm assigns all the tasks of a critical path on a single processor to minimize the computational time of the tasks on the critical path. In HEFT the tasks are prioritized according to rank value and a task having highest value of rank is assigned to a grid resource where it can complete in minimum time. The CPOP algorithm uses a different method of ranking the tasks of a workflow. CPOP adds the upward and downward ranks of a task of a workflow and assigns the sum value as a rank.

Ma and Buyya [Tianchi and Rajkumar, 2005] presented an xDCP scheme for workflow scheduling. The research presented in [Tianchi and Rajkumar, 2005] is valid only for the workflows’ tasks, which can be further divisible i.e., for the tasks of workflows, which belongs to parametric sweep application type.

Rahman et al. presented a Dynamic Critical Path for Grid (DCP-G) scheduling algorithm in [Rahman et al., 2007]. DCP-G was inherited from DCP [Kwok and Ahmad, 1996] and the new features in the DCP-G are the dynamically changing values of earliest/latest start time (AEST, ALST) of the tasks of a workflow. For further detail of DCP and DCP-G one may consult the [Kwok and Ahmad, 1996; Rahman et al., 2007].
8.2.2. Queue wait time prediction-based grid workflow scheduling algorithms

Currently in action all scheduling systems are queuing systems, examples are PBS [Hanhua et al., 2007], LSF [Xu, 2001], Condor [Douglas et al., 2002], gLite WMS [Burke et al., 2009], Maui and Moab [Moab, 2012], GridWay [Eduardo et al., 2005], Unicore [Romberg, 2000], GRMS [Jiandong et al., 1997], Grid Service Broker [Venugopal et al., 2004] etc. These systems use one or more queues where incoming jobs are stored until they are scheduled for execution. Some of the above systems support the workflow scheduling by the workflow management tools like DAGMan [Thain et al., 2005], Kepler [Ludäscher et al., 2006], Taverna [Hull et al., 2006], Triana [Taylor et al., 2007], Genius [TheGeniusPortal, 2012] etc. The tools are specifically for modeling the workflows and the queue wait time prediction-based workflow scheduling algorithm is still missing in them. Scheduling the workflows with an assumption that the tasks will start immediately after their arrival to a grid resource is not a feasible solution. Realizing this problem some researchers tried to solve the scheduling with the queuing concepts.

Wu et al in [Wu et al., 2007] presented an M/G/n queue model. In [Wu et al., 2007], the authors assumed that the grid resources are in operation with different service rate. Each resource service time follows an exponential distribution. There is only one queue, which receives the jobs from the grid users, and all the resources can access this queue. The authors provided a simulation result of the system by simulating only three grid resources and the characteristics of jobs are not defined at all.

Afzal et al in [Afzal et al., 2006] solve the grid workflow scheduling problem by treating the grid as an open queuing network. Each service of grid in [Afzal et al., 2006] is treated as a M/G/1 queue model. The authors of [Afzal et al., 2006] used the P-K formula (details of P-K formula can be found in [Afzal et al., 2006]) to find mean response time and variance of response time at each service queue. In [Afzal et al., 2006] it is considered that the communication time of workflows is negligible. Also, [Afzal et al., 2006] did not discuss the workflows used in experiments in detail. [Afzal et al., 2006] did not discuss the realization of the basic parameters of a queue like the arrival rate and the response time.
Yu et al in [Yu and Shi, 2010] solved the workflow scheduling problem in multi-cluster environments. They used the upward ranking mechanism to prioritize the tasks of a workflow. In [Yu and Shi, 2010] the authors assumed the dynamic grid environment but at the same time the workflow scheduling scheme transfer the data to a grid resource prior the task assigned to a grid resource. In other words, the task placement decision is static. The results presented in [Yu and Shi, 2010] shows that a lot of time is spent in data communication when the proposed queue wait time predictions are in action. The [Yu and Shi, 2010] gives probability measurements that a task will execute on a grid resource by its dead line and how long the task will wait in the queue is not determined. Moreover, in the said algorithm there is only one workflow in execution, whereas in a real grid environment many workflows’ tasks may compete with each other for grid resources. All the said problems are tried to solve in the proposed workflow scheduling algorithm.

8.3. Introduction to Queuing Concepts

An abstract data structure, which is used as waiting line (for the tasks of workflows and for workflows) is referred as queue [Queue, 2012]. The queuing theory is the mathematical study of queues to analyze the arrival of jobs at the rare of a queue, waiting time in the queue and getting a service (when a job reaches at the front of a queue) [Robertazzi, 2007]. The queuing theory permits the calculation of performance measures like average queue length, wait time to get a free grid resource, the expected service time at a grid resource etc. The role of queues is considered important while building a model to measure or predict the performance of a parallel and distributed system like grids. The queue of various types and sizes are present in a grid system e.g., at grid scheduler, at grid resources etc. In this research the queue are considered for keeping the waiting workflows and tasks to get the required resources.

8.3.1. Kendall notations A/B/m

Kendall introduced a three part code A/B/m for the explaining queues [Gunter et al., 1998]. The first, second and third letters specify the inter-arrival time distribution, service time distribution and number of resources respectively as shown in Figure 8-1. These three letter notation can be extended to A/B/m/n/s as described in [Gunter et al., 1998; Robertazzi, 2007] here n stands for the queue capacity and s for service discipline (FIFO,
LIFO etc.). When the letter n and s are omitted, than it is explicitly understood that the queue has infinite capacity and service discipline is FIFO (First in First Out) [Ivo and Jacques, 2001]. Usually the arrival process is modeled as Markov Modulated Poisson, which is also referred as memory less distribution or simply Poisson in this case letter M is used at first place. The letters are revised according to the distribution used e.g., for a general distribution letter G and for deterministic D is used [Gunter et al., 1998; Ivo and Jacques, 2001; Robertazzi, 2007].

![Queue model](image)

**Figure 8-1: Queue model**

### 8.3.2. Important definitions

The task of a workflow when scheduled to a grid resource and all the processors of the resource are busy then it has to wait in the queue of the resource. The queuing theory facilitated to study such system. In order to analyze a system by the queuing theory the following characteristics of the system are needed to be specified.

**Arrival process**: The workflows arrival times at a grid scheduler are considered as random variables, and the difference of arrival times between two successive workflows is called the inter arrival time. It is generally assumed that inter arrival times are calculated from a sequence of independent and identically distributed (IID) random variables. Commonly used arrival process is Poisson, where inter arrival times are IID from an exponential distribution.
**Service time distribution:** A grid scheduler can schedule the tasks of a workflow to any resource, and a grid resource may have one or more processors. All processors of a grid resource are reachable from its queuing system and are assumed identical. Therefore, a task in the grid resource queue can be assigned to any of its processors. The time a task spends on a processor of a grid resource is known as service time. Mostly in the queuing system, it is assumed that the service times are also IID random variables. Most often, the exponential distribution is used to calculate service times. Due to the heterogeneity of grid resources service times of grid resources cannot be identical in a grid environment. So the grid resources are divided into groups on the basis of identical service rates and for each group there is a single queue. In this study, it is assumed that each grid resource has a single queue. All the processors of a grid resource are reachable from this queue. The service rates of the grid resources are different from each other.

**Number of resources/processors:** It is assumed that the grid has $M$ number of resources and $R_i$ represents the $i^{th}$ resource of the grid where $1 \leq i \leq M$. Moreover, each grid resource $R_i$ has a definite number of processors denoted as $m_i$, where subscript $i$ is the resource number. It is assumed that all processors in a grid resource $R_i$ are identical and are reachable by a single queue of the grid resource.

**System capacity:** The maximum number of workflows and tasks, which can stay in the queue at grid scheduler level and resource level, are considered as system capacity. Since it is aimed to analyze the queuing characteristics in grid with huge number of workflows therefore the infinite capacity of the queues at grid scheduler and grid resources levels are considered.

**Service discipline:** The ordered in which the workflows and tasks are scheduled is referred as service discipline. Here First Come First Serve (FCFS) service discipline is considered at the grid resource level as well as the workflow scheduler level. The tasks arriving at a grid resource and the workflows arriving at a grid workflow scheduler are placed in the same order as they were received.

**LUs:** The LUs belong to the same organization, which owns the grid resource. Each LU can submit his/her jobs to one and only one grid resource throughout the simulation of an
experiment. Moreover the LUs have no priority over the tasks of workflow and the tasks are served in FCFS service discipline as discussed above.

**Workflows:** The grid workflow scheduler receives the workflows from the grid user and schedules the tasks of the workflows to the grid resources.

**Critical Path (CP):** A critical path in a workflow is defined, as the longest path in a workflow [Kwok and Ahmad, 1999a]. In order to find a critical path [Kwok and Ahmad, 1999a] calculates the ranks of the tasks of a workflow. The upward rank or down word ranks are used to find a critical path. In this thesis the critical path is calculated using the ranking scheme presented in [Topcuouglu et al., 2002]. There can be more than one critical path in a workflow. The paths others than the longest one are simply referred as the second CP₂, third CP₃ and so on.

### 8.4. Grid Resource’s Queue Model Architecture

As explained subsection 6.2.1, in a decentralized grid workflow scheduler architecture more than one grid workflow scheduler is in action. Each of them may submit the tasks of workflow to any resource of the grid. At a grid resource there may be one or more LUs. The LUs may submit the tasks to the grid resource. The tasks of the workflows from the LUs and tasks of the workflows submitted from different Grid Workflow Schedulers follows the Poisson distribution.

Figure 8-2 depicts the split property of the Poisson distribution. According to the split property of Poisson a Poisson distribution having a mean value of \( \lambda \) can be split into \( M \) streams each of them will have Poisson distribution and its mean is equal to \( \lambda_i \) where as \( \lambda_i = \rho_i \lambda \) and \( \rho_i \) is the probability of dividing it into \( i^{th} \) distributions.
Similarly Figure 8-3 depicts the merge property of the Poisson distribution, where $M$ Poisson distributions each with a mean of $\lambda_i$ are joint to form a single Poisson distribution with mean of $\lambda$ where $\lambda = \sum_{i=1}^{M} \lambda_i$. Since the submitted tasks either form the LUs or from the grid workflow schedulers follow the Poisson distribution, therefore at a grid resource $R_i$ the arrival of tasks follows the Poisson distribution with an arrival rate of $\delta_i$.

Figure 8-4 depicts a grid resource $R_i$ which has a single queue and the tasks received from the LUs or from grid workflow schedulers are placed in this queue at Rare. The tasks are placed according to their arrival times and are served from the Front of the queue i.e. the tasks are served in FCFS fashion. It is assumed that each grid resource has a definite number of processors. Let in the grid resource $R_i$ the number of processors is $m_i$. Each
processor of a resource has identical service rate. The service rate of a single processor of
grid resource $R_i$ is denoted as $\mu_i$. The total service rate of the grid resource $R_i$ having $m_i$
processors is $m_i\mu_i$. It is assumed that the queues of the grid resources have infinite
capacity and any task submitted to a grid resource will not drop due to finite capacity of
the grid resource’s queue. So the queue model of the grid resource follows the M/M/m
queue model, where each server (processor of the grid resource) has an identical service
rate of $\mu_i$.

8.4.1. Variables analysis
In this section, some important variables are defined and some important relationships
among these variables are explained.

The inter arrival time between the two successive tasks at a grid resource $R_i$ is
symbolized as $\tau_i$.

$\delta_i$ is the mean arrival rate of tasks at $i$th resource of the grid and it is equal to
$\frac{1}{E[\tau_i]}$.

The service time of a task submitted to a grid resource $R_i$ is given as $S_i$.

The mean service rate of a single processor of the grid resource $R_i$ is as $\mu_i = \frac{1}{E[S_i]}$, since
there are $m_i$ processors in the grid resource $R_i$ therefore, the service rate for $m_i$
processors is $m_i \mu_i$.

The number of tasks queued at a grid resource $R_i$ are represented as $n_{iq}$.

The number of tasks getting the service (i.e., in execution) at a grid resource $R_i$ is
represented as $n_{ts}$.

Total number of tasks at a grid resource $R_i$ is $n_i$, this include the tasks waiting in the
resource’s queue and the tasks being executed.

The queue’s wait time of a task when submitted to a grid resource $R_i$ is given as $w_{iq}$.

The response time of a task, when it is submitted to the grid resource $R_i$ is represented
as $\tau_i$ this includes the wait time and the execution time.
All the above variables except $\delta_i$ and $\mu_i$ are random variables. There are many relationships among these variables, and each relationship defines certain characteristics of an M/M/M queueing system.

### 8.4.2. Rules for grid resource queue

The following rules are followed, while designing and analyzing the grid workflow scheduling system.

1. **Stability condition**: It is assumed that mean arrival rate $\delta_i$ is always less than mean service rate: i.e. $\delta_i < m_i \mu_i$.

2. **Total number of tasks at a grid resource $R_i$ is the sum of the tasks queued and in execution** i.e., $n_i = n_{iq} + n_{is}$. Since $n_i, n_{iq}, n_{is}$ are all random variables therefore the two equation leads to the relationships among their means $E[n_i] = E[n_{iq}] + E[n_{is}]$. Moreover, if the service rate of the grid resource $R_i$ is independent of the number of tasks queued and in execution then the covariance (Cov) is given as $Cov(n_{iq}, n_{is}) = 0$. Also variance of the number of tasks at a grid resource $R_i$ is equal to the sum of the variance of the number of tasks queued and the variance of the number of tasks in execution at the grid resource i.e. $Var(n_i) = Var(n_{iq}) + Var(n_{is})$.

3. **It is assumed that all the grid resources and the grid workflow schedulers have infinite capacity queues. Therefore any workflow submitted to a grid workflow scheduler and any task submitted to any grid resource will not drop due to the finite capacity of the receiving queue.** The mean number of tasks at a grid resource $R_i$ related to the mean response time according to following equations. The Equation 8-1 and Equation 8-2 is also known as “Little law”. The derivation of little law can be found in [Rubinstein and Kroese, 2008].

$$\text{Mean number of tasks at } R_i = \text{arrival rate of tasks at } R_i \times \text{mean response time of } R_i$$  \hspace{1cm} \text{Equation 8-1}
Mean number of tasks queued at $R_i$ = arrival rate of tasks at $R_i \times$ mean waiting time in at $R_i$ queue

4. The response time of a task submitted to a grid resource $R_i$ is the sum of the execution time of the task and the wait time of the task. The response time of a task submitted to a grid resource $R_i$ is given in Equation 8-3.

$$r_i = w_{iq} + S_i$$  \hspace{1cm} \text{Equation 8-3}

5. Since $r_i, w_{iq}, S_i$ are all random variables therefore the relationships among their means is also true and are defined in Equation 8-4, i.e., the mean response time is equal to the sum of mean wait time and mean service time (execution time).

$$E[r_i] = E[w_{iq}] + E[S_i]$$  \hspace{1cm} \text{Equation 8-4}

6. If the service rates of grid resource are independent of the tasks in resource queue then their covariance is equal to zero as given in Equation 8-5.

$$Cov(r_i, w_{iq}) = 0$$  \hspace{1cm} \text{Equation 8-5}

7. Equation 8-6 shows that the variance of the response time at a resource is the sum of the variances of queue wait time and service time.

$$Var(r_i) = Var(w_{iq}) + Var(S_i)$$  \hspace{1cm} \text{Equation 8-6}

8.4.3. Assumptions

The following assumptions are taken about the queues of the grid resources.

1. The queues at the grid resources are of infinite capacity.
2. The FCFS service discipline is followed by all the queues of the grid resources.

3. A task being executed on a grid resource cannot be interrupted.

4. The grid resources do not join or leave the grid environment during the execution of an experiment, i.e., failure of grid resources is beyond the scope of this thesis.

5. The grid resources have space shared scheduling policy [Buyya and Murshed, 2002].

6. The workflow’s tasks can be executed on any of the grid resource i.e., there is no compatibility issues like specific hardware or software requirement for the workflow’s tasks.

7. Every task of a workflow executes on a single processor.

### 8.5. Mathematical Modeling of a Grid Resource Queue Wait Time

This section presents a mathematical model to calculate the mean queue wait time at a grid resource $R_i$. This mean queue wait time is then used by a workflow scheduling algorithm for taking the decisions of task to grid resource mapping. In following table a list of notations and their explanations are given. These variables are used in the queue wait time prediction.

<table>
<thead>
<tr>
<th>List of notations</th>
<th>Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_i$</td>
<td>Arrival rate at a grid resource $R_i$</td>
</tr>
<tr>
<td>$S_i$</td>
<td>Service time of a single processor of a grid resource $R_i$</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>Service rate of a single processor of a grid resource $R_i$</td>
</tr>
<tr>
<td>$m_i$</td>
<td>Number of processors in a grid resource $R_i$</td>
</tr>
<tr>
<td>$n_{iq}$</td>
<td>Number of tasks queued at a grid resource $R_i$</td>
</tr>
<tr>
<td>$n_{is}$</td>
<td>Number of tasks in execution at a grid resource $R_i$</td>
</tr>
<tr>
<td>$n_i$</td>
<td>Total number of tasks at a grid resource $R_i$</td>
</tr>
<tr>
<td>$w_{iq}$</td>
<td>Queue wait time at a grid resource $R_i$</td>
</tr>
<tr>
<td>$r_i$</td>
<td>Response time of a task submitted to a grid resource $R_i$</td>
</tr>
<tr>
<td>$U_i$</td>
<td>Utilization of the grid resource $R_i$</td>
</tr>
<tr>
<td>$p_{oi}$</td>
<td>Probability of the resource idle time of a grid resource $R_i$</td>
</tr>
<tr>
<td>$p_{mi}$</td>
<td>Probability of that there are $m$ number of tasks at a grid resource $R_i$</td>
</tr>
<tr>
<td>$\varphi_i$</td>
<td>Probability of queuing at a grid resource $R_i$</td>
</tr>
</tbody>
</table>
8.5.1. Erlang B formula for grid resource

As described above $S_i$ is a random variable representing the service time of a single processor of the grid resource $R_i$, it is given in Equation 8-7.

$$E[S_i] = \frac{1}{\mu_i} \quad \text{Equation 8-7}$$

The service rate of a single processor of the grid resource $R_i$ is represented as $\mu_i$, where subscript $i$ represents the resource number. In Figure 8-5 the state transition diagram of the grid resource $R_i$ is shown, it is a stochastic infinite state machine. The circles are states representing that there are (0, 1, 2...) number of tasks on the resource $R_i$. Each transition here corresponds with an arrival of a task or departure of a task from the lower state to a higher state or from a higher state to a lower state respectively. In this case the tasks leaving the resource or entering in the resource just by one, this type of model is also known as birth-death model [Ivo and Jacques, 2001]. The models where, more than one tasks may enter or leave leads to a state transition diagram where many other states are reachable from one state it details can be seen in [Cooper, 1976], however, this type of model is not a focus here. As shown in Figure 8-5, it can be observed that, above state $m$ all departure transitions have value $m_i \mu_i$ because as the number of tasks on resource $R_i$ reaches to $m$ or greater than $m$, then almost $m_i$ processors are in busy state and this leads to the departure rate of $m_i \mu_i$. For stability condition, it is considered that $\delta_i < m_i \mu_i$. The utilization of resource $R_i$ of the grid is represented as $U_i$ and it is given in Equation 8-8.
The probability of idle time of the resource $R_i$ is represented as $p_{0i}$ and it is given in Equation 8-9.

$$p_{0i} = 1 - U_i$$

Equation 8-9

The state of the grid resource $R_i$ is represented by the number of tasks at the resource. From the state transition diagram shown in Figure 8-5, the number of tasks at the grid resource seems to follow a birth death process and the probability of a number of tasks at the resources can be calculated as follows.

$$\delta_i p_{0i} = \mu_i p_{0i}$$

$$\delta_i p_{1i} = 2\mu_i p_{1i}$$

$$\delta_i p_{2i} = 3\mu_i p_{2i}$$

$$\ldots \ldots \ldots$$

$$\delta_i p_{n-1} = n\mu_i p_{n-1}$$

$$\ldots$$

$$\delta_i p_{m-1} = m\mu_i p_{m-1}$$

Equation 8-10

Equation 8-11

The following equations are produced by solving the above equations algebraically, and chaining the solutions, whereas these equations are subjected to condition i.e. $n_i \leq m_i$.

$$p_{1i} = \left(\frac{\delta_i}{\mu_i}\right) p_{0i}$$

$$p_{2i} = \left(\frac{\delta_i}{2\mu_i}\right) p_{1i} = \frac{1}{2} \left(\frac{\delta_i}{\mu_i}\right)^2 p_{0i}$$

$$p_{3i} = \left(\frac{\delta_i}{3\mu_i}\right) p_{2i} = \frac{1}{6} \left(\frac{\delta_i}{\mu_i}\right)^3 p_{0i}$$

$$\ldots \ldots \ldots$$

$$p_{n_i} = \left(\frac{\delta_i}{n_i\mu_i}\right) p_{n_i-1} = \frac{1}{n_i!} \left(\frac{\delta_i}{\mu_i}\right)^{n_i} p_{0i} \text{ For } n_i \leq m_i$$

Equation 8-12

$$p_{m_i} = \left(\frac{\delta_i}{m_i\mu_i}\right) p_{m_i-1} = \frac{1}{m_i!} \left(\frac{\delta_i}{\mu_i}\right)^{m_i} p_{0i} \text{ For } n_i \leq m_i$$

Equation 8-13
The term \( p_{ni} \) is the general term here for all \( n_i \leq m_i \). For states below or at \( n_i \leq m_i \) the resource \( R_i \) model is just like M/M/m_i/m_i queuing system, i.e., having a finite queue of size \( m_i \). For states above the state \( m_i \):

\[
\delta_i p_{m_i} = m_i \mu_i p_{m_i+1} \\
\delta_i p_{m_i+1} = m_i \mu_i p_{m_i+2} \\
\delta_i p_{m_i+2} = m_i \mu_i p_{m_i+3} \\
\ldots \ldots
\]

Again solving the equations algebraically, and chaining the solutions the following equations are obtained subjected to the condition that \( n_i > m_i \).

\[
p_{m_i+1} = \frac{1}{m_i} \left( \frac{\delta_i}{\mu_i} \right) p_{m_i}
\]

\[
p_{m_i+2} = \frac{1}{m_i} \left( \frac{\delta_i}{\mu_i} \right) p_{m_i+1} = \frac{1}{m_i^2} \left( \frac{\delta_i}{\mu_i} \right)^2 p_{m_i}
\]

\[
p_{m_i+3} = \frac{1}{m_i} \left( \frac{\delta_i}{\mu_i} \right) p_{m_i+2} = \frac{1}{m_i^3} \left( \frac{\delta_i}{\mu_i} \right)^3 p_{m_i}
\]

\[
p_{ni} = \left( \frac{\delta_i}{m_i \mu_i} \right)^{n_i-m_i} p_{m_i} \quad \text{For} \ n_i > m_i \quad \text{Equation 8-14}
\]

The result of substituting the value of \( p_{m_i} \) from Equation 8-13 in Equation 8-14 the resulting equations given as follows.

\[
p_{ni} = \left( \frac{\delta_i}{m_i \mu_i} \right)^{n_i-m_i} \left( \frac{1}{m! \mu_i} \right)^{m_i} p_{0i} \quad \text{For} \ n_i > m_i \quad \text{Equation 8-15}
\]

\[
p_{ni} = \frac{1}{m_i} \left( \frac{\delta_i}{m_i \mu_i} \right)^{n_i} p_{0i} \quad \text{For} \ n_i > m_i
\]

\[
p_{ni} = \frac{m_i^{m_i}}{m_i!} \left( \frac{\delta_i}{m_i \mu_i} \right)^{n_i} p_{0i} \quad \text{For} \ n_i > m_i \quad \text{Equation 8-16}
\]

\[
p_{ni} = \frac{m_i^{m_i}}{m_i!} U_i^{n_i} p_{0i} \quad \text{For} \ n_i > m_i
\]

Substituting the values of \( p_{ni} (n_i \leq m_i, n_i > m_i) \) in the normalized Equation 8-17 and solving it for \( p_{0i} \), the solution is given in Equation 8-18.

\[
p_{0i} + p_{1i} + p_{2i} + \ldots + p_{ni} + \ldots = 1 \quad \text{Equation 8-17}
\]
The probability that all the processors of the resource $R_i$ of the grid are busy is an important

$$p_{oi} = \left[ 1 + \sum_{n_i=1}^{m_i} \left( \frac{\delta_i}{\mu_i} \right)^{n_i} + \sum_{n_i=m_i}^{\infty} \frac{m_i^{m_i}}{m_i!} U_i^{n_i} \left( \frac{1}{1-U_i} \right) \right]^{-1}$$

Equation 8.18

The identity $\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$
performance measure. It is the fraction of time that an arriving task will be dismissed or blocked until the availability of a free processor at resource $R_i$. In this case the performance measure is simply the $p_{mi}$.

$$p_{mi} = \frac{1}{m_i!} \left( \frac{\delta_i}{\mu_i} \right)^{m_i} \frac{1}{1 + \sum_{n_i=1}^{m_i} \frac{1}{n_i!} \left( \frac{\delta_i}{\mu_i} \right)^{n_i}}$$

Equation 8-19

This is true, when the grid resource has not any queue attached with. This equation is also known as the “Erlang B” formula.

8.5.2. Erlang C Formula for grid resource

Since it is supposed that each grid resource has a queue, where the incoming tasks are placed until all the processors of a resource are busy. So in the case of the grid resource $R_i$, the incoming tasks have to wait in its queue until all $m_i$ processors are busy. The sum of probabilities that there are $(m_i, m_i + 1, m_i + 2, ...)$ tasks on the resource $R_i$ is actually the probability that an upcoming task will wait in the resource queue before it is placed on a free processor. The probability of queuing at the resource is represented as Prob[ Queue at $R_i$] and it is denoted by $\varphi_i$. $\varphi_i$ is an important factor, and it is used in almost all calculations of queuing theory and hence it is calculated before all of them.

$$Prob[ Queue at R_i] = \varphi_i = \sum_{n_i=m_i}^{\infty} p_{n_i}$$

Equation 8-20

Here the states $n_i \geq m_i$ are involved this type of formula is known as “Erlang C” formula, it is given in Equation 8-22. If only the probabilities$(m_i + 1, m_i + 2, ...)$ terms are used then the term $m_i \mu_i$ becomes equal to $\delta_i$, and this violates the stability condition, which states that arrival rate always remain less than the service rate.

$$\varphi_i = \frac{1}{m_i!} \left( \frac{\delta_i}{\mu_i} \right)^{m_i} p_{0i} + \sum_{n_i=m_i+1}^{\infty} \frac{1}{m_i! m_i^{n_i-m_i} \left( \frac{\delta_i}{\mu_i} \right)^{n_i}} p_{0i}$$

$$\varphi_i = \frac{1}{m_i!} \left( \frac{\delta_i}{\mu_i} \right)^{m_i} p_{0i} \left( 1 + \sum_{n_i=m_i+1}^{\infty} \left( \frac{\delta_i}{m_i \mu_i} \right)^{n_i-m_i} \right)$$

8.5.3. Grid resource’s mean queue length

The expected number of tasks queued at the grid resource \( R_i \) is given in Equation 8-29. It shows that when the numbers of tasks are greater than \( m_i \) (number of processors) the queue begins to grow.

\[
\varphi_i = \frac{1}{m_i!} \left( \frac{\delta_i}{\mu_i} \right)^{m_i} p_{oi} \left( 1 + \frac{\delta_i}{m_i \mu_i} + \left( \frac{\delta_i}{m_i \mu_i} \right)^2 + \cdots \right) \quad \text{Equation 8-21}
\]

\[
\varphi_i = \frac{1}{m_i!} \left( \frac{\delta_i}{m_i \mu_i} \right)^{m_i} m_i \mu_i p_{oi} \left( \frac{1}{1 - \frac{\delta_i}{m_i \mu_i}} \right) \quad \text{Equation 8-22}
\]

\[
\varphi_i = \frac{(m_i U_i)^{m_i} p_{oi}}{m_i!} \left( \frac{1}{1 - U_i} \right)
\]

\[
E[n_{iq}] = \sum_{n_i = m_i + 1}^{\infty} (n_i - m_i) p_{ni} \quad \text{Equation 8-23}
\]

\[
E[n_{iq}] = \sum_{j=1}^{\infty} j \ p_{j+m_i} \quad \text{let } j = n_i
\]

\[
E[n_{iq}] = \sum_{j=1}^{\infty} j \ \frac{m_i U_i^{j+m_i}}{m_i!} p_{oi} \quad \text{By substituting the value of } p_{j+m_i}
\]

\[
E[n_{iq}] = \frac{m_i^{m_i}}{m_i!} U_i^{m_i+1} p_{oi} \sum_{j=1}^{\infty} j \ U_i^{j-1}
\]

\[
E[n_{iq}] = \frac{m_i^{m_i}}{m_i!} U_i^{m_i+1} p_{oi} \sum_{j=1}^{\infty} \frac{d}{dU_i} (U_i^j)
\]
Grid Workflow Scheduling Based on Queuing Theory

The identity (referred on page 213) is used for calculation in Equation 8-24 and using the differential quotient formula the Equation 8-25 is obtained. Finally the expected number of jobs in the resource queue given in Equation 8-26, which, is obtained after substituting the value of \( \phi_i \) in Equation 8-25 from Equation 8-22.

\[
E[n_{iq}] = \frac{U_i \phi_i}{1 - U_i}
\]

Equation 8-26

8.5.4. Grid resource’s mean number of tasks in execution

The expected number of tasks in execution is given in Equation 8-27.

\[
E[n_{is}] = \sum_{n_i=1}^{m_i-1} n_i p_{ni} + \sum_{n_i=m_i}^{\infty} m_i p_{ni}
\]

Equation 8-27

\[
E[n_{is}] = \left( p_{ni} + \frac{2}{2!} \frac{\delta_i}{\mu_i} p_{ni} + \frac{3}{3!} \frac{\delta_i}{\mu_i}^2 p_{ni} + \cdots + \frac{(m_i - 1)}{(m_i - 1)!} \frac{\delta_i}{\mu_i}^{m_i-1} p_{ni} \right) + m_i \phi_i
\]

\[
E[n_{is}] = \left( \frac{\delta_i}{\mu_i} p_{ni} + \frac{\delta_i}{\mu_i} p_{ni} + \frac{1}{2!} \frac{\delta_i}{\mu_i}^2 p_{ni} + \cdots + \frac{1}{(m_i - 2)!} \frac{\delta_i}{\mu_i}^{m_i-2} p_{ni} \right) + m_i \phi_i
\]

\[
E[n_{is}] = m_i \frac{\delta_i}{m_i \mu_i} \left( p_{ni} + m_i \frac{\delta_i}{m_i \mu_i} p_{ni} \right) + \frac{1}{2!} \frac{\delta_i}{m_i \mu_i}^2 p_{ni} + \cdots + \frac{1}{(m_i - 2)!} \frac{\delta_i}{m_i \mu_i}^{m_i-2} p_{ni} \right) + m_i \phi_i
\]

\[
E[n_{is}] = m_i U_i \left( p_{ni} + m_i U_i p_{ni} \right) \frac{1}{2!} \left( m_i U_i \right)^2 p_{ni} + \cdots + \frac{1}{(m_i - 2)!} \left( m_i U_i \right)^{m_i-2} p_{ni} \right) + m_i \phi_i
\]
8.5.5. Grid resource’s mean number of tasks

Expected number of tasks at a grid resource $R_i$ is given Equation 8-29. It is actually the sum of the expected number of tasks in execution on the said resource and the expected number of tasks queued.

$$E[n_i] = m_i U_i + \frac{U_i \varphi_i}{1 - U_i} \quad \text{Equation 8-29}$$

8.5.6. Variance of number of tasks in the grid resource’s queue

The variance of number of tasks in grid resource queue is given in Equation 8-30.

$$Var(n_{iq}) = E[n_{iq}^2] - (E[n_{iq}])^2 \quad \text{Equation 8-30}$$

Considering the Equation 8-23 the value of $E[n_{iq}^2]$ is given in Equation 8-31.

$$E[n_{iq}^2] = \sum_{n_i = m_i + 1}^{\infty} (n_i - m_i)^2 p_{n_i} \quad \text{Equation 8-31}$$

$$E[n_{iq}^2] = \sum_{j=1}^{\infty} j^2 p_{j+m_i} \quad \text{Let } j = n_i$$

$$E[n_{iq}^2] = \frac{(U)m_i^{m_i}}{m_i!} \sum_{j=1}^{\infty} j^2 U_i \quad \text{By substituting the value of } p_{j+m_i}$$
Using an identity from [Barner and Flohr, 1987] the above equation can be written as follows.

\[
E[n_{iq}^2] = \frac{(U_i m_i)^{m_i}}{m_i!} \sum_{j=1}^{\infty} j^2 U_i = \frac{(U_i m_i)^{m_i}}{m_i!} \left( \frac{2}{(1-U_i)^3} - 3 \sum_{j=0}^{\infty} jU_i^j - 2 \sum_{j=0}^{\infty} U_i^j \right)
\]

\[
E[n_{iq}^2] = \frac{(U_i m_i)^{m_i}}{m_i!} \left( \frac{2}{(1-U_i)^3} - 3 \left( \frac{U_i}{(1-U_i)^2} \right) - 2 \frac{U_i}{1-U_i} \right)
\]

\[
E[n_{iq}^2] = \frac{(U_i m_i)^{m_i}}{m_i!} \left( \frac{2 - 3U_i(1-U_i) - 2(1-U_i)^2}{(1-U_i)^3} \right)
\]

\[
E[n_{iq}^2] = \frac{(U_i m_i)^{m_i}}{m_i!} \left( \frac{U_i(1+U_i)}{(1-U_i)^3} \right)
\]

By replacing the value of \( \varphi_i \) from Equation 8-22 the above equation can be written as follows.

\[
E[n_{iq}^2] = \frac{\varphi_i U_i (1+U_i)}{(1-U_i)^2}
\]  
Equation 8-32

Substituting the values of \( E[n_{iq}^2] \) and \( (E[n_{iq}])^2 \) from Equation 8-32 and Equation 8-26 in Equation 8-30 following equation is achieved.

\[
Var(n_{iq}) = \frac{\varphi_i U_i (1+U_i)}{(1-U_i)^2} - \frac{U_i^2 \varphi_i^2}{(1-U_i)^2}
\]

Simplifying the above equation \( Var(n_{iq}) \) is given in Equation 8-33.

\[
Var(n_{iq}) = \varphi_i U_i \left( \frac{1+U_i - \varphi_i U_i}{(1-U_i)^2} \right)
\]  
Equation 8-33

8.5.7. Variance of number of tasks in service at a grid resource

The variance of number of tasks in service at a grid resource \( R_i \) is given in. Equation 8-34.

\[
Var(n_{is}) = E[n_{is}^2] - (E[n_{is}])^2
\]  
Equation 8-34

4. \( \sum_{i=0}^{\infty} i^2 x^i = \frac{2}{(1-x)^3} - 3 \sum_{i=0}^{\infty} i x^i - 2 \sum_{i=0}^{\infty} x^i \)
The value of $E[n_is^2]$ can be achieved by using the Equation 8-27 and it is given as follows.

$$E[n_is^2] = \sum_{n_i=1}^{m_i-1} n_i^2 p_{n_i} + \sum_{n_i=m_i}^{\infty} m_i^2 p_{n_i}$$

$$E[n_is^2] = (p_{i1} + 2^2 p_{2i} + 3^2 p_{3i} + 4^2 p_{4i} + \cdots + (m_i - 1)^2 p_{(m_i-1)i}) + m_i^2 \sum_{n_i=m_i}^{\infty} p_{n_i}$$

$$E[n_is^2] = \left(\frac{\delta_i}{\mu_i} p_{0i} + \frac{2^2}{2} \left(\frac{\delta_i}{\mu_i}\right)^2 p_{0i} + \frac{3^2}{6} \left(\frac{\delta_i}{\mu_i}\right)^3 p_{0i} + \cdots + \frac{(m_i - 1)^2}{(m_i - 1)!} \left(\frac{\delta_i}{\mu_i}\right)^{(m_i-1)} p_{0i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = \left(\frac{\delta_i}{\mu_i} p_{0i} + \frac{2}{2} \left(\frac{\delta_i}{\mu_i}\right)^2 p_{0i} + \frac{3}{6} \left(\frac{\delta_i}{\mu_i}\right)^3 p_{0i} + \cdots + \frac{(m_i - 1)}{(m_i - 2)!} \left(\frac{\delta_i}{\mu_i}\right)^{(m_i-2)} p_{0i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = \left(\frac{\delta_i}{m_i \mu_i} p_{0i} + 2 m_i \left(\frac{\delta_i}{m_i \mu_i}\right)^2 p_{0i} + \frac{3 m_i^2}{2!} \left(\frac{\delta_i}{m_i \mu_i}\right)^2 p_{0i} + \cdots + \frac{(m_i - 1)m_i^{m_i-2}}{(m_i - 2)!} \left(\frac{\delta_i}{m_i \mu_i}\right)^{(m_i-2)} p_{0i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = m_i U_i \left(\frac{\delta_i}{m_i \mu_i} p_{0i} + 2 m_i \left(\frac{\delta_i}{m_i \mu_i}\right)^2 p_{0i} + \frac{3 m_i^2}{2!} \left(\frac{\delta_i}{m_i \mu_i}\right)^2 p_{0i} + \cdots + \frac{(m_i - 1)}{(m_i - 2)!} \left(\frac{\delta_i}{m_i \mu_i}\right)^{(m_i-2)} p_{0i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = m_i U_i \left(\sum_{n_i=1}^{m_i-1} n_i p_{(n_i-1)i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = m_i U_i \left(\sum_{n_i=1}^{m_i-2} (n_i + 1) p_{n_i}\right) + m_i^2 \varphi_i$$

$$E[n_is^2] = m_i U_i \left(\sum_{n_i=1}^{m_i-2} n_i p_{n_i} + \sum_{n_i=1}^{m_i-2} p_{n_i}\right) + m_i^2 \varphi_i$$

Since $m_i \varphi_i (1 - U_i) = m_i U_i p_{(m_i-1)i} = m_i p_{m_i}$

$$E[n_is^2] = m_i U_i (1 - p_{(m_i-1)i} - \varphi_i) - (m_i - 1) p_{(m_i-1)i}) + m_i U_i \sum_{n_i=1}^{m_i-2} p_{n_i} + m_i^2 \varphi_i$$

$$E[n_is^2] = m_i U_i (m_i U_i (1 - p_{(m_i-1)i} - \varphi_i) - (m_i - 1) p_{(m_i-1)i}) + m_i U_i \sum_{n_i=1}^{m_i-2} p_{n_i} + m_i^2 \varphi_i$$
\[ E[n_{is}^2] = m_i U_i (m_i U_i (1 - \varphi_i) - m_i p_{(m_i-1)i} (1 + U_i) + p_{(m_i-1)i}) + m_i U_i \sum_{n_i=1}^{m_i-2} p_{n_i} \]

\[ + m_i^2 \varphi_i \]

\[ E[n_{is}^2] = m_i U_i \left( m_i U_i (1 - \varphi_i) - \frac{m_i \varphi_i (1 + U_i) (1 - U_i)}{U_i} + \varphi_i (1 + U_i) \right) + m_i U_i \sum_{n_i=1}^{m_i-2} p_{n_i} \]

\[ + m_i^2 \varphi_i \]

\[ E[n_{is}^2] = m_i \left( m_i U_i^2 - m_i \varphi_i (1 - U_i^2) + \varphi_i (1 + U_i) \right) + m_i U_i \sum_{n_i=1}^{m_i-2} p_{n_i} + m_i^2 \varphi_i \]

\[ E[n_{is}^2] = m_i \left( m_i U_i^2 - m_i \varphi_i + \varphi_i U_i \right) + m_i U_i \left( 1 - \sum_{n_i=m_i}^{\infty} p_{n_i} - p_{n_{(i-1)i}} \right) + m_i^2 \varphi_i \]

\[ E[n_{is}^2] = m_i^2 U_i^2 + m_i \varphi_i + m_i U_i - m_i U_i p_{n_{(i-1)i}} \]

\[ E[n_{is}^2] = m_i^2 U_i^2 + m_i \varphi_i + m_i U_i - m_i \varphi_i (1 - U_i) \]

\[ E[n_{is}^2] = m_i^2 U_i^2 + m_i U_i + m_i \varphi_i U_i \]  \hspace{1cm} \text{Equation 8-35}

Substituting the values of \( E[n_{is}^2] \) and \((E[n_{is}])^2\) from Equation 8-35 and Equation 8-28 in Equation 8-34 following equation is achieved.

\[ Var(n_{is}) = m_i U_i + m_i \varphi_i U_i \]  \hspace{1cm} \text{Equation 8-36}

\[ 8.5.8. \text{Variance of number of tasks at a grid resource} \]

The variance of the number of total tasks at a grid resource is given by the sum of variance of tasks queued and the variance of the number of tasks in execution. The value of \( Var(n_i) \) is given Equation 8-37, which is a sum of Equation 8-36 and Equation 8-33.

\[ Var(n_i) = m_i U_i + \varphi_i U_i \left[ \frac{1 + U_i - \varphi_i U_i}{(1 - U_i)^2} + m_i \right] \]  \hspace{1cm} \text{Equation 8-37}
8.5.9. Expected response time of a task submitted to a grid resource

The mean response time of a task submitted to a resource $R_i$ denoted as $E[r_i]$ according to little law [Ivo and Jacques, 2001; Ng and Boon-Hee, 2008] can be achieved by dividing the mean number of tasks in the resource (including tasks in execution and tasks waiting in the resource queue) by the mean arrival rate as follows.

$$E[r_i] = \frac{E[n_i]}{\delta_i} = \frac{1}{\delta_i} \left( m_i U_i + \frac{U_i \varphi_i}{1 - U_i} \right)$$

Since $U_i = \frac{\delta_i}{m_i \mu_i}$, therefore the above equation can be written as follows.

$$E[r_i] = \frac{1}{\mu_i} \left( 1 + \frac{\varphi_i}{m_i (1 - U_i)} \right) \quad \text{Equation 8-38}$$

8.5.10. Expected queue wait time of a task submitted to the grid resource

The mean queue wait time of a task submitted to a resource $R_i$ denoted as $E[w_{iq}]$. According to little law [Ivo and Jacques, 2001; Ng and Boon-Hee, 2008] it can be achieved by dividing the mean number of tasks in the queue by mean arrival rate as follows.

$$E[w_{iq}] = \frac{E[n_{iq}]}{\delta_i} = \frac{1}{\delta_i} \left( \frac{U_i \varphi_i}{1 - U_i} \right)$$

Using the value of $U_i = \frac{\delta_i}{m_i \mu_i}$, the above equation can be simplified to Equation 8-39.

$$E[w_{iq}] = \frac{\varphi_i}{m_i \mu_i (1 - U_i)} \quad \text{Equation 8-39}$$

8.6. Realization of Queue Wait Time Prediction

The queue wait time measurement at a grid resource $R_i$ requires the two parameters i.e., arrival rate at a grid resource and the service rate of the grid resource. The arrival rate at a grid resource and the service rate at a grid resource may vary over the time and so is the queue wait time of the grid resource. In this section the calculation of the arrival rate at a grid resource in a specific time interval and the service rate of the grid resource in a
specific time interval is calculated, these parameters are then used to predict the arrival rate at a grid resource for the next interval.

The arrival rate at a grid resource $R_i$ in a time interval $\Gamma$ is denoted as $\delta_{i,\Gamma}$. The service rate of the grid resource $R_i$ in a time interval $\Gamma$ is denoted by $\mu_{i,\Gamma}$, this is the total service rate of all the processors of the grid resource. In this thesis the simulation-based approach is used for evaluation and the GridSim simulator is selected as a simulation tool. When a task is submitted to a grid resource, the GridSim treats it as an object of the class `ResGridlet`, which represents the tasks submitted to a grid resource. The grid resource assigns an identification number to each incoming task. This identification number is consecutive and is assigned according to the arrival time of the tasks. When a task completes its execution the grid resource sends its results to the corresponding grid workflow scheduler. The results include the number of processors assigned to the tasks, execution start time and end time, identification assigned to it at a grid resource, the number of resources this task visited during execution its wait time in the queue etc. In order to measure the arrival rate at a grid resource $R_i$ in a time interval $\Gamma$ the grid workflow scheduler at the beginning of a time interval sends a task with negligible computation weight (since the zero computation weight of a task is not allowed to send on a grid resource in GridSim). When the results of the task are received the identification number of the task at the grid resource is saved as $intervalBeginID_{i,\Gamma}$. At the end of the interval another task is sent to the grid resource and its identification number at the grid resource is marked as $intervalEndID_{i,\Gamma}$. The arrival rate at a grid resource $R_i$ in a time interval $\Gamma$ is calculated as a difference between $intervalEndID_{i,\Gamma}$ and $intervalBeginID_{i,\Gamma}$ it is given Equation 8-40.

$$\delta_{i,\Gamma} = intervalEndID_{i,\Gamma} - intervalBeginID_{i,\Gamma}. \quad \text{Equation 8-40}$$

The arrival rate at the grid resource $R_i$ in a time interval $\Gamma + 1$ is predicted using the exponential smoothing method. In this method the history values are used to predict a value of the future. Some methods take all the previous values into account and some may prefer to use a subset of the history values. However the sum of the smoothing constant must be equal to one. Further details of the exponential smoothing can be [Box
and Jenkins, 1990; Mitsa, 2010]. Since it is observed that the arrival rate at a grid resource is dependent only upon recent history measurements and therefore only three previous intervals’ measurements are taken into account and the most recent history is waited more. The Equation 8-41 gives the value predicted arrival rate at a grid resource $R_i$ in a time interval $\Gamma + 1$.

$$
P_{-\delta_i,\Gamma+1} = 0.6 \times \delta_{i,\Gamma} + 0.3 \times \delta_{i,\Gamma-1} + 0.1 \times \delta_{i,\Gamma-2}
$$

Equation 8-41

The predictor at a grid workflow scheduler periodically gets the value of tasks at the grid resources as discussed in subsection 7.3.8. The GIS gives a value of $\text{currentTasks}(R_i)$. At the end of the interval the mean number of tasks at a grid resource is calculated by dividing the sum of the values $\text{currentTasks}(R_i)$ by the number of observations made in the interval. This value is named as $\text{tasksAtResource}(R_i, \Gamma)$ by the number of observations made in the interval. This value is then calculated according to Equation 8-42.

$$
\hat{\mu}_{i,\Gamma} = \text{tasksAtResource}(R_i, \Gamma-1) + \delta_{i,\Gamma} + \text{tasksAtResource}(R_i, \Gamma)
$$

Equation 8-42

Using the values of $\hat{\mu}_{i,\Gamma}$ and the previous history values $\hat{\mu}_{i,\Gamma+1}$ is then calculated according to the exponential smoothing method according to Equation 8-43.

$$
P_{-\hat{\mu},\Gamma+1} = 0.6 \times \hat{\mu}_{i,\Gamma} + 0.3 \times \hat{\mu}_{i,\Gamma-1} + 0.1 \times \hat{\mu}_{i,\Gamma-2}
$$

Equation 8-43

Using the values obtained through Equation 8-41 and Equation 8-43 the queue wait time at a grid resource is predicted using the Equation 8-39. The Sections 8.5-8.6 provide the requirement (Requirement R9) to be functional. The next section provides a Queue Wait Time Prediction-based Grid Workflow Scheduling (QWP-GWS) algorithm. This algorithm uses the predicted queue wait time at the grid resources and uses it to map the tasks of a workflow to the grid resources.
8.7. Important models for queue wait time scheduling

In this section some important models used in the QWP-GWS algorithm are discussed. These models include workflow model, resource model, average computation time of the tasks of a workflow, average communication time of the edges of a workflow and critical paths in a workflow. In following list the notations used in these models and their explanation is given.

<table>
<thead>
<tr>
<th>List of notations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G$</td>
<td>Workflow</td>
</tr>
<tr>
<td>$T$</td>
<td>Set of tasks in $G$</td>
</tr>
<tr>
<td>$t_i$</td>
<td>Task number $i$ of a workflow $G$</td>
</tr>
<tr>
<td>$E$</td>
<td>Set of Edges in $G$</td>
</tr>
<tr>
<td>$e_{i,j}$</td>
<td>Edge joining the tasks number $i$ and $j$</td>
</tr>
<tr>
<td>$W$</td>
<td>Set of weight of the task of $G$</td>
</tr>
<tr>
<td>$C$</td>
<td>Set of weight of the edges of $G$</td>
</tr>
<tr>
<td>$M$</td>
<td>Number of total resources in Grid</td>
</tr>
<tr>
<td>$R$</td>
<td>Set of resources</td>
</tr>
<tr>
<td>$R_r$</td>
<td>Resource number $r$ of the Grid</td>
</tr>
<tr>
<td>$\gamma_r$</td>
<td>The number of processors in resource $r$</td>
</tr>
<tr>
<td>$Speed(P_r)$</td>
<td>Processing speed of a single processor of resource number $r$</td>
</tr>
<tr>
<td>$PC(R_r)$</td>
<td>Processing capacity of the grid resource number $r$</td>
</tr>
<tr>
<td>$a_r$</td>
<td>Number of memory units in the grid resource number $r$</td>
</tr>
<tr>
<td>$MemorySpace(MS_{k,r})$</td>
<td>Memory capacity of the $k$th memory unit of the grid resource number $r$</td>
</tr>
<tr>
<td>$MC(R_r)$</td>
<td>Memory capacity of the grid resource number $r$</td>
</tr>
<tr>
<td>$TT_{k,r}$</td>
<td>Unit data transfer time between the grid resource $k$ and $r$</td>
</tr>
<tr>
<td>$\bar{W}_i$</td>
<td>Average computation time of task $t_i$</td>
</tr>
<tr>
<td>$c_{j,i}$</td>
<td>Average communication time of the edge $e_{j,i}$</td>
</tr>
<tr>
<td>$rank_u(t_i)$</td>
<td>Upward rank of the task $t_i$</td>
</tr>
<tr>
<td>$rank_d(t_i)$</td>
<td>Downward rank of the task $t_i$</td>
</tr>
<tr>
<td>$rank(t_i)$</td>
<td>Rank of the task $t_i$</td>
</tr>
<tr>
<td>$CP_1$</td>
<td>Critical path of the workflow</td>
</tr>
<tr>
<td>$CP_2,...,CP_j$</td>
<td>All possible paths in a workflow</td>
</tr>
<tr>
<td>$Q$</td>
<td>Number of tasks in the critical path $CP_1$</td>
</tr>
<tr>
<td>$ECP_1(R_r)$</td>
<td>Expected completion time of the critical path $CP_1$ on grid resource number $r$</td>
</tr>
</tbody>
</table>
8.7.1. Workflow model

It is assumed that each workflow is represented as a directed acyclic task graph $G (T, E, W, C)$.

1. There are a total $N$ number of tasks in a workflow. $T$ is the set of tasks and $t_i$ represents the $i$th task of set $T$ and $1 < i \leq N$.

2. $E$ is the set of edges. The directed edge $e_{i,j}$ joins tasks $t_i$ and $t_j$ and $1 < i, j \leq N$, where task $t_i$ is called the parent task and task $t_j$ is called the child task. This also implies that $t_j$ cannot start until task $t_i$ completes its execution and sends its data to the resource where $t_j$ has to start, $t_j \in \text{child}(t_i)$ where $\text{child}(t_i)$ is the set of all immediate children of $t_i$. Similarly $t_j \in \text{parent}(t_i)$, whereas $\text{parent}(t_i)$ is the set of all immediate parents of $t_i$.

3. $W$ is the set of computations weights of the tasks of workflow, where $w_i$ is the computation weight of $t_i$.

4. $C$ is the set of communications weights of the tasks of workflow, where $c_{i,j}$ is the computation weight $e_{i,j}$. The cardinality of the set $C$ is denoted as $D$, where $D$ is a positive integer value.

8.7.2. Resource model

It is assumed that there are total $M$ resources available in the Grid, denoted by the set $R$. $R_r$ represent the $r$th resource of the Grid where $1 \leq r \leq M$.

1. Each resource is a cluster.

2. Each resource $R_r$ has a definite number of processors denoted by $m_r$.

3. All processors of a resource have equal processing speed.

4. The processing speed of a processor $P_r$ is denoted by $\text{Speed}(P_r)$, where $r$ is the resource number.

5. The processing capacity of the resource $R_r$ is given in Equation 8-44.
PC(R_r) = Speed(P_r) \times m_r \tag{8-44}

6. Each resource R_r has a definite number of memory units present in it and are denoted by \alpha_r. The Memory space of kth memory unit of rth resource is denoted as MemorySpace(MS_{k,r}) where 1 \leq k \leq \alpha_r.

7. The memory capacity of a resource R_r is given in Equation 8-45.

\[ MC(R_r) = \sum_{k=1}^{\alpha_r} \text{MemorySpace}(MS_{k,r}) \tag{8-45} \]

8. It is assumed that the grid resources have two types of network connections, local network connections within a cluster and internet connections. Communications using the internet across the different resources is considered and the network communication within a cluster is assumed negligible.

9. The latencies, bandwidth and unit data transfer rate across the grid resources are denoted as ‘L’, ‘B’ and ‘TT’ respectively and are M \times M matrices. It is assumed that the values of L and B are known and value of TT is calculated using the matrix B. The unit data transfer rate between the resources R_k and R_r also TT_{k,r} = 0 when k = r.

8.7.3. Average computational and communication weights

The average computational and communication weights of a task t_i and an edge e_{j,i} of a workflow are represented as \overline{w}_i and \overline{c}_{j,i} and these are calculated according to Equation 8-46 and Equation 8-48. In Equation 8-46 \text{Exe}(t_i, R_r) is the execution time of the task t_i on resource R_r its value is given in Equation 8-47. The average communication time of the edge e_{j,i}, where task t_i is executed on a resource R_r and task t_j is executed on a resource R_k can be calculated by multiplying communication weight the c_{j,i} to the unit data transfer time and summing the latency value of the link between the two resources. Let all the M resources are capable of executing the tasks t_i and t_j, in this case the average
communication time of a task $e_{j,i}$ is given in Equation 8-48, this equation considers all possible pairs of resources. If there are $M$ resources the number of links between the $M$ resources is $\frac{M(M-1)}{2}$. Being more realistic and assuming that the link between resources are not equal than the number of total links is doubled in this case and it is given as $\frac{2M(M-1)}{2}$, so total number of links are $M^2 - M$. This is used to calculate the average communication time of an edge as shown in Equation 8-48.

$$\bar{W}_i = \frac{\sum_{r=1}^{M} (Exe(t_i, R_r))}{M}$$  \hspace{1cm} \text{Equation 8-46}$$

$$Exe(t_i, R_r) = \frac{w_i}{\text{Speed}(P_r)}$$  \hspace{1cm} \text{Equation 8-47}$$

$$\overline{C}_{j,i} = \frac{\sum_{k=1}^{M} (c_{i,j} \times TT_{r,k} + L_{r,k})}{M^2 - M}$$  \hspace{1cm} \text{Equation 8-48}$$

### 8.7.4. Workflow’s tasks rank

The workflow’s tasks can have two types of ranks namely, upward rank and downward ranks. Both the upward and downward ranks of the task of a workflow are calculated and are represented as $rank_u(t_i)$ and $rank_d(t_i)$ respectively. The upward rank of a task $t_i$ is calculated recursively according to Equation 8-49. Similarly, the downward ranks are recursively calculated according to Equation 8-50. In Equation 8-49 an array of $upward$$_$Child$[t_i]$ is saved which will be later used for calculating the critical path of the workflow.

$$rank_u(t_i) = \begin{cases} 
\bar{W}_i & \text{if } t_i \text{ is exit node} \\
\bar{W}_i + \max_{t_j \in \text{child}(t_i)} (\bar{e}_{i,j} + rank_u(t_j)) & \text{otherwise} \\
\text{upward}$$_$Child$[t_i] = j 
\end{cases}$$  \hspace{1cm} \text{Equation 8-49}$$
According to Zhao and Sakellariou the two different ranking schemes may yield different schedule [Zhao and Sakellariou, 2003]. Therefore instead of using \( \text{rank}_u(t_i) \) or \( \text{rank}_d(t_i) \) alone, the rank of a task \( t_i \) is calculated by summing the \( \text{rank}_u(t_i) \) and \( \text{rank}_d(t_i) \) as given in Equation 8-51.

\[
\text{rank}(t_i) = \text{rank}_u(t_i) + \text{rank}_d(t_i)
\]

8.7.5. Critical paths

After calculating the rank of all the tasks of a workflow all possible paths between the entry and exit node of a workflow are discovered. It is ensured that there is one entry and one exit node in a workflow. If there is more than one entry node or more than one exit node than a dummy entry and dummy exit node is added.
tasks exist then a dummy exit task is added and it is made the child task of all exit tasks with ‘0’ dependency. This phenomena is shown in Figure 8-6, in (a) there are two entry tasks so task 0 is added with 0 computational weight and 0 communication weights with task 1 and 2, (b) there were two exit nodes so task 6 is added as a dummy exit node, in (c) a dummy entry task 0 and a dummy exit task 6 are added. The entry task is referred $t_{entry}$ and the exit task is referred as $t_{exit}$. After ensuring that there is a single entry and single exit node the average computational time of all tasks of a workflow and average communication time of all edges are calculated using the Equation 8-46 and Equation 8-48. After that the rank of each task of the workflow is calculated according to Equation 8-51. The critical path in a workflow is found by the Algorithm 8-1. Since in a workflow there may be more than one critical path and it is necessary to identify all of them. As a workflow scheduling algorithm may fail if it works for one critical path and all the others are even not identified. In this thesis it is assumed that critical path may be more than one but the critical path which has the entry and exit node is only considered as critical path. The remaining paths are simply referred as critical path 2 ($CP_2$) critical path 3 ($CP_3$) and so on.

After getting first critical path all the tasks on the critical path are marked as discovered. Next another path is found in the workflow. The next path starts from a task which has the highest values of its rank and recursively its undiscovered child tasks and undiscovered parent tasks are visited and these are added to $CP_2$. If a task has more than one parent or child task undiscovered than the child task having the greatest value of rank

<table>
<thead>
<tr>
<th>Algorithm 8-1:Find Critical Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROCEDURE</strong> critical_Path(W)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>\hspace{1cm} Max_Rank &lt;- t_{entry} = t_{exit}</td>
</tr>
<tr>
<td>\hspace{1cm} CP_i &lt;- ∅</td>
</tr>
<tr>
<td>\hspace{1cm} t_i &lt;- t_{entry}</td>
</tr>
<tr>
<td>\hspace{1cm} WHILE (t_i ≠ t_{exit}) DO</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>\hspace{2cm} SELECT t_j where t_j = upward_Child[t_j] ^ rank(t_j) = Max_Rank</td>
</tr>
<tr>
<td>\hspace{2cm} CP_i &lt;- CP_i ∪ t_j</td>
</tr>
<tr>
<td>\hspace{2cm} t_i &lt;- -t_j</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>\hspace{1cm} CP_i &lt;- -CP_i ∪ t_{exit}</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>\hspace{1cm} RETURN CP_i</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
is selected to insert in the current path. Once the path reaches at a point where next parent is already discovered and a next child is already discovered the second path is discovered. This process continues until all of the tasks of a workflow are discovered. Any new path number is just an increment from the previously discovered path. If two undiscovered tasks have the same value of their ranks, tie is broken randomly.

Here the procedure of finding all paths is explained by an example. Let’s suppose that there are three resources as shown in Table 8-3. The unit data transfer time and latencies between the resource are given in Table 8-4 and Table 8-5 respectively.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Number of PES</th>
<th>Processing speed of one PE</th>
<th>Storage Units</th>
<th>Storage capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>3</td>
<td>10</td>
<td>2</td>
<td>100</td>
</tr>
<tr>
<td>$R_2$</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>$R_3$</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transfer Time</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$R_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$R_3$</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$R_2$</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$R_3$</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 8-7 depicts an example workflow. For this workflow the mean computation time of all tasks and mean communication time for all edges are calculated and these values are given in Table 8-6 and Table 8-7.

Table 8-6: Mean computation time

<table>
<thead>
<tr>
<th>Task ($t_i$)</th>
<th>Mean Computational Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>5</td>
</tr>
<tr>
<td>$t_2$</td>
<td>5</td>
</tr>
<tr>
<td>$t_3$</td>
<td>6.6</td>
</tr>
<tr>
<td>$t_4$</td>
<td>3.3</td>
</tr>
<tr>
<td>$t_5$</td>
<td>1.6</td>
</tr>
<tr>
<td>$t_6$</td>
<td>1.6</td>
</tr>
<tr>
<td>$t_7$</td>
<td>3.3</td>
</tr>
<tr>
<td>$t_8$</td>
<td>11.6</td>
</tr>
<tr>
<td>$t_9$</td>
<td>13.3</td>
</tr>
<tr>
<td>$t_{10}$</td>
<td>13.3</td>
</tr>
</tbody>
</table>
The upward and downward ranks of the tasks are calculated the rank of each task is the sum of downward and upward ranks. The ranks of the tasks are given in Table 8-8. The value \( \text{upward}_\text{Child}[t_i] \) saved during the calculation of upward ranks of the tasks and it is used for finding the first critical path in the workflow.

### Table 8-7: Mean communication time

<table>
<thead>
<tr>
<th>edge ( (e_{ij}) )</th>
<th>Mean Communication Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_{1,2} )</td>
<td>32</td>
</tr>
<tr>
<td>( e_{1,3} )</td>
<td>32</td>
</tr>
<tr>
<td>( e_{1,4} )</td>
<td>32</td>
</tr>
<tr>
<td>( e_{2,5} )</td>
<td>32</td>
</tr>
<tr>
<td>( e_{2,6} )</td>
<td>32</td>
</tr>
<tr>
<td>( e_{3,7} )</td>
<td>42</td>
</tr>
<tr>
<td>( e_{4,7} )</td>
<td>22</td>
</tr>
<tr>
<td>( e_{5,8} )</td>
<td>12</td>
</tr>
<tr>
<td>( e_{6,8} )</td>
<td>22</td>
</tr>
<tr>
<td>( e_{6,9} )</td>
<td>102</td>
</tr>
<tr>
<td>( e_{7,9} )</td>
<td>202</td>
</tr>
<tr>
<td>( e_{8,10} )</td>
<td>72</td>
</tr>
<tr>
<td>( e_{9,10} )</td>
<td>52</td>
</tr>
</tbody>
</table>

### Table 8-8: Ranks of tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Downward rank</th>
<th>Upward rank</th>
<th>( \text{upward}_\text{Child}[t_i] )</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>0</td>
<td>369.4</td>
<td>( t_3 )</td>
<td>369.4</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>37</td>
<td>219.2</td>
<td>( t_6 )</td>
<td>256</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>37</td>
<td>332.4</td>
<td>( t_7 )</td>
<td>369.4</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>37</td>
<td>308.7</td>
<td>( t_7 )</td>
<td>345.7</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>74</td>
<td>110.5</td>
<td>( t_8 )</td>
<td>184.5</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>74</td>
<td>182.2</td>
<td>( t_8 )</td>
<td>256.2</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>85.5</td>
<td>283.9</td>
<td>( t_9 )</td>
<td>369.4</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>97.6</td>
<td>96.9</td>
<td>( t_{10} )</td>
<td>194.5</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>290.8</td>
<td>78.6</td>
<td>( t_{10} )</td>
<td>369.4</td>
</tr>
<tr>
<td>( t_{10} )</td>
<td>356</td>
<td>13.4</td>
<td></td>
<td>369.4</td>
</tr>
</tbody>
</table>
Figure 8-8: All critical paths in a workflow

According to Algorithm 8-1 the entry tasks and exit tasks are marked as highest rank value which is 369.4. According to line number 5 of algorithm 8-1 starting from the entry task and in line number 6 checking the upward_Child$[t_j]$ is having the rank equal to max_Rank than this child is inserted in CP$_1$. The first task next than entry task which is added in the critical path is $t_3$. Next the child tasks of $t_3$ is checked and task $t_7$ is inserted in the CP$_1$. Recursively $t_9$ and after the loop the exit task which is $t_{10}$ is added in the CP$_1$. All the tasks appearing in CP$_1$ are marked as discovered and after that a task having highest value of its rank and it is not discovered is added in CP$_2$. The task having the highest value of the rank in Figure 8-7(c) is $t_4$. It is then inserted in the CP$_2$. All the parents
and child tasks of the task $t_4$ are already discovered so CP$_2$ contains only one task i.e. $t_4$. After the discovery of $t_4$ the task having the highest value of its rank and it is not discovered yet is $t_2$. The task $t_2$ is selected and is marked as discovered and added in CP$_3$. The task $t_2$ has two child tasks which are not discovered yet. So the current path which is CP$_3$ can go in either direction but a child task of $t_2$ having the greatest value of its rank will be inserted in CP$_3$ and it is task $t_6$. Next the child and parents of $t_6$ which are still undiscovered are checked. There is no parent left which is undiscovered from task $t_6$ and the only child remained undiscovered is $t_8$. Therefore $t_8$ is inserted in CP$_3$. Next the remaining undiscovered tasks are checked there is only one task remained which is $t_5$ and it is inserted in CP$_4$. Figure 8-8 (a) depicts a workflow. Figure 8-8 (b) the critical path is found according to Algorithm 8-1. In next step Figure 8-8 (c) shows a path CP$_2$ is found which consist of single task $t_4$. Finding of another critical path is shown in Figure 8-8 (d) this path consists of three tasks namely $t_2, t_6, t_8$. Last path CP4 consist of only one path $t_5$ and its discovery of is shown in Figure 8-8 (e). The workflow of Figure 8-8 (a) has the following critical paths.

$$CP_1 = \{t_1, t_3, t_7, t_9, t_{10}\}, \ CP_2 = \{t_4\}, \ CP_3 = \{t_2, t_6, t_8, t_9\} \text{ and } CP_4 = \{t_5\}$$

The authors in [Topcuoglu et al., 2002] took a similar assumption that the tasks having the rank equal to the entry task are said to be critical tasks and are assigned to a critical path processor. However, their critical path finding algorithm may fail as there exist many workflows where the tasks ranks of the workflow are equal to the entry node but these tasks are not on the critical path. Moreover, if there are more than one critical path in a workflow in this case also the CPOP algorithm presented in [Topcuoglu et al., 2002] will fail as it cannot distinguish between the tasks of two critical paths and it will assign them on the single processor marked are critical path processor.

### 8.8. Scheduling and Data Saving Algorithms

The requirements (Requirements **R11** and **R12**) suggest the designing of a grid workflow scheduling algorithm which utilizes a predicted queue wait time of the grid resources and a data saving algorithm which saves the data generated by the tasks of a workflow during the execution of a workflow. In subsections 8.8.1 and 8.8.2 provide the realization of requirement **R11**. The next section 8.8.3 provides a data saving algorithm in order to full
fill the requirement R12, this algorithm saves the data files produced by the tasks of a workflow during the execution of a workflow.

8.8.1. Queue wait time prediction-based grid workflow scheduling algorithm (QWP-GWS)

In this section a queue wait time prediction-based grid workflow scheduling algorithm (QWP-GWS) is presented to fill the requirement (Requirement R11). A workflow when received at a grid workflow scheduler it is placed in the queue of the workflow according to its arrival time. When a workflow reaches at the front of the queue following steps are followed.

1. Ensure there is only one entry/source and one exit/sink node in a workflow. For this purpose if there is more than one entry or exit node the workflow manager component inserts a dummy entry or exit node.
2. Apply the depth first search (DFS) [Cormen et al., 2009] algorithm to find the topological sort of the workflow.
3. Find mean computational and communication time of all the tasks and edges on the available resources.
4. The ranks of the tasks of a workflow are calculated as follows
   a. Use the topological sort of the workflow to find the downward ranks of the tasks.
   b. Use the reverse topological sort to find the upward ranks of the tasks.
   c. Sum the upward and downward rank of each task and assign it to the rank value of the task.
5. The first path includes the entry and exit nodes and consists of nodes of the workflows between the entry and exit nodes. In this research paths are named as critical path one (CP1), critical path two (CP2) and so on. The method of finding paths of a workflow is explained in subsection 8.7.5. The first path is also referred as a critical path.
6. For each critical path a ready tasks list is made and named them as readyTasksList1, readyTasksList2 .... readyTasksListj. Where j is the total number of critical paths in a workflow.
7. Find the queue wait time at each grid resource according to Equation 8-39.
Chapter 8  Grid Workflow Scheduling Based on Queuing Theory

8. After calculating the above parameters the expected completion time of the longest critical path i.e. \( CP_1 \) is calculated. The expected completion time of a critical path is calculated by summing the computation time of the each task on the \( CP_1 \) at a grid resource and the average communication time of each edge on the critical path on a grid resource. The computation time of a task is calculated by dividing the task computation weight by the processing speed of a single processor of the grid resource. The expected completion time of the critical path \( CP_1 \) on the resource \( R_r \) is denoted by \( ECP_1(R_r) \) it can be calculated according to Equation 8-52. Where \( Q \) is the number of tasks in the \( CP_1 \).

\[
ECP_1(R_r) = \left( Q \times E\left[w_{rq}\right]\right) + \frac{\sum w_i}{\text{Speed}(P_i)} + \sum_{k=1}^{M} \left[ \sum (c_{i,j} \times TT_{r,k} + L_{r,k}) \right] \frac{M - 1}{\forall t_i \text{and } e_{i,j} \in CP_1 \text{ and } |CP_1| = Q}
\]

Equation 8-52

9. After calculating the \( ECP_1(R_r) \) value at each grid resource, the resources are arranged in ascending order of the \( ECP_1(R_r) \) values.

10. Then the tasks from the \( CP_1 \) are assigned to the resource number 1, which has a minimum value of \( ECP_1(R_r) \) value. The tasks from the \( CP_2 \) are assigned to the resource number 2, which has second minimum value of \( ECP_1(R_r) \) and so on.

11. Step number 10 is feasible when the number of ready tasks lists is less than or equal to the grid resource. When the number of ready tasks lists is greater than the number of grid resources available, using the quotient remainder theorem the number of ready tasks list assigned to each grid resource are calculated. Moreover the dynamic programming technique is applied so that the tasks belonging to less number critical path should not wait for the greater number critical path’s tasks.

In this section, the queue wait time prediction-based scheduling algorithm is presented in Algorithm 8-2. This algorithm first finds all the paths between a single source and single destination. After finding all critical paths in a workflow named them as \( CP_1, CP_2, CP_3 \ldots CP_j \), Showing that there are \( j \) paths in a workflow where \( j \) is a positive integer. For each critical path a ready task list is created and named as \( \text{readyTasksList}_k \), where the subscript \( k \) shows the critical path number and \( 1 \leq k \leq j \). This \( \text{readyTasksList}_k \) contains all the tasks from critical path \( k \) whose parents have been completed, the data files required to start its execution are saved, and their locations are known. The
ECP₁(Rᵱ) values for all the grid resources are calculated and resources are arranged in ascending order of ECP₁(Rᵱ) value. On this way resource1 can complete the critical path tasks in minimum time, resource2 critical path completion time is more than resource1 but less than resource3 and so on. The two cases elevated while mapping the readyTasksLists to the available grid resources are:

1. Number of non-empty ready lists of a workflow are less than or equal to the number of available grid resources.
2. Number non-empty ready lists are greater than the number of available grid resources.

![Figure 8-9: A matching approach](image)

In case 1 the ready lists are assigned to the grid resources in such a way that the ready list from the CP₁ is assigned to the grid resource having minimum ECP₁(Rᵱ) value, the ready list from the CP₂ (if not empty) is assigned to the resource having second minimum ECP₁(Rᵱ) value and so on. In case 2 where the number of available resources are less and number of non-empty ready tasks lists are greater the famous Quotient-Reminder theorem⁵ and dynamic programming technique is used to solve the problem.

E.g., If the number of non-empty ready tasks lists=10 and number of grid resources=3. It means that only 3 ready lists tasks can be assigned to the resources. The values of

---

⁵. For every pair of integers a,b where b≠0, there exist unique integers q, r such that a=qb+r and 0≤r<|b|: ∀ a,b∈Z,b≠0:∃! q, r∈Z: a=qb+r,0≤r<|b|.
n (quotient) and r (remainder) are calculated, where \( M \) is the number of available resources. Here \( n = 3, r = 1 \) and \( M = 3 \). So each resource will be assigned the 3 ready lists. When the first 3 lists are assigned to the first resource and next 3 to second resource, this scheme will delay some of the critical tasks. So ready tasks lists are assigned to the grid resources in such a way that the smallest and longest are mapped on the same resource. So that any task which appears in a critical path of less number does not wait to complete for a task of greater numbered critical path. This is known as dynamic programing. Assignment of the least numbered critical path ready list and greatest numbered critical path to the resources are shown in Figure 8-9. In this way the ready list 1, 9 and 10 are mapped to the best resource1, the 2, 7, 8 to resource2 and the 3, 5, 6 to resource3. After this run the scheduler checks if there is any remainder than the first reminder list is assigned to the best resource and so on. i.e., if there were 2 remaining lists in above case than the first should be assigned to first best resource and second to second best resources and so on. This process continues until there remains no unscheduled task of a workflow. The number of nonempty readyTaskLists is checked and if it is less than or equal to the number of available resources \( M \), then the tasks from first readylist1 are assigned to resource1, The tasks from readylist2 are assigned to resource2 and so on. IF the number of nonempty readyTasksLists is greater than the number of available resources then using the quotient remainder theorem the number of ‘n’ readyTasksLists assigned to each resource are calculated, \( r \) is the number of remaining unassigned readyTasksLists. Now if the first \( n \) readyTasksLists are assigned to resource1 and next \( n \) readyTasksLists are assigned to resource2 and so on, this will not only overload the resources but also the tasks from the longer critical paths will suffer from long waiting times in resources’ queues. So the dynamic programming technique is used for the readyTasksLists assignment. The readyTasksList1 is assigned to resource1 and n-1 readyTasksLists from last are assigned to resource1, e.g., if \( n=3 \) there are total 3 resources and \( j=10 \) then readyList1, readyList10 and readyList9 are assigned to the grid resource1. The last thing is the remaining readyTasksLists, which are given in this algorithm as \( r \). The first remaining ready list which is represented as subscript \( v \), where \( v=M+1 \), is assigned to resource1 and second remaining readyTasksList which is readyList\( v+1 \) is assigned to resource2 and so on until there is no remaining readyTasksList. E.g., if \( r=3 \) and total available resources are 4 then readyList5 is assigned to resource1, readyList6 is assigned to
resource\textsubscript{2} and readyList\textsubscript{7} is assigned to resource\textsubscript{3}. As shown in algorithm 8-3 that procedure call of \texttt{schedule(readyTasksList\textsubscript{k}, resource\textsubscript{k})} means that all the tasks in the readyTasksList\textsubscript{k} are sent to the resource\textsubscript{k}.

Algorithm 8-2: Queue wait time aware workflow scheduling algorithm

```
1. \textbf{PROCEDURE} Schedule\_workflow(\emph{W}) {
2. \hspace{1em} \textbf{Find} all the critical paths (CP) in a workflow \emph{W} and Name them as \emph{CP\textsubscript{1}}, \emph{CP\textsubscript{2}}, \emph{CP\textsubscript{3}}.....\emph{CP}\textsubscript{j} 
3. \hspace{1em} \texttt{FOR INTEGER k=1; k<=j; k++ DO} 
4. \hspace{2em} \texttt{readyList\textsubscript{k}<- unscheduled ready tasks (ready tasks whose parents had been completed their execution and the data required to start their execution is available) from \emph{CP\textsubscript{k}}} 
5. \hspace{1em} \texttt{IF (j<=M)} 
6. \hspace{2em} \{ 
7. \hspace{3em} \texttt{FOR INTEGER k=1, r=1; k<=j ; k++,r++ DO} 
8. \hspace{4em} \texttt{IF readyTasksList\textsubscript{k} is not empty} 
9. \hspace{5em} \texttt{schedule (readyTasksList\textsubscript{k}, resource\textsubscript{k})} 
10. \hspace{4em} \texttt{ELSE r--} 
11. \hspace{2em} \}WHILE (\exists unscheduled task in any readyTasksList of the workflow) 
12. \}ELSE 
13. \{ 
14. \hspace{1em} \texttt{Do} 
15. \hspace{2em} \{ 
16. \hspace{3em} \texttt{INTEGER n= Number of readyTaskslists assigned to each resource(quotient reminder theorem )} 
17. \hspace{3em} \texttt{INTEGER r= Number of remaining readyTasksLists(quotient reminder theorem)} 
18. \hspace{2em} \} 
19. \hspace{1em} \{ 
20. \hspace{2em} \texttt{For k=1,m=1; k<=j ; k+=n,m++ DO} 
21. \hspace{3em} \{ 
22. \hspace{4em} \texttt{schedule (readyTasksList\textsubscript{k}, resource\textsubscript{m})} 
23. \hspace{5em} \texttt{l++} 
24. \hspace{4em} \}Do 
25. \hspace{3em} \{ 
26. \hspace{4em} \texttt{schedule (readyTasksList\textsubscript{v}, resource\textsubscript{m})} 
27. \hspace{5em} \texttt{l++, s--} 
28. \hspace{4em} \}WHILE (l<n) 
29. \hspace{4em} \texttt{l=0} 
30. \hspace{3em} \texttt{IF ( r!=0)} 
31. \hspace{4em} \{ 
32. \hspace{5em} \texttt{schedule(readyTasksList\textsubscript{v}, resource\textsubscript{m})} 
33. \hspace{6em} \texttt{v++, r--} 
34. \hspace{4em} \} 
35. \hspace{2em} \} //end for 
36. \} (while \exists unscheduled task in any readyTasksList of the workflow)
```
8.8.2. Receiving a completed task of a workflow

A grid resource after executing a task of a workflow sends the results back to the corresponding grid workflow scheduler. The Algorithm 8-4 shows the procedure of receiving a workflow task\( (t_i) \) at a grid workflow scheduler. After receiving a task the grid workflow scheduler follows following steps.

1. The results of task \( (t_i) \) are saved.
2. The data files consumed by the task \( (t_i) \) are deleted.
3. The data files produced by the task \( (t_i) \) are saved according to the data save algorithm presented in algorithm 8-5.
4. The dependency constraints of all the child tasks of the task \( (t_i) \) are removed so that the child tasks may get ready for execution.
5. The prediction accuracy of the task placement decision of task \( (t_i) \) and the prediction accuracy of the queue wait time of the task is calculated.
6. If all the tasks of a workflow have been completed then the result of the workflow is sent to the corresponding user.

Algorithm 8-3: Schedule a ready List

1. PROCEDURE schedule \((\text{readyList}_k, \text{resource}_k)\)
2. {{
   Send all the tasks in the \text{readyList}_k to \text{resource}_k
   }}
3. } //end PROCEDURE
In order to fulfill the requirement R12 a data saving algorithm is designed which works in cooperation with the workflow scheduling algorithm. As discussed in 7.4 the data files of the workflow’s tasks are categorized into 1. Data files produced and 2. Data file consumed. When a task completes its execution and its results are received at the scheduler. The data files produced by the tasks are saved and data files consumed by the task are deleted from the corresponding resources. The list of data files produced by a task \( t_i \) for its children tasks are denoted as \( DFP(t_i) \) and the data files consumed list of task \( t_i \) is represented as \( DFC(t_i) \). The size of \( DFC(t_i) \) when \( t_i \) is entry task is zero and the size of \( DFP(t_i) \) is zero if the task \( t_i \) is an exit node.

Algorithm 8.5 shows the procedure of saving the data file produced by a task \( t_i \). First of all the data file of the child task belonging to same CP as task \( t_i \) is saved on the resource where \( t_i \) had completed its execution. The rest of files are arranged in descending order file sizes and are saved to the grid resources where data movement time is low. The data movement time is lowest i.e., zero for the resource where the task \( t_i \) has completed its execution. Therefore in first attempt the data files are saved on the said resource and if not possible i.e. Algorithm 8.5 returns false, then ID of this resource is added in the not_this_location list. And the files are saved on the resource where data movement time is low.
Algorithm 8-5: Produced files saving algorithm for queue wait time prediction-based scheduler

1. PROCEDURE Save DFP(t_i)

2. |

3. SET resource_ID = get the resource ID where task t_i has completed its execution

4. FILE file = get file from DFP(t_i) for the child of same path as t_i

5. BOOLEAN success = PROCEDURE: Save_file(file, Resource_ID)

6. CREATE LIST not_this_location

7. IF (success == TRUE) 

8. DELETE file from DFP(t_i) list

9. ELSE ADD resource_ID in the LIST not_this_location

10. WHILE success == false DO

11. |

12. INTEGER rsid = id of the resource where the data movement time of file is minimum and rsid is not in the not_this_location list

13. success = PROCEDURE: Save_file(file, rsid)

14. |

15. DELETE all entries of not_this_location and Re-initialize it

16. DELETE file from DFP(t_i) list

17. REARRANGE DFP(t_i) in descending order of file sizes

18. SET resource_ID = get the resource ID where task t_i has completed its execution

19. SET INTEGER s = 0

20. WHILE s < size of DFP(t_i) DO

21. |

22. file = get file from DFP(t_i) from index s

23. success = PROCEDURE: Save_file(file, Resource_ID)

24. IF success = TRUE

25. INCREMENT s

26. ELSE ADD Resource_ID in the LIST not_this_location

27. INTEGER rsid = 0

28. rsid = id of the resource where the data movement time of file is minimum and rsid is not in the not_this_location list

29. success = PROCEDURE: Save_file(file, rsid)

30. IF (success == TRUE)

31. |

32. INCREMENT s

33. DELETE all entries of not_this_location and Re-initialize it

34. |

35. ELSE

36. |

37. add rsid in the LIST not_this_location

38. |

39. GO to step 28

40. |

41. } //end PROCEDURE
### 8.8.4. An example scenario

In order to show the QWP-GWS working an example is presented in this section. The workflow shown in Figure 8-8 has the four critical paths CP₁, CP₂, CP₃ and CP₄. This example workflow is now used to show the working of algorithm 8-2. The workflow is scheduled on the three resources whose specifications can be seen in 8.7.5. The queue wait time of the grid resources is given in Table 8-9. The \( ECP_1(R_r) \) value of the workflow on all of the three resources is shown in Table. 8-10.

<table>
<thead>
<tr>
<th>Resource Number</th>
<th>( E[w_{rq}] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>5</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>10</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource Number</th>
<th>( ECP_1(R_r) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>377</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>427</td>
</tr>
<tr>
<td>( R_3 )</td>
<td>452</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ready Task List</th>
<th>Step 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CP_1 - ReadL = {t_1} )</td>
<td>( T_1 \rightarrow R_1 )</td>
</tr>
<tr>
<td>( CP_2 - ReadL = \emptyset )</td>
<td>( R_2 )</td>
</tr>
<tr>
<td>( CP_3 - ReadL = \emptyset )</td>
<td>( R_3 )</td>
</tr>
<tr>
<td>( CP_4 - ReadL = \emptyset )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>

As in a workflow a task cannot execute until all its parent tasks have completed their execution, so the ready task lists for each critical path in each step are computed as shown in Table. 8-11 the entry task is scheduled to \( R_1 \). The symbol

```
 Algorithm 8-6: Save the file on the resource specified by resource ID

1. PROCEDURE save_file(file, resourceID)
2.   IF (Resource of resourceID capacity allows to save file)
3.     RETURN TRUE
4.     RETURN FALSE
5. } //end PROCEDURE
```
∅ for a ready task list shows that a ready task list is empty and in resource column, it means that no task is assigned to the resource.

So the task $t_1$ is assigned to $R_1$. When $t_1$ finishes its execution the ready list is updated. The task $t_1$ is allocated one processor and it wait in the resource queue for 5 time units, so the finish time of task is 8. As the task $t_1$ finishes its execution and results are received at the scheduler the data files produces by the tasks are saved. The task $t_1$ produced three data files for tasks $t_2$, $t_3$ and $t_4$. The data file for the task $t_3$ is saved on the resource $R_1$. Let this data saving is successful. The data files produced for other tasks if not successful it is saved on the other resources, the locations of the data files produced by $t_1$ are given in Table 8-12.

<table>
<thead>
<tr>
<th>Edge</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{1,3}$</td>
<td>$R_1$</td>
</tr>
<tr>
<td>$e_{1,4}$</td>
<td>$R_2$</td>
</tr>
<tr>
<td>$e_{1,2}$</td>
<td>$R_3$</td>
</tr>
</tbody>
</table>

Table 8-13: Workflow task assignment step 2

<table>
<thead>
<tr>
<th>Ready Task List</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CP_1 - ReadL = {T_3}$</td>
<td>$t_3 \rightarrow R_1$ 17</td>
</tr>
<tr>
<td>$CP_2 - ReadL = {T_4}$</td>
<td>$t_4 \rightarrow R_2$ 22</td>
</tr>
<tr>
<td>$CP_3 - ReadL = \emptyset$</td>
<td></td>
</tr>
<tr>
<td>$CP_4 - ReadL = {T_2}$</td>
<td>$t_2 \rightarrow R_3$ 38</td>
</tr>
</tbody>
</table>

The second step of scheduling is shown in Table 8-13. In all the tasks the finish time of their parent is added as before parent any task of the workflow cannot start. In the next step when the task $t_2$, $t_3$, $t_4$ finishes their execution the ready lists are updated and now only three ready list are there as $CP_2$ has only one task which was executed. The ready lists are updated rapidly just after the completed task is received. In this case if $t_3$ completes before $t_2$ and $t_4$ the readyList$_1$ is updated and vice versa. After the completion task $t_2$, $t_3$ and $t_4$ the locations of the data files produced are given in Table 8-14.
Table 8-14: Locations of data files produced by \( t_2 \), \( t_3 \) and \( t_4 \)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_{3,7} )</td>
<td>( R_1 )</td>
</tr>
<tr>
<td>( e_{4,7} )</td>
<td>( R_2 )</td>
</tr>
<tr>
<td>( e_{2,5} )</td>
<td>( R_3 )</td>
</tr>
<tr>
<td>( e_{2,6} )</td>
<td>( R_1 )</td>
</tr>
</tbody>
</table>

In Table 8-15 the third step of scheduling is shown. As explained before if a task completes before the other tasks assigned to different resources in same step and its child task/ tasks are ready, will start their execution on the free resource.

Table 8-15: Workflow tasks assignment step 3

<table>
<thead>
<tr>
<th>Ready Task List</th>
<th>Step 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CP_1 ) ( \rightarrow ) ( ReadL = { T_7 } )</td>
<td>( t_7 \rightarrow R_1 ) 53</td>
</tr>
<tr>
<td>( CP_2 ) ( \rightarrow ) ( ReadL = { T_6 } )</td>
<td>( t_6 \rightarrow R_2 ) 83</td>
</tr>
<tr>
<td>( CP_3 ) ( \rightarrow ) ( ReadL = { T_5 } )</td>
<td>( t_5 \rightarrow R_3 ) 55</td>
</tr>
</tbody>
</table>

The data locations for the tasks \( t_8 \) and \( t_9 \) are given in Table 8-16 and the step four and five are shown in Table 8-17 and Table 8-19 respectively whereas the data files produces by tasks \( t_8 \) and \( t_9 \) are given in Table 8-18.

Table 8-16: Locations of data files produced by \( t_7 \), \( t_6 \) and \( t_5 \)

<table>
<thead>
<tr>
<th>Edge</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>( e_{7,9} )</td>
<td>( R_1 )</td>
</tr>
<tr>
<td>( e_{6,9} )</td>
<td>( R_1 )</td>
</tr>
<tr>
<td>( e_{6,8} )</td>
<td>( R_2 )</td>
</tr>
<tr>
<td>( e_{5,8} )</td>
<td>( R_3 )</td>
</tr>
</tbody>
</table>

Table 8-17: Workflow tasks assignment step 4

<table>
<thead>
<tr>
<th>Ready Task List</th>
<th>Step 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( CP_1 ) ( \rightarrow ) ( ReadL = { T_9 } )</td>
<td>( t_9 \rightarrow R_1 ) 96</td>
</tr>
<tr>
<td>( CP_2 ) ( \rightarrow ) ( ReadL = { T_8 } )</td>
<td>( t_8 \rightarrow R_2 ) 119</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8-18: Locations of data files produced by $t_8$ and $t_9$

<table>
<thead>
<tr>
<th>Edge</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_{8,10}$</td>
<td>$R_2$</td>
</tr>
<tr>
<td>$e_{9,10}$</td>
<td>$R_1$</td>
</tr>
</tbody>
</table>

Table 8-19: Workflow tasks assignment step 5

<table>
<thead>
<tr>
<th>Ready Task List</th>
<th>Step 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CP_1 - ReadL = {T_{10}}$</td>
<td>$T_{10} \rightarrow R_1$</td>
</tr>
<tr>
<td>$\quad \quad \quad 209$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

Here it can be seen that the schedule length or makespan of the workflow is 209, which is good in the sense that it is even less than the $ECP_1(R_r)$ value of $R_1$, where $R_1$ is considered as the best resource of the grid. This is because of the lower data movement time and consideration of queues’ wait times. Due to file saving algorithm the time spent in communication due to inter task dependencies of the workflow is very low. As it can be seen that in longest path $CP_1$ of Figure 8-7 the inter tasks communication is 0.

Considering the multiple critical paths of the workflow adds this procedure to the $CP$-based scheduling procedures. The main characteristic of this scheduling algorithm is the critical path tasks must be scheduled to the resources where the queue wait time is low as well as the they can complete their execution in less time and the data movement time due to inter task dependency is also low. i.e., the tasks in the less numbered CP have the highest priority and are assigned to the best resource of the grid. Since this procedure depends on the queue wait time of resources so the critical path tasks will wait less, the tasks from the second critical path will wait greater than first critical path tasks but less than third critical path and so on. As compared to HEFT where the only upward ranking scheme is used to find the ranks in this research the ranks of the workflow’s tasks as calculated by summing their upward and downward ranks. This method of critical path calculation also effects on the makespan as in the example if only the upward ranking scheme is used the second critical path begins from $t_2$, it can be seen that assigning this task to the second best resource of the grid will yield in the delaying of longest critical path task $t_7$, which is dependent on task $t_4$. 
Due to less wait time in queues, high priority to the tasks on the critical path and minimum communication times, this procedure completes a workflow execution with low value of makespan. Moreover, scheduling the tasks of workflow to all grid resources ensures that there is no resource in idle state and some tasks are waiting for the faster resources. This will definitely add to resource utilization.

8.9. Experiment Design and result Evaluation

8.9.1. Algorithms to evaluate

In order to evaluate the effectiveness of QWP-GWS algorithm it is compared with Sufferage, Min-Min and Max-Min and HEFT workflow scheduling algorithms. The two main properties of QWP-GWS are it is a list-based and critical path-based scheduling algorithm. Therefore Max-Min, Min-Min and Sufferage are comparable due to list-based scheduling algorithm and HEFT is comparable because of the involvement of priority assignment to the tasks of the workflow on the basis of rank value. The details of HEFT can be found in [Topcuouglu et al., 2002]. In case of Sufferage, the scheduler first calculates the Sufferage value of the tasks in the ready list and secondly schedules the task with highest Sufferage value to the resource where it completes in less time. In case of Min-Min and Max-Min, the highest priority is assigned to the minimum task and maximum task length. Each time the highest priority task is assigned to the grid resource where it completes in less time. In these experiments HEFT, Max-Min, Min-Min and Sufferage are blind of the LUs’ load and other competing workflows. The details of the Min-Min, Max-Min, Sufferage and HEFT can be found in [Yu et al., 2008].

8.9.2. Workload generation

Same variation of the HC-test bench is used in these experiments as that of chapter 7. The only difference is that in these experiments the workflows having tasks in the range of 200-250 are considered.

**Number of competing workflows:** Concurrent execution of more than one workflow is considered in these experiments. The number of competing workflows considered here are, 5, 10, 15, 20 and 25.
**Arrival rate:** The arrival of workflow is simulated using the Poisson distribution, with a mean arrival rate of 5, 10, 15 and 20.

**LUs’ workload:** Along with the workload from grid schedulers grid resources have LUs present on them. These LUs submit the tasks to the grid resources according to DAS2 (fs0-fs4) workload archive [DAS-2, 2012] but with the different wait time. As now, the grid resources have workload from the grid schedulers so the wait time given is workload archive is not taken into account.

### 8.9.3. Resource model

The real grid resources configurations of DAS2 [DAS-2, 2012] are simulated for these experiments. DAS2 is a grid environment composed of five clusters. These clusters are located at five different universities in the Netherlands. DAS stands for the “Distributed ASCI Supercomouter-2” and ASCI stands for “Advance School for Computing and Imaging in the Netherlands”. The cluster located at Vrije Univ. Amsterdam has 144 processors and rest of four have 64 processors. Each cluster has 20GP data saving capacity per processor. The resources’ configuration is shown in Table 8-20.

<table>
<thead>
<tr>
<th>Resources</th>
<th>Number of PES</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>144</td>
<td>Vrije University Amsterdam</td>
</tr>
<tr>
<td>$R_2$</td>
<td>64</td>
<td>Leiden University</td>
</tr>
<tr>
<td>$R_3$</td>
<td>64</td>
<td>University of Amsterdam</td>
</tr>
<tr>
<td>$R_4$</td>
<td>64</td>
<td>Delf University of technology</td>
</tr>
<tr>
<td>$R_5$</td>
<td>64</td>
<td>Utrecht University</td>
</tr>
</tbody>
</table>

### 8.9.4. Performance metrics

QWP-GWS is multi objective workflow scheduling algorithm. It maps the tasks of workflows to the grid resources with an objective to achieve low makespan as well the fair utilization of grid resources. Moreover, this algorithm works along with the data saving algorithm so that the data movement time do not overwhelm the benefit of parallelism. Following performance metrics are considered for comparing the workflow scheduling algorithms.
1. **Makespan**: Makespan is defined as the time difference between the minimum submission time of the tasks of a workflow and the maximum end time of the tasks of a workflow. This time measures the performance of the scheduling algorithm. The results show the mean makespan achieved in each simulated experiment.

2. **Makespan difference ratio**: The makespan difference ratio is calculated by using the Max-Min algorithm as base algorithm and all other algorithms are compared with it. Thus the makespan difference ratio of Max-Min is always 0. The formula is given Equation 7-33, where HEFT passed algorithm is used as a reference but in this case, the Max-Min is used as reference.

3. **Data movement time**: The data movement time in each scenario shows the time spent on moving the data between different resources of the grid.

4. **Data movement time difference ratio**: The data movement time difference ratio is calculated by using the HEFT algorithm as base algorithm and all other algorithms are compared with it. Therefore the data movement time difference ratio of HEFT is always 0. As described in the chapter 7.

5. **Queue wait time**: The queue wait time of an experiment shows the time spent by the tasks of workflows in queues. It is the time difference of a task’s execution start time and its arrival time at a grid resource queue. The tasks waited in the queue for their turn to get the free processor of a grid resource and for the availability of the data files on the resource.

6. **Computation to data movement time ratio**: The computation time in each scenario is calculated and then the ratio of the computation to data movement time is obtained by dividing the computational time by data movement time.

7. **Resource effective utilization**: The resource effective utilization of a workflow having $N$ tasks is denoted as $REU(W)$ and it is given in Equation 7-34. In the QWP-GWS algorithm, it is considered that gain of makespan may not affect the resources effective utilizations. Although the makespan and resource effective utilization are contradicting objectives but the results shows that the proposed
algorithm simultaneously considered both of them and proved its functionality by obtaining good results as compared to other workflow scheduling algorithms like Min-Min, Max-Min etc.

8. **Prediction Accuracy (%)**: The prediction accuracy of the QWP-GWS algorithm is checked for the completion time of tasks and queue wait time. When a task is completed on a grid resource it is checked that whether it has spent more time in the grid resource queue than the predicted queue wait time of the grid resource or less. Also the completion time is checked whether it is more than the expected completion time. The expected completion time is defined as the sum of a task execution time plus the data movement time and the wait time in the queue.

8.9.5. Results

All the experiments were carried on a computer having Intel dual core processor of 2.6 GHz. The GridSim simulator was used for the simulation. Every experiment was executed five times to get the mean values. The mean arrival rate of workflows in an interval and the number of concurrently executing workflows were varied in order to study their impact on the performance metrics introduced in subsection 8.9.4. The QWP-GWS algorithm uses the queue wait time prediction-based on the queuing theory. Since the queuing theory-based predictions are applicable for a system which is in the steady state, a large number of workflows (i.e., approximately 1000) were executed in each simulated experiment in order to ensure the steady state. The results presented next were collected after the system reached the steady state.

Figure 8-10 shows the makespan of the workflows under different mean arrival rate and concurrently executing workflows. The **QWP-GWS** algorithm achieved less makespan as compared to the **Max-Min**, the **Min-Min**, the **Sufferage** and the **HEFT** scheduling algorithms. However, the performance of the **HEFT** scheduling algorithm was better than the **Max-Min**, the **Min-Min**, and the **Sufferage** scheduling algorithms in terms of the makespan. This is because the **HEFT** algorithm uses the averaged values to estimate the resource performance, whereas all the other algorithms except the **QWP-GWS** algorithm are unaware of the LUs’ load at the grid resources and multiple competing workflows. So, these algorithms schedule the workflows’ tasks to the most powerful resources in the grid.
environment. This leads to longer wait times of the workflows’ tasks in the grid resources queue and poor utilization of grid resources as shown in Figure 8-14 and Figure 8-16 respectively.

In order to study the achievement in terms of the makespan, the makespan difference ratios of the **QWP-GWS** algorithm, the **Min-Min**, the **Sufferage**, and the **HEFT** scheduling algorithms were calculated using the **Max-Min** scheduling algorithm as a reference. Figure 8-11 shows the makespan difference ratios. It can be seen that the **QWP-GWS** algorithm achieved more than 85% less makespan as compared to the **Max-Min** scheduling algorithm. Also, due to the almost identical performance of the **Min-Min**, the **Sufferage** and the **Max-Min** scheduling algorithms, this 85% achievement also holds for the comparison of the **QWP-GWS** algorithm with the **Min-Min** and the **Sufferage** scheduling algorithms. However, when the **QWP-GWS** algorithm was compared with the **HEFT** scheduling algorithm, the makespan difference ratio of the two algorithms remained in the range of (5-7) %. The less difference between the makespan of the two algorithms is justified because the resource effective utilization of the **QWP-GWS** algorithm remained higher than that of the **HEFT** scheduling algorithm, as shown in Figure 8-16. Moreover, it can also be seen in Figure 8-12 that the data movement time of the **HEFT** scheduling algorithm was higher as compared to the **Min-Min**, the **Max-Min** and the **Sufferage** scheduling algorithms and it remained significantly higher as compared to the **QWP-GWS** algorithm. The percentage of the data movement time difference ratio is shown in Figure 8-13. It is observed that the data movement time achieved by the **Min-Min**, the **Max-Min** and the **Sufferage** scheduling algorithms was approximately 10% less than the data movement time achieved by the **HEFT** scheduling algorithm. But in case when the mean arrival rate of workflows was 10 per interval and the number of competing workflows were 20, the data movement time achieved by the **HEFT** scheduling algorithms was 1.5%-2% better. This increase in the data movement time achieved by the **Min-Min**, the **Max-Min** and the **Sufferage** scheduling algorithms can be justified because these algorithms save the data files by randomly selecting the grid resources. This random change in graph can also be observed in case of the **HEFT** scheduling algorithm when the mean arrival rate of the workflows per interval was 10 and the number of competing workflows was 25.
Figure 8-14 shows the queue wait time the workflows’ tasks in each experiment. The queue wait time incurred by the QWP-GWS algorithm remained less in all the experiments because the algorithm was based on the queue wait time prediction. The HEFT scheduling algorithm performed better as compared to the Max-Min, the Min-Min and the Sufferage scheduling algorithms in terms of the queue wait time.

The HEFT scheduling algorithm has produced good results in terms of the makespan but this makespan gain not only drastically increased the data movement time but also led to the poor utilization of the grid resources. So, it can be concluded that the HEFT scheduling algorithm outperformed Max-Min, the Min-Min and the Sufferage scheduling algorithms in terms of the makespan and the queue wait time but this performance gain is achieved at the cost of the resource effective utilization and the data movement time as shown in Figure 8-12 and Figure 8-16 respectively.

Figure 8-15 shows the computation time to data movement time ratio of each algorithm. The HEFT scheduling algorithm showed the lowest ratio because this algorithm incurred higher data movement time as compared to other algorithms. This low computation time and high data movement time led to the poor performance of the algorithm because the algorithm scheduled the tasks of the workflows such that they spent less time on the grid resources. The algorithm incurred a significant amount of time on data movement and the grid resources remained idle during this time. The ratio of the computation time to the data movement time of the QWP-GWS algorithm remained lower than that of the Max-Min, the Min-Min and the Sufferage scheduling algorithm and higher than that of the HEFT algorithm. This means that the QWP-GWS spent very small amount of time in data movement and computation time was also not high, so the QWP-GWS successfully balanced between the makespan and resource utilization. The prediction accuracy of the QWP-GWS algorithm was checked for the workflows’ tasks completion times and the queue wait time as shown in Table 8-21.
In conclusion, the results showed the following facts:

1. The queue wait time, the data movement time and the makespan and the resource effective utilization achieved a grid workflow scheduling algorithm by prove its effectiveness.

2. The significant amount of time can be saved by employing the data file saving algorithm proposed in subsection 8.8.3.

3. Better resource effective utilization can be achieved by considering multiple paths in a workflow and assigning each path to a grid resource.

4. Only considering one of the conventional rank calculation methods, i.e. the upward rank and the downward rank is not sufficient for the priority assignment to the tasks of a workflow. This is because a task having a high value of one of the ranks does not necessarily be a critical task as discussed in subsection 8.7.5.

5. The queue wait time predictor can be used to achieve lower makespan of the workflows.
Figure 8-10: Makespan vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.1: Makespan difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.12: Data movement time vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.13: Data Movement time difference ratio (%) vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.14: Queue wait time vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.15: Computation to data movement time ratio vs. total number of concurrent workflows under different mean arrival rate per interval.
Figure 8.16: Resource effective utilization (%) vs. total number of concurrent workflows under different mean arrival rate per interval.
8.10. Summary

This chapter presents the prediction of the queue wait time of grid resources (Requirement R9). The QWP-GWS algorithm is presented, which utilizes the predicted queue wait time (Requirement R11). The QWP-GWS algorithm partitions a workflow in different critical paths. Each critical path consists of a list of tasks. The tasks of the workflow are prioritized according to their critical path number. The tasks belonging to the CP1 are scheduled to the best grid resource, where a grid resource is considered the best if it has the minimum value of the computation time and data movement time for the tasks belonging to CP1. The tasks from the CP2 are scheduled to the second best grid resource, and so on. A data saving algorithm is also presented in this chapter (Requirement R12). Using the data saving algorithm, the data files produces by the workflows’ tasks are saved on the different grid resources so that the data movement time is reduced and it does not overwhelm the benefits of parallelism. The QWP-GWS algorithm is compared with the HEFT, the Min-Min, the Max-Min and the Sufferage scheduling algorithms. The QWP-GWS algorithm outperformed the other algorithms in terms of the makespan, the data movement time, the queue wait time, and the resource effective utilization.
9. Conclusions and Future Work

This chapter consists of three parts. First, section 9.1 summarizes the research work presented in this thesis. Second, section 9.2 presents the original contributions of this thesis and compares them to the state of the art. Third and last, section 9.3 provides a research outlook by pointing to some directions for future research.

9.1. Summary

This thesis tackles the problem of workflow scheduling in the dynamic grid environments. Specifically, the research problems investigated in this thesis include: workflow-based workload modeling (Research Problem RP1; cf. section 1.3), handling the concurrent execution of multiple workflows (Research Problem RP2), predicting the LUs load at the grid resources (Research Problem RP3), handling the data movement time of the tasks of the workflows (Research Problem RP4), predicting the queue wait time suffering at the grid resources (Research Problem RP5), and new workflow scheduling algorithms utilizing the predicted LUs’ load at the grid resources or the queue wait time at the grid resources (Research Problems RP6 and R7).

Chapter 2 analyzed the research problems and identified the set of 12 requirements (Requirements R1-R12; cf. section 2.3) to be addressed in order to solve the research problems. Chapter 3 analyzed contemporary research approaches to workflow-based workload modeling and grid workflow scheduling with respect to the identified requirements, and highlighted their deficits. Chapter 4 presented a survey of the existing grid simulators and the workflow test bench suites. The GridSim grid simulator and the HC-test bench suite were selected to be used in this thesis. Chapter 5 presented the design of a workflow-based workload model for dynamic grid environments, which follows the Standard Workload Format (SWF) (Requirements R1-R4). This chapter also presented the steps to be followed to create a simulated grid environment (Requirement R5) and a workflow-based application model for the GridSim simulator. Chapter 6 described the design of a prediction-based decentralized grid workflow scheduler (Requirements R6). Chapter 7 presented a Local User Load Prediction (LULP) model (Requirements R7 and R8). The LULP was then utilized for designing LULP-based workflow scheduling algorithms (Requirement R10). A data
saving algorithm (Requirement R12) was also presented in chapter 7, which was utilized by LULP-based scheduling algorithms to achieve low data movement time during the execution of the workflows. The proposed LULP-based workflow scheduling algorithms include: the LULP-Min-Min, the LULP-Max-Min and the LULP-Sufferage. The LULP-Min-Min, the LULP-Max-Min and the LULP-Sufferage scheduling algorithms were evaluated against the Heterogeneous Earliest Finish Time First (HEFT) scheduling algorithm. The performance metrics used for the evaluation include: the makespan of the workflows, the time spent on data movement during the execution of the workflows and the resources effective utilization. Chapter 8 presented a prediction model for predicting the queue wait time at a grid resource (Requirement R9). The prediction model was utilized to propose the Queue Wait time Prediction-based Grid Workflow Scheduling algorithm (QWP-GWS) (Requirement R11). A data saving algorithm was also presented in chapter 8 which is utilized by the QWP-GWS to achieve low data movement time (Requirement R12). The QWP-GWS algorithm was evaluated against the Min-Min, the Max-Min, the Sufferage and the HEFT scheduling algorithm. The performance metrics utilized to evaluate the effectiveness of the QWP-GWS algorithm include the makespan, the resource effective utilization, the data movement time, the makespan difference ratio, the computation to data movement time ratio, and the queue wait time.

9.2. Contributions and Comparison with State of Art

This thesis addresses the issues of modeling the workflow-based workload and scheduling the workflows over dynamic grid environments. The following original contributions are delivered by this thesis to tackle these issues:

- **Workflow-based workload model**: Contemporary researchers use randomly generated workflows or workflows from a real application for testing their developed algorithms. However, workflows having a variety of characteristics (e.g., the size of workflows, the number and density of the dependencies among the tasks of a workflow) are required to objectively evaluate the grid workflow scheduling algorithms. [Hönig and Schiffmann, 2004; Hönig, 2007] modeled a workflow test bench suite, but a grid workflow-based workload model was still missing. This thesis for the first time presents a synthetic workflow-based workload model, which is based on the HC-test bench suite [Hönig and Schiffmann, 2004; Hönig, 2007]. The
proposed workflow-based workload model allows evaluation of newly developed workflow scheduling algorithms in the dynamic grid environments.

- **Design of a prediction-based decentralized grid workflow scheduler:** This thesis presents the design and development of a decentralized architecture of a grid workflow scheduler in order to enable the prediction-based grid workflow scheduling. In contrast to the work of [Ranjan et al., 2008; Baghban and Rahmani, 2008], the proposed architecture has the predictor component, which enables the scheduling algorithms to cope with the dynamicity of the grid environment.

- **Local users’ load prediction-based scheduling:** This thesis considers the non-dedicated nature of grid resources while scheduling workflows for the first time. Three scheduling algorithms based on the prediction of the LUs’ load at the grid resources are presented in this thesis. The results show that these algorithms achieved low makespan of workflows. In contrast to [Sakellariou et al., 2007; Yu and Shi, 2007; Yu, 2007], the algorithms presented in this thesis schedule multiple workflows in the presence of LUs’ load at the grid resources. [Wiriyaprasit and Muangsin, 2004] also consider the non-dedicated nature of grid resources but they don’t consider workflows.

- **Queue wait time prediction-based workflow scheduling:** This thesis presents the Queue Wait time Prediction-based Grid Workflow Scheduling algorithm (QWP-GWS). This algorithm schedules multiple critical paths of a workflow to the different resources of the grid. This algorithm considers multiple objectives (i.e., the makespan and the resource effective utilization) for optimization. In comparison with [Afzal et al., 2006], the QWP-GWS algorithm not only allows the multiple workflows execution but it also considers the data movement time between the tasks of workflows. The QWP-GWS uses the predicted queue wait time of the grid resources, whereas [Yu and Shi, 2010] only consider the chances of executing a task by its deadline without showing the actual queue wait time sufferings. Moreover, [Yu and Shi, 2010] minimize the makespan on the cost of excessive data movement time. QWP-GWS considers multiple critical paths and each critical path is assigned to a grid resource. [Topcuouglu et al., 2002] used the Critical Path on a Processor (CPOP) scheduling algorithm where the main critical path is assigned to a grid resource.
However, this algorithm may fail to distinguish between the tasks of a critical path and other tasks of the workflow when the rank values of the tasks which are not on the critical path are equal to the rank value of the tasks on the critical path. Moreover, in CPOP a processor is considered as the critical processor if it has the minimum value of the computation time of the tasks on the critical path. The communication time of the tasks on the critical path is not considered at all. Therefore, this algorithm may not be able to efficiently handle the situation where the data file of a task on the critical path does not get a storage space on the critical processor but the data file is needed to be saved for later use.

- **Data saving algorithms:** This thesis proposes two data saving algorithms. These data saving algorithms work in cooperation with the workflow scheduling algorithms to achieve low data movement time and small makespan.

- **Multiple workflow scheduling:** As the concurrent execution of workflows increases, the data at the scheduler also increases none-linearly (as explained in section 2.1.4). This thesis considers a decentralized architecture of workflow scheduler, which allows the execution of multiple workflows simultaneously without creating huge amount of data at a single scheduler.

- **Multi objective optimization:** The QWP-GWS algorithm presented in this thesis successfully managed to achieve high resource effective utilization and low makespan (which are contradicting objectives).

9.3. Outlook

Although the evaluation shows that the proposed scheduling and data saving algorithms outperformed their competitors, it is believed that further improvements are possible if the following limitations are considered:

- **Assumptions:** Some assumptions of this thesis are not always true in a real grid environment, e.g., the assumption that a task of a workflow is not allowed to execute across the grid resources or the assumption that all the grid resources and grid schedulers' queues have infinite capacity.
• **Failure prediction**: This thesis assumes that the grid resources do not fail during the simulation and all the sites are fully trusted because these types of scheduling strategies are not the focus of this thesis. However, in real grid environment Service Level Agreements (SLA) and failure predictions could possibly provide a failure free execution of grid applications.

• **Utility grid objectives**: The utility grid objectives (grid users’ budget, grid resources providers’ demands) are not addressed in this thesis because this thesis only focuses on community grids.

The preceding limitations indicate the following directions for future work.

• The data movement involves network and storage resources of the grid. The storage space was considered in this thesis while scheduling workflows. However, more research is needed in network-aware scheduling algorithms.

• The workflow-based workload model proposed in this thesis can be reused for studying the effectiveness of other workflow scheduling, load balancing, and fault tolerance algorithms in the dynamic grid environments.

• It is assumed in this thesis that the grid resources do not fail, however resource failure is not an abnormal behavior in real grid environment. Therefore, a future work could be the development of a failure prediction model and its use for studying grid workflow scheduling algorithms.

• The thesis only considered community grids. However, utility grid is also an important type of grid. Redefining the objectives of grid workflow scheduling according to utility grid stakeholders is an interesting future direction of research.


Appendix A  Configuration Formats for the Workflow-based Workload Model

A.1 Format for Specifying the Workflow-based Workload Configurations

```
#workflowBasedWorkload.config
#===================================
#Interval Length (in Seconds)
900
#Arrival Rate (in Workflows/Second)
3
#Number of Workflows Required

#Path of the gridResourcesProcessingCapabilities.config File
...\gridResourcesProcessingCapabilities .config
#Path of the gridResourcesLinksBandwidth.config File
...\gridResourcesLinksBandwidth.config
#Path of the gridResourcesLinksLatency.config File
...\gridResourcesLinksLatency.config
#Path of the Root Directory of the HC Test Bench
...\HC_Test_Bench\n```

Figure A-1: Format for specifying the Workflow-based workload model configurations
A.2  Format for Specifying the Grid Resources' Processing Capabilities

![Table showing grid resources' processing capabilities](image)

Figure A-2: Format for specifying the grid resources’ processing capabilities
A.3 Format for Specifying the Grid Resources Links’ Bandwidth and Latency

![Format for specifying the grid resources' links bandwidths](image1)

![Format for specifying the grid resources' links latencies](image2)