UTILITY DRIVEN GRID SCHEDULING FRAMEWORK

by

ENIS AFGAN

PURUSHOTHAM BANGALORE, COMMITTEE CHAIR
BRANDON EAMES
ELLIOT LEFKOWITZ
ANTHONY SKJELLUM
ALAN SPRAGUE

A DISSERTATION

Submitted to graduate faculty of the University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

BIRMINGHAM, ALABAMA

2009

UTILIY DRIVEN GRID SCHEDULING FRAMEWORK

ENIS AFGAN

COMPUTER AND INFORMATION SCIENCES

ABSTRACT

The grid computing paradigm enables access to geographically and administratively distributed networked resources, and delivers functionality of those resources to individual users. Stemming from the core composition and aggregation of individual resources, the grid is primarily characterized by the heterogeneity it offers. Although such heterogeneity is often considered a feature, it also presents an obstacle in terms of application execution patterns and expectations (in terms of job runtime, resource utilization, and/or user Quality of Service (QoS)). Typical users have little or no knowledge about the concrete requirements their application imposes on such resources and thus have to stumble through a sea of options and uncertainties when submitting a job, leading to inefficient use of available resources.

In order to alleviate the user from having to understand existing dependencies and make low-level decisions, a grid metascheduling framework has been devised that enables automated *application- and user-oriented job metascheduling*. In order to enable *application-oriented metascheduling*, a set of core grid services, Application Information Services (AIS), were designed and developed to provide application metaschedulers with relevant information regarding each application's execution requirements and preferences. With such information, a metascheduler is capable of automatically realizing more job-to-resource mappings. In order to enable *user-oriented metascheduling*, a novel mode of user-scheduler interaction has been devised that builds on top of AIS. The model

is realized in terms of two-way communication between a user and the scheduler enabling strict focus on an individual user and their current job. Overall, this dissertation makes contributions regarding efficiency of use and ease of access for grid resources.

Results of grid job metaschedulers implementing the devised framework are shown as capable of consuming application-specific data in a manner that leverages existing heterogeneity and, in turn, automatically deliver effective application-to-resource mappings. Results achieved are two-fold: (1) behavior of application jobs across grid resources has been significantly improved in terms of job execution control, capable of increasing resource utilization and achieving significant runtime reduction (up to 50%), and (2) each job submission is being tailored specifically to an individual user and their respective job, delivering significantly higher QoS to the user.

Keywords: grid, application-oriented job scheduling, Application Information Services, job execution space, user-scheduler interaction

DEDICATION

Za mamu i tatu

ACKNOWLEDGEMENTS

First of all, a sincere thank you to my mom and dad who have, year after year, supported my endeavors and decisions. You have provided me with unprecedented encouragement and love that I have cherished and carried with me anywhere I would go. Maći, Maći, Maći – I guess I need to thank you too for all the pictures you have uploaded and the nights I kept you awake and on the phone! I am thankful to have a brother like you. To Steve and Lavonne, without whom I would probably not have remained in the US and gotten to where I am now – thank you. Thank you for your love, mindful support and opening your arms to your Croatian son. Finally, there is Ula, thank you for bringing unconditional happiness and inspiration into my daily routine. It is my greatest pleasure to be part of your life.

I would like to express my sincere gratitude to my advisor, Dr. Purushotham Bangalore. Thank you for letting me swim but also thank you for providing a helpful hand when I was on my way down or headed in the wrong direction. Your openness and willingness to help has been not only of tremendous value but it has also been an inspiration to follow as people are starting to turn to me for help. Also, thank you for letting me keep both monitors on my desk.

I would like to thank to my Ph.D. committee for their valuable insight and guiding comments. Specifically, I thank to Dr. Brandon Eames for the approachable attitude and openness. I value and strive to develop such attitude toward professorship. I would like to thank Dr. Elliot Lefkowitz for directing this document into its current format, thus

making it considerably more comprehensible and contained. I would like to express a big thank you to Dr. Anthony Skjellum for the continuous input during my Ph.D. studies. I look up to Dr. Skjellum regarding comprehensible knowledge and ability to immediately understand just about any topic. Dr. Alan Sprague, thank you for the mindful and deliberate discussion regarding the mathematical portions of my work.

I would like to thank Vijay Velusamy and Zhije Guan for your initial patience with showing me how to use various systems or applications and the developed friendship. To this day, I am not sure how would all of this have developed without your help. Leonard Jowers, Vertia Byrd, and Billy Jones – you were the best Level I study group one could wish for. As much pain as it was to learn all that material, I truly enjoyed the time we got to spend together. I furthermore enjoyed and cherished all of the subsequent lunches and associated discussions that we have had. It is a pleasure to be your friend. To Francisco Hernandez, thank you for your willingness to help as well as the times we have spent discussing my work. Thank you for staying in touch and being a friend. I would like to thank John-Paul Robinson for the insightful discussions and, most importantly, the enthusiasm and dedication you have toward computers and science in general. These admirable traits have not only inspired me but have also affected my approach to life.

I would like to thank Igor Nikolić for his friendship as well as his helpful and generous hand. I continuously remind myself of your ‘do-until-perfect’ approach regarding any task undertaken; it is part of that methodology that has led to completion of this dissertation. I would like to thank Bill Pierce and UAB Outdoor Pursuits for getting me outside the lab and for significantly changing my outlook toward the wilderness world and life directions.

Lastly, I would like to thank the Department of Computer and Information Sciences that has provided me with support during this endeavor.

TABLE OF CONTENTS

Page

ABSTRACT iii
DEDICATION v
ACKNOWLEDGEMENTS vi
LIST OF TABLES xiii
LIST OF FIGURES xiv
1. Introduction 1
1.1. Grid and Applications 2
1.1.1. Resource Influence 3
1.1.2. Parameters' Influence 4
1.1.3. Data Influence 6
1.1.4. Grid Access 8
1.2. Problems with Current Grid Access Model 8
1.3. Research Objectives 9
1.3.1. Approach Overview 10
1.3.2. Contributions 14
1.4. Broader Impact 16
1.5. Overview 17
2. Background and Motivation 20
2.1. Glossary of Frequently Used Terms 20
2.2. Grid Computing 23
2.3. Grid Applications 29
2.3.1. Grid Application Classification 33
2.4. Grid Languages and Technologies 35
2.4.1. Resource Specification Language (RSL) 35
2.4.2. Job Submission Description Language (JSDL) 36
2.4.3. Resource Description Language (RDL) 38
2.5. Scheduling Background 41
2.5.1. Local Resource Managers 42
2.5.2. Grid Metaschedulers 43
2.5.3. Types of Metascheduling 46
2.6. Grid Scheduler Approaches and Implementations 47

2.6.1. Community Scheduler Framework (CSF) 48
2.6.2. AppLeS Project 48
2.6.3. GrADS Project 50
2.6.4. Condor-G 52
2.6.5. Nimrod/G 54
2.6.6. Gridbus Broker 55
2.6.7. Michigan Advanced Resource Scheduler (MARS) 56
2.6.8. GridWay 57
2.6.9. Workflows and Multi-objective Scheduling 60
2.6.10. Critique 62
2.7. Application Performance Modeling and Monitoring 65
2.7.1. Using Historical Information to Predict Application Run Times 65
2.7.2. Prophesy Performance Database 67
2.7.3. GridBench 69
2.7.4. GrapBench 71
2.7.5. BioPerf 71
2.7.6. STAPL 72
2.7.7. Application Skeletons 74
2.7.8. Critique 75
2.8. Embarrassingly Parallel (EP) Application Domain 78
2.8.1. Selection of the EP Application Domain 79
2.8.2. EP Applications 80
2.8.3. EP Application Metascheduling Considerations 81
2.9. Other Related Work 88
2.9.1. Bioinformatics Application Domain 88
2.9.2. Statistical Genomics Domain 91
2.9.3. Simulating Grid Resources 93
2.9.4. Automating Service Descriptions 95
2.9.5. Cloud Computing 96
3. Approach 98
3.1. Complexity of Grid Job Metascheduling 98
3.2. Rationale 99
3.3. Approach Overview 101
3.4. Requirements for Described Approach 109
3.5. Application-oriented Metascheduling 113
3.5.1. Application Specification Language (ASL) 117
3.5.2. Historical Application Performance Database (AppDB) 122
3.5.3. GridAtlas 126
3.5.4. Security Considerations in AIS 129
3.5.5. Composition of Services 130

3.5.6. AIS Implementation 131
3.5.7. AIS Usage Scenario 131
3.6. User-oriented Metascheduling 137
3.6.1. Interacting with a User 137
3.7. Realizing Described Approach 140
4. Performance Analysis and Modeling 142
4.1. EP Application Metascheduling 142
4.1.1. EP Application Taxonomy 143
4.1.2. EP Application Execution Model 146
4.1.3. EP Application Scheduling Framework 151
4.2. Performance Analysis 154
4.2.1. Task Input Data Influence 154
4.2.2. Task Execution Resource Influence 157
4.2.3. Task Parameters Influence 159
4.2.4. Job Parameterization 162
4.2.5. Performance Analysis Observations 165
4.3. Metascheduling Models 166
4.3.1. Homogeneous Resources Model 167
4.3.2. Heterogeneous Resources Model 169
4.4. Reflections on the Approach 172
5. Realizing Application- and User-oriented Metascheduling 175
5.1. Bioinformatics Application 175
5.1.1. Dynamic BLAST Architecture 178
5.1.2. Dynamic BLAST Performance Results 182
5.2. Statistical Genetics Domain 188
5.3. Realizing User-oriented Metascheduling 193
5.3.1. OptionView Architecture 194
5.3.2. The Controller 196
5.3.3. Metascheduling Algorithm 200
5.3.4. User Interaction Module 202
5.3.5. Experimental Validation of OptionView through Simulation 205
5.3.6. Experimental Validation of OptionView on Real-World Resources 209
6. Summary and Conclusions 213
6.1. Selected Highlights 213
6.1.1. Contributions 213
6.1.2. Validation 216
6.1.3. Future Directions 217
6.2. Vision 217
7. Future Work 219

7.1. Extensions 219
7.1.1. Automating the Process 219
7.1.2. Metascheduler Pool 220
7.1.3. Workflow Applications 221
7.1.4. Extensions Beyond EP Applications 222
7.2. Moving into the Clouds 223
7.3. Metascheduling-as-a-Service (MaaS) 227
LIST OF REFERENCES 228
GRID USER CATEGORY CLASSIFICATION 246
APPLICATION SPECIFICATION LANGUAGE 251
TECHNICAL RESOURCE DETAILS 272

LIST OF TABLES

Table *page*

1 Steps required to run a job on the grid. 8

2 Availability of resources used during experimentation. 154

3 Availability of resources used during experiments with Dynamic BLAST and AIS. 184

4 Technical details of resources used during experimentation with R code. 189

5 Resource details used during experiments. PS refers to Processing Slot or a node. PE refers to a Processing Element or a core. MIPS stands for Millions Instructions per Second of a single PE and is a resource performance metric employed by GridSim toolkit. PE MIPS were derived based from normalized application-specific performance benchmarks for given resource. 207

6 Statistics of differences between generated and simulated job execution options. Numbers indicate difference in respective units and the corresponding percentage of simulated results when compared to the estimated values. 209

7 Statistical analysis of runtime accuracy across all executed job execution options 211

LIST OF FIGURES

Figure *page*

1 Runtime of sample BLAST searches on specified architectures using 1.1 GB nr database and a 10 query input file randomly generated 4

2 Comparison of execution times of mpiBLAST and qsBLAST algorithms on different architectures (Olympus=3.2 GHz Intel Xeon with 4GB RAM, Everest=1.6 GHz AMD Opteron with 2 GB RAM) using 1.1 GB nr database and a 10,000 input query file on specified number of CPUs. qsBLAST refers to query splitting BLAST that enabled streamlined distribution of input data across resource nodes. 6

3 Difference in job runtime, load balance and cost between a naïve job submission and an optimized job submission of a BLAST job. Jobs performed the search against the 1.6 GB nr database using 4,096 query input file. Technical resource details are available in Appendix C. 7

4 A new classification of possible topics and paths into which general metascheduling can be decomposed. Topics and paths explored in this dissertation are highlighted. 11

5 A high-level architecture and control flow of the devised grid metascheduling framework utilizing application-specific metascheduling and delivering two-way interaction between a user and the metascheduler Highlighted boxes indicate contribution of this work 13

6 Visual chapter layout and content summary 19

7 Layered structure of grid computing 29

8 Sample RSL document for an MPI type job 36

9 Sample JSDL document describing a BLAST job 37

10 Relationship between end users, WSRF, RSL and JSDL at the basic level showing user's need to directly interact with the low-level infrastructure components (i.e., RSL & JSDL). 41

11 Interactions and relationship between an end users, a grid metascheduler and grid resources 44

12 a) Balanced structure workflow, and b) unbalanced structure workflow 60

13 Execution model for embarrassingly parallel applications 80

14 Runtimes of tasks distributed across three resources when using a job plan (left most three bars - colored in red) and when using first-come-first-serve approach with 120 tasks (remainder of bars). Overall, job planning leads to 30% shorter turnaround time. Jobs executed 1,024 query BLAST search against the 1.6 GB nr database. 83

15 Significance of task-level parameterization - improperly parameterized task can result in a resource behaving poorly. Parameter set 1 used basic invocation of BLAST where only a single thread was instantiated to process the input. Parameter set 2 matched the number of threads to the number of cores available on the resource. 84

16 Variation in runtimes of a set of BLAST tasks caused by load imbalance between the tasks. Tasks performed a search against the 1.6 GB nr database using 4,096 query input file. Load imbalance was caused by disproportionate workload assignment to individual nodes in terms of query lengths. 85

17 Difference in job runtime, load balance and cost between a naïve job submission and an optimized job submission of a BLAST job. Jobs performed the search against the 1.6 GB nr database using 4,096 query input file. 87

18 Two models for parallelizing BLAST: (a) query splitting and (b) database splitting 90

19 Computational model for statistical SSG analysis code indicating viable levels of parallelism for application execution. 92

20 Value for completed job from user's perspective and associated change through a typical day 101

21 A representation of job execution space allowing the user to choose among available job execution options after having considered associated tradeoffs. Each dot represents a job execution option (i.e., fully parameterized job plan) mapped onto the two objectives. 107

22 Architecture (showing individual components and interactions between those) for a metascheduler supporting contributions devised as part of this dissertation. 108

23 A high-level architecture of Application Information Services (AIS) deployed at the level of a VO with only major communication links shown. 116

24 Fourth pillar of grid computing..... 117

25 High-level architecture of AppDB..... 125

26 Snapshot of AppDB web interface showing a list of jobs and task details for one job..... 126

27 Architecture and interaction modes of GridAtlas 129

28 An event diagram for registration or update of data incorporating data propagation from GAD to GAA..... 133

29 Sample job specification provided by a user at the time of job submission to AIS-integrated metascheduler..... 134

30 An event diagram for GridAtlas wrapper on top of GridWay metascheduler. User job submission is streamlined by extracting application- and resource-specific information automatically on user's behalf from GridAtlas service. 135

31 Integration of AIS into the application execution control flow. 136

32 A general model of support mechanisms for delivering application- and user-oriented metascheduling solutions that can range in level and type of user support. 141

33 Heterogeneity of task runtimes for a job that is executed across heterogeneous resources. Tasks assigned to individual resources exhibit comparable runtimes but runtimes of tasks assigned to different resources vary significantly indicating the impact a resource can have on task's (and in turn, job's) runtime. 148

34 Illustration of the job parameterization process aiming at minimizing load imbalance..... 149

35 Devised two-step EP application metascheduling framework..... 152

36 Impact of number of queries used as input for a BLAST task on task's runtime..... 155

37 Physical characteristics of two input files used to test impact of query length on BLAST application runtime: (a) one file has a large number of short queries while the other has a small number of long queries and (b) runtime of tasks parameterized with corresponding input files..... 156

38 Comparison of BLAST execution times across resources. Architectural details of machines used are provided in Appendix C

with resource availability listed in Table 2. Presented experiments searched 32 queries randomly selected from the VBRC database against the nr database (1.6GB in size). 159

39 Effect on BLAST task runtime characteristics when varying number of threads option across resources. Application scales efficiently to the point where number of threads matches the number of processing cores on a given resource. 160

40 Runtimes of two parameterizations of the same BLAST task on one node. Tasks used 128 queries as input and searched against the 1.6 GB nr database (note that the y-axis in the figure does not start at zero). 161

41 Runtimes of a BLAST job using 1,024-query input file against the nr database (1.6GB) when the workload is divided into specified number of tasks across multiple nodes. The same job was executed on three resources to analyze variation in performance across resource architectures. Each task initiated two execution threads. 163

42 BLAST job execution efficiency across two resources differing in their architectures. Efficiency of Everest (AMD based) resource dips faster but remains constant longer while the Coosa (Intel based) resource shows continuous decrease in job efficiency. 165

43 High-level diagram of interactions between grid components and Dynamic BLAST 177

44 Internal dataflow for Dynamic BLAST 180

45 Runtime characteristics of a set of BLAST jobs ranging from plain query splitting BLAST parallelization to Dynamic BLAST. Different data distributions indicate the restructuring of the data assigned to individual resources and assignment of data amount to individual resources that is proportional to resources' capability. Experiments were performed against the 1.6 GB nr database using 4,096 query input file. 185

46 Difference in query distribution between (a) simple query splitting model and (b) BLAST-specific data distribution model. Dotted boxes indicate the amount of data assigned to individual resources while solid boxes indicate data portions assigned to individual nodes (i.e., tasks) on any one resource. Consistent distribution of queries results in consistent node and resource performance reducing load imbalance. 187

47 Different data distributions across resources based on (1) resource size only, and (2) resource performance. 188

48 Runtime characteristics of a set of R jobs highlighting importance and effects of applying derived data distributions and utilizing publicized resource allocation policies. More specifically, jobs executed a single parameter set for 10,000 iterations and multiple processes were started on each node to correspond to the total number of processing cores available per resource node..... 190

49 Two data distributions across resources based on (1) resource size only, and (2) application-specific resource performance. Values indicate number of iterations to be performed against the parameter set under analysis. 191

50 (a) Initial irregularity among runtimes of individual tasks within one resource (i.e., Olympus), and (b) restricted irregularity leading to more controlled load balancing. Control was achieved through heterogeneous assignment of iterations to individual nodes..... 192

51 High-level OptionView architecture with numbers indicating general progress flow. Following user initiated job submission, the scheduling algorithm is repeatedly invoked to generate the job execution space. After presenting the job execution space to the user, the user selects desired job execution option. 195

52 A sample two job execution options. Based on maximum resource availability, different configurations of resource availability are artificially constrained by the Controller and used to invoke the Scheduler, which automatically generates job execution option. Repeated invocations of the Scheduler lead to generation of job execution space. 197

53 A snapshot of the OptionView GUI module presenting a job execution space to the user 205

54 Job option execution space as (a) generated by OptionView, and (b) simulated through GridSim. Each individual point shown represents a single job execution option, namely all the details required to submit a job in an application-oriented fashion (e.g., resource(s) selected for execution, data distribution under resource capability constraints, and individual task parameterizations). 208

55 Experimental runtime data for real-world resources for selected job execution options. Circles represent job execution option runtime estimation generated by OptionView while x's represent observed runtime characteristics after job's execution on real-world resources as per instructions of the job plan generated by OptionView. 211

56 A sample of task-level workflow optimization..... 221

1. INTRODUCTION

One of the key purposes of computer science, among many goals and challenges (*e.g.*, [1]), is the facilitation of the scientific process. From the idea and experiment stages of a project down to data analysis and observations, computer science seeks to enable, enhance, and speed up this never-ending process. In order to achieve set goals, continuous research and improvements to the existing technologies must be made to keep up with the demand imposed by the scientific community. Improvements range from changes and updates with individual hardware components all the way to computer interfaces and overall capabilities presented to the users. One of these technologies fostering development is grid computing. Grid computing [2] can be seen as a culmination of distributed computing [3] and high-performance computing [4]; it integrates networking, communication, computation and information to provide a virtual platform for unlimited computing power and data management [5]. Overall, grid computing can be defined as a hardware and software infrastructure composed of multiple resources that do not belong to single administrative domain, use standard and general purpose protocols and interfaces for communication, and deliver non-trivial Quality of Service (QoS) [6]. Virtual Organizations (VO) [7] established atop a grid computing paradigm represent aggregations of communities that may span national and international boundaries but share common objectives.

1.1. Grid and Applications

The size of individual grids can vary greatly; a grid may include only a few machines in a department or it may include multiple machines spanning several organizations from around the world (*e.g.*, [8, 9]). Reasons for establishing a grid infrastructure can also differ greatly. An organization may decide to deploy a grid infrastructure in order to unify and simplify management of available resources, even if alternative technologies such as web services, parallel programming methods, RMI (Remote Method Invocation) enable similar overall functionality (*e.g.*, [4, 10]). Alternatively, deploying a grid infrastructure may bridge gaps between available technology to realize otherwise unattainable goals, such as access and management of otherwise independently administered and geographically distributed resources [11]. Obviously, there are multiple degrees of grid adoption in between these two extremes.

In order for a technology to become accepted there must be a viable use for it; irrespective of the grid size, applications that make use of available resources are the key component contributing to the acceptance of the grid [12]. Because the grid provides a novel paradigm for application execution and resource sharing, adjustments to applications are likely to be required. Such adjustments may include modifications to the application source code, creation of custom application wrappers that adopt execution patterns of an existing application to the grid paradigm, or use of the initially available application in a manner that is compatible with the grid paradigm [13].

Regardless of the method employed, applications impose requirements on the underlying resources [14]. Meeting those requirements as closely as possible can be seen as the ultimate goal of grid computing. The grid, with a wide range of available

resources, offers the optimal platform for meeting such requirements. Nevertheless, recognizing and meeting such requirements is a non-trivial task.

1.1.1. *Resource Influence*

Accompanied with the growth in infrastructure size there is an increasing number of choices regarding which of the available resources to choose for application job execution. This is further complicated with the heterogeneity such resources offer [14]. For example, consider runtimes of a set of executions of the Basic Local Alignment Search Tool (BLAST) application [15] across several resources found on UABgrid and SURAgrid [8] (see Figure 1). BLAST is a commonly used bioinformatics sequence analysis tool that performs similarity searches between a short query sequence and a database of infrequently changing information such as DNA and amino acid sequences [15]. Because of the inherent heterogeneity of grid resources (in terms of hardware and available software characteristics), the application's performance differs because it is executed on various resources or different invocation parameters are used [14, 16, 17].

Figure 1 depicts a small sample of a performance variance for a set of BLAST jobs on varying hardware architectures. From the perspective of a typical user or a domain scientist, all these resources might appear equivalent, inherent differences would not be recognized, and the selection of which resource they run the job on might be random or based on previous experiences. Once a routine has been established, even though the input data, algorithms, or even the applications change, the user may always choose the same resource [18]. As can be seen in Figure 1, differences in the execution times of the application jobs can vary greatly across available machines. For instance, comparing the 3.2 GHz Intel Xeon versus the E450 Sun Sparc, the runtime of the entire job varies as

much as 1,200%. Yet, the given example depicts the variance in execution times of two single CPU machines executing the same algorithm without inter-task communication.

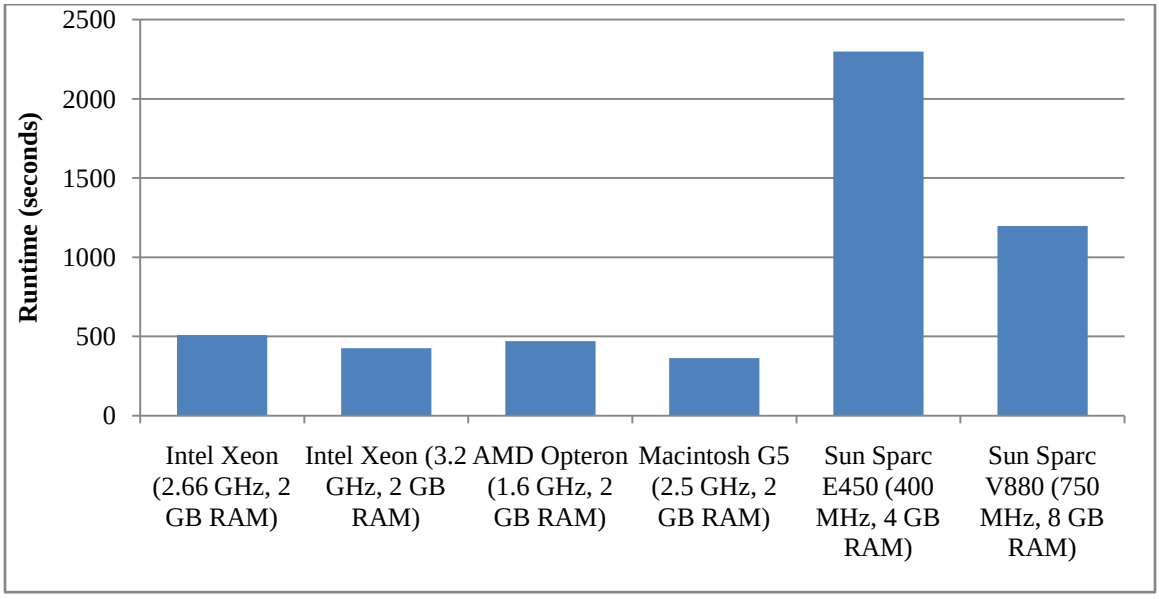


Figure 1. Runtime of sample BLAST searches on specified architectures using 1.1 GB nr database and a 10 query input file randomly generated

1.1.2. *Parameters' Influence*

Beyond executing a job on a single workstation with a single processing core, if multiple CPU machine or a cluster is considered, the choices for invoking an application grow significantly. Initially, one is faced with the same problem as earlier of choosing the appropriate architecture; but now there is the choice of additional application invocation parameters such as the number of CPUs to request or minimum amount of available main memory. Furthermore, continuing with the BLAST example, there are numerous versions of the algorithm performing sequence alignment (*e.g.*, FASTA [19], SSEARCH [20], HMMer [21]), including tightly parallel implementations (*e.g.*, mpiBLAST [22]). This is important to note because different resources may have different versions of the same application installed. Furthermore, individual resources may have different versions of

the same application available, and it has been shown that those applications often have differing runtime characteristics [17].

Figure 2 presents an example of executing only two different algorithms across two resources of different architectures. In Figure 2, query splitting BLAST (qsBLAST) refers to the standard implementation of BLAST available from the National Center for Biotechnology Information (NCBI)[1] with the input file evenly divided across available resources. qsBLAST represents a locally developed Perl script that automates the process of input file splitting and task submission. Optimized version of qsBLAST refers to the same software version but where the input data was distributed across the nodes in a uniform fashion regarding query lengths (Section 5.1.2 provides a detailed example). From this figure, it is apparent that runtime differences between the two BLAST algorithms are significant. From the data on optimized jobs, it can be concluded that the characteristics of input parameters fed to the job can also have significant impact on job runtime.

Even though provided data presents a significant advantage of qsBLAST over mpiBLAST, such results do not necessarily represent the constant behavior of the two algorithms because mpiBLAST was designed to allow for efficient searching of databases that do not fit into main memory of a computer [23]. Because the database size for shown examples was smaller than the amount of memory available on individual nodes, the inter-task communication used by mpiBLAST caused unnecessary runtime overhead resulting in slower job turnaround time. Overall, depending on the characteristics of a machine and requirements of a job, one algorithm may be preferable over another, resulting in an additional variable to include during grid job submission.

[1] http://www.ncbi.nlm.nih.gov/

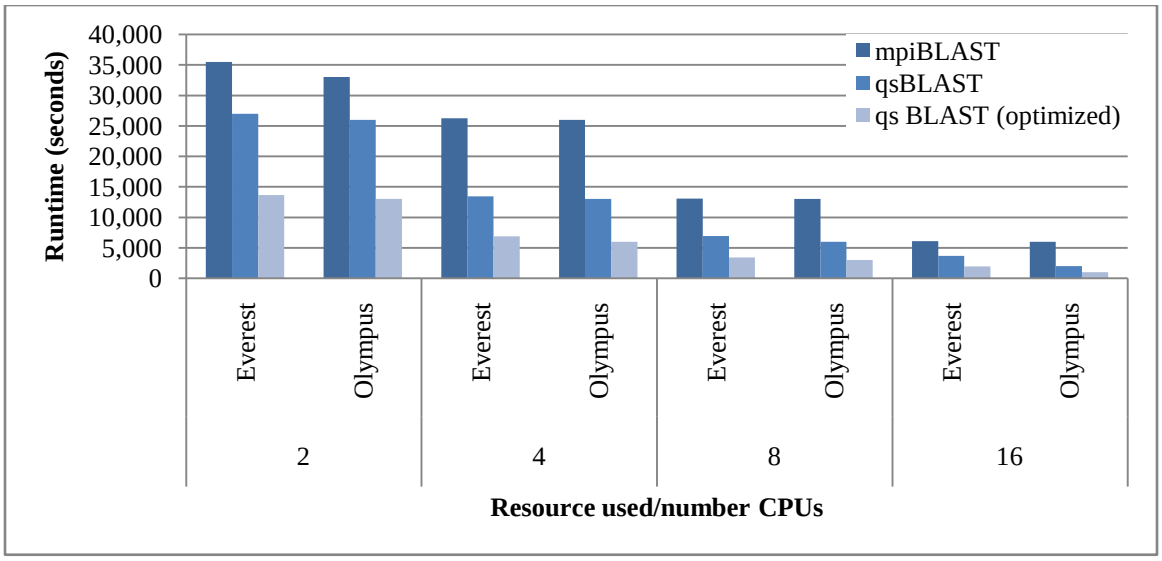


Figure 2. Comparison of execution times of mpiBLAST and qsBLAST algorithms on different architectures (Olympus=3.2 GHz Intel Xeon with 4GB RAM, Everest=1.6 GHz AMD Opteron with 2 GB RAM) using 1.1 GB nr database and a 10,000 input query file on specified number of CPUs. qsBLAST refers to query splitting BLAST that enabled streamlined distribution of input data across resource nodes.

1.1.3. *Data Influence*

As the complexity of user environment grows toward a large scale grid, the user is faced with choices regarding all the parameters that affect job performance characteristics, namely individual resource selection, available application installation versions, and job input parameters. Furthermore, grid enables user's jobs to be divided into multiple tasks that can be executed across multiple resources simultaneously. Within such execution model, there is a benefit in coordinating execution of individual tasks to minimize load imbalance across the resources [24]. Coordinating the load imbalance in such an environment is dependent on all the mentioned factors but also the size of input data assigned to individual tasks. In such a case, there is a need to coordinate submission of tasks between faster or larger resources to process larger data segments while weaker or smaller resources process smaller data segments. The goal is that by the time the job finishes, it did so in a shorter amount of time than it would have on any single resource

(or it realized a higher value for an alternative utility such as accuracy or cost), but also no individual resource caused excessive load imbalance.

An example of the effects input data distribution may have on overall BLAST job runtime when a job is executed across multiple resources is shown in Figure 3. The three cases shown represent the outcome of manipulating the internal characteristics of the input data files and the distribution size of data chunks that were passed to individual resources (for more details on techniques used see [14]). As can be seen from the figure, the runtime reduction is on the order of 50%.

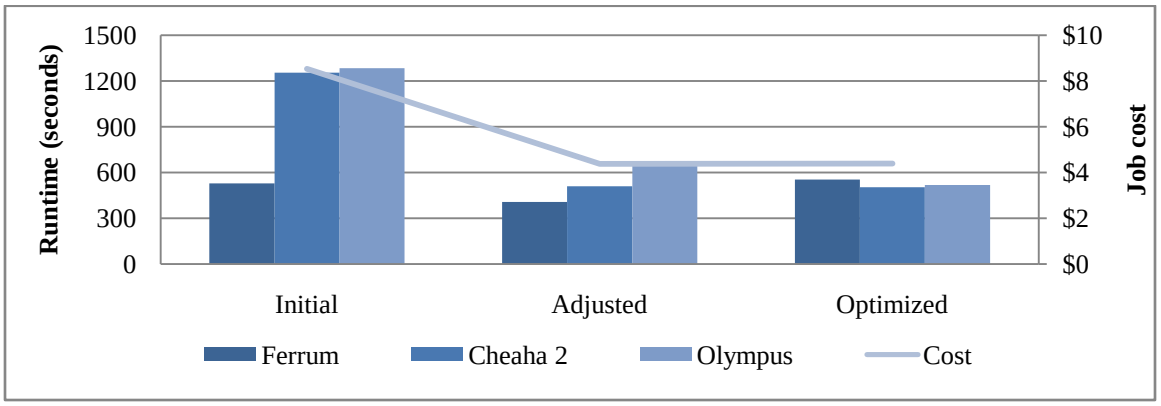


Figure 3. Difference in job runtime, load balance and cost between a naïve job submission and an optimized job submission of a BLAST job. Jobs performed the search against the 1.6 GB nr database using 4,096 query input file. Technical resource details are available in Appendix C.

In an economic model where users pay for utilized resources, it is not only beneficial but necessary to allocate jobs to resources in an efficient manner to prevent excessive costs. Figure 3 also depicts the effects of an excessive job runtime on the job cost caused by a load imbalance across individual resources. Dependant on the employed cost polices of the individual resources, the job cost can be reduced proportionally to the job runtime. Figure 3 also points out that careful manipulation of job parameters has a positive effect on user's experience; namely, runtime for the "Adjusted" case is approximately 10%

longer than the runtime for the "Optimized" case. However, the cost remains constant across both cases allowing the user of the "Optimized" case to enjoy a higher utility (in terms of runtime when compared against cost).

1.1.4. *Grid Access*

Typical access to grid resources is realized through direct use of grid middleware requiring the user to manually perform all the steps required for job submission and, at the same time, incorporate all of the factors influencing and affecting the execution characteristics of user's job. This may require a significant amount of knowledge, as can be seen in Table 1 where a simple use case scenario involving job submission of a grid application is presented:

Table 1. Steps required to run a job on the grid.

1. Find the desired application in the grid
2. Obtain requirements/instructions on how to invoke the application
3. Select a resource to use for application execution and provide appropriate job parameters, by matching application requirements/capabilities (*e.g.*, CPU speed, memory, storage requirements) against the list of available resources (*e.g.*, obtained from Grid Information Services (GIS) [25])
4. Transfer all the required files to the remote resource, this may include executable as well as any input and/or parameter files
5. Invoke the application (through a command line interface, API, or a portal)
6. Wait for the application to complete, possibly monitor application status
7. Obtain any output files by copying result files back to the user machine using grid file copy tools

1.2. Problems with Current Grid Access Model

Each of the steps in Table 1 requires a high-level of expertise, which can be seen as a major obstacle in attracting grid users. Users have established protocols and routines they are familiar with, which usually involves using their home institution's resources. They are less likely to venture into new areas such as grid computing to adopt an entirely new

model for their daily business - unless the obstacles are realized, worked on, and finally overcome.

When a user wants to submit a job to the grid and achieve a desired level of utility, as illustrated by above scenarios, the user first needs to learn the submission protocol (as outlined in Table 1). Next, the user needs to understand and coordinate characteristics and dependencies between an application, available resources, resource configurations, and input data. Understanding this process enables them to achieve load balance across started tasks and realize desirable job performances, like job runtime, cost, accuracy or a combination of those. Such factors, combined with the dynamic state and number of resources in the grid, are all seen as barriers and should be handled with new mechanisms and technologies. Alleviating the need for individual users to operate with the grid at this level while realizing user's goals would lead toward a broader acceptance of the infrastructure. By realizing user's goals not only would available resources be used more effectively, but users could focus on their work as opposed to dealing with the infrastructure operating details. In general, the grid can be seen as an ecosystem comprised of several user categories (discussed in Appendix A), applications, and the underlying resources; in order for this ecosystem to prosper, balance between those needs to be established and finely tuned.

1.3. Research Objectives

Several projects developed throughout the world (presented and discussed in Chapter 2) have recognized the aforementioned scenario as a problem and attempted to solve it from different angles. The major aspect unifying all of the discussed projects, including the major topic of this dissertation, deals with abstracting users from the complexities

introduced by grid computing and involved in a typical job submission. As long as grid resources are viewed as individual machines, instruments, algorithms, software packages, or data storage locations, the final goal of the grid has not been reached. Mapping between resources and users needs to be hidden where resource discovery, requirements, acquisition, and reliability are part of a larger package, which a user can easily access when participating in the grid ecosystem.

The aim of this work is to deliver a unified view of the grid to a user in terms that are relevant to them directly, while hiding all the low-level details not directly pertinent to the user. This is in contrast to the popular notion of requiring the user to adopt and learn how a technology operates and what it means. Results of this work can thus be seen as an increase in the level of abstraction required for the user to interact with the grid in such as way that the low-level operating details are not only automatically handled but are also customized for an individual user.

1.3.1. *Approach Overview*

Work presented in this document focuses on metascheduling of individual user jobs across grid resources accompanied with effective interaction between the metascheduler and the user. These goals are achieved by understanding and coordinating the parameters influencing execution characteristics of the job and by automating the application metascheduling process (*i.e.*, developing notion of application-oriented metascheduling). Benefits of application-oriented metascheduling are then manipulated to enable focus on an individual user and an individual job enabling notion of user-oriented metascheduling.

Figure 4 captures a classification of the field of metascheduling resulting from study and work done as part of this dissertation (for complete discussion of this figure, see

Section 2.5.3). Topics and paths traversed within this dissertation are highlighted in dark and are connected with solid arrows. Within the application-oriented domain, the focus of this dissertation is on single objective minimization problem with job runtime as the main objective (Section 5.1 and Section 5.2). Within the user-oriented metascheduling domain, the tradeoff problem with cost and runtime as main objectives is explored (Section 5.3). These objectives were selected based on their potential merit to the wider community and because of the general interest across the field of metascheduling. Some of the paths not traversed have been explored previously (as described throughout Chapter 2), while some paths have been left for future work including extensions to work presented here.

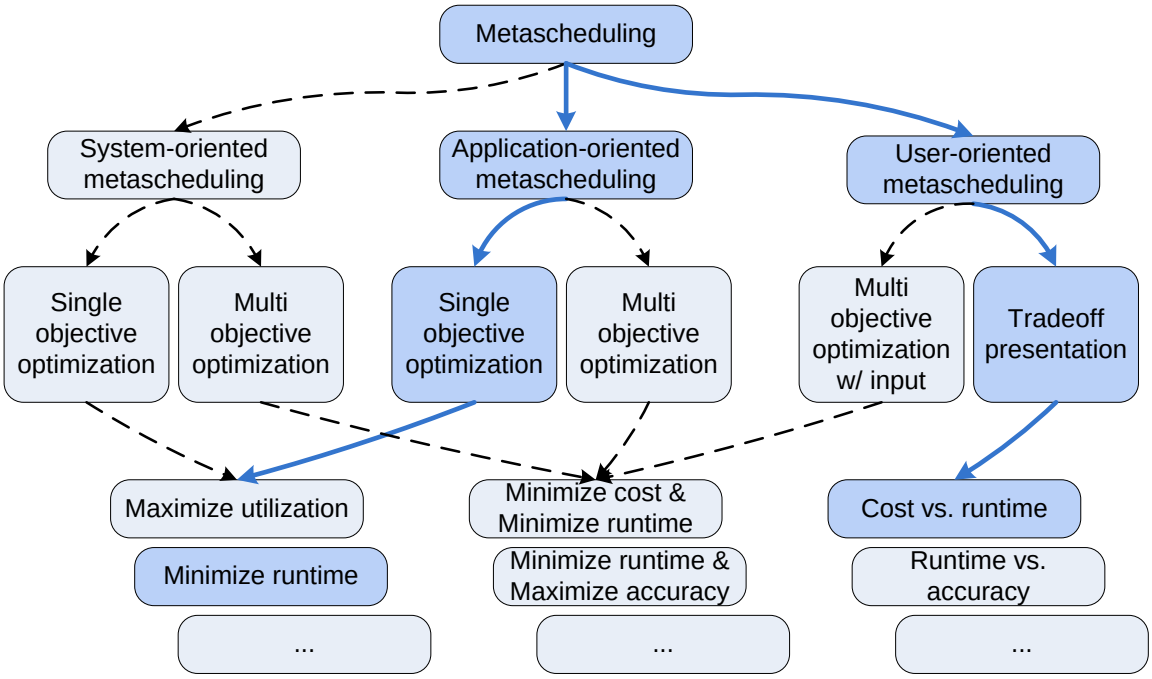


Figure 4. A new classification of possible topics and paths into which general metascheduling can be decomposed. Topics and paths explored in this dissertation are highlighted.

In order to realize application-specific and effective metascheduling of user jobs, a framework for collecting application- and resource-specific information has been devised. This framework enables collection of relevant application and resource

information, which enables its interpretation and use by the metascheduler. The metascheduler is then capable of combining resource availability with application suitability for those resources to generate application- and resource-specific job execution plan that enables customization of the job submission process by the user. This leads to achieving required user utility, which is defined as a measure of value delivered to the user as a function of job execution time. Finally, the overall approach aims at redefining the interaction mode of the scheduler and a user by focusing on an individual user and introducing two-way communication between a user and the scheduler.

During a typical job submission, a user submits a job request to a scheduler upon which the scheduler acts. In the presented work, the focus is instead put on an individual user and their individual job. With such a narrow focus, and combined with the application-specific metascheduling, it is possible to automatically derive and analyze characteristics of multiple job plans (*i.e.,* job execution alternatives) for each job submission and deliver those to the user prior to job's submission. The analysis of possible alternatives, as composed by the metascheduler, then enables mapping of those alternative job executions onto conflicting objectives (*e.g.,* cost vs. runtime, accuracy vs. runtime) and delivering them to the user (*i.e.,* job execution space). Subsequently, the user can easily and effectively consider possible tradeoffs regarding their job submission. Upon the user making the decision on which alternative to execute, user's job is passed to a job submission manager for execution. The scenario and the overall approach just described are depicted in Figure 5 where the major contributions of this dissertation are colored in orange. Metascheduler component (pattern colored) indicates that although the concept of metascheduler is not a key contribution of this work, a specific metascheduler

instance has been devised and developed that enables realization of the other two contributions.

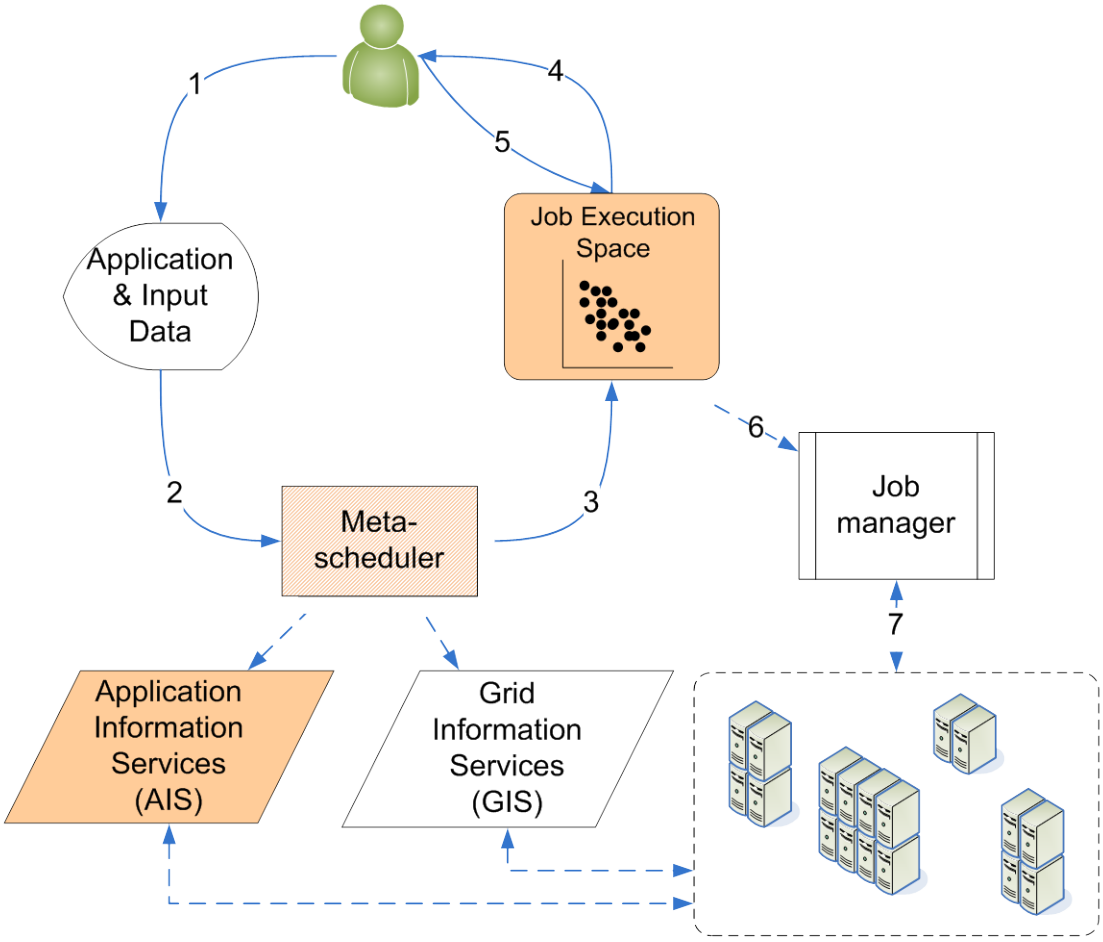


Figure 5. A high-level architecture and control flow of the devised grid metascheduling framework utilizing application-specific metascheduling and delivering two-way interaction between a user and the metascheduler Highlighted boxes indicate contribution of this work.

By adopting the described approach, the user experiences a job-specific presentation of available job execution alternatives mapped onto relevant objectives. Composition of such alternatives discloses relevant *job execution space* to the user. Furthermore, each of the job execution alternatives is an application- and resource-specific job plan aimed at maximizing performance under selected constraints (*i.e.*, by minimizing load imbalance across tasks). Through such an approach, the user is not only completely abstracted from

the low-level details of grid computing infrastructure, but is also presented with a range of job execution options allowing them to selectively choose the most suitable alternative for their current situation.

1.3.2. *Contributions*

Based on the established observations and methodologies relating performance of an individual job to characteristics of the application and a resource (*e.g.,* [14, 26, 27]), this dissertation builds on such observations with the goal of enabling and automating the job allocation process in grid environments (*i.e.,* focusing on and enabling *application-oriented metascheduling*). Furthermore, it attempts to significantly alter the interaction model between a user and the grid infrastructure by empowering the user with the potential of grid resources (*i.e.,* focusing on and enabling *user-oriented metascheduling*). These contributions can be realized with a focus on two aspects of grid job execution, as follows:

1. **Enable application-oriented scheduling in the grid to realize effective allocation of application jobs across grid resources.** Metascheduling, or allocation of user jobs to available resources, is an area covered by many projects (discussed in Chapter 2). Nevertheless, only a select few (*i.e.,* AppLeS [28] and GRADS [29]) have approached this problem strictly from the application's perspective, aiming at supporting individual applications in the context of a heterogeneous, distributed environment. Building on the foundations established by those projects, this dissertation advances the field of application-oriented scheduling by delivering a generic solution for capturing and retrieving necessary application- and resource-specific information in grid environments. More

specifically, a core set of grid services, namely **Application Information Services (AIS)** was devised and developed that enables storage and retrieval of application-specific information. Such information enables an application-independent but information-aware metascheduler to access and process application-specific information on as needed basis and interpret such information to deliver application- and resource-specific metascheduling functionality for any application. Enabling such functionality permits application requirements to be more appropriately matched with the resource capabilities, generally delivering higher resource utilization.

2. **Focus on individual user by enabling two-way interaction between a user and the scheduler to realize desired utility for the user.** With user adoption of a technology encompassing as much diversity as grid computing does, it does not take long to realize that a user alone will not be able to effectively harness technology's true potential [18]. To remedy arising difficulties, the user typically turns to a specific tool for help. However, if the selected tool is not aware of or does not explore and show the user possible opportunities, the user is hardly any better off. At that point, the user is still controlled by the technology. In order to empower the user in such a way that the technology can be customized to user's needs and desires, the user needs to be offered and presented with the true potential of the technology. By enabling application- and resource-specific information collection, delivery, and metascheduling, a user can be served on a case-by-case basis; each individual job submitted by a user can be automatically analyzed with a significant level of information about relevant requirements and

capabilities to deliver a highly rated match between the two. Furthermore, rather than assuming all the users are after the same utility, and thus acting in a fully automated fashion, by communicating with individual users, their needs are likely to be more closely matched. Therefore, rather than assuming absolute minimization of an objective (*e.g.*, runtime, cost) is the only goal any and all users are ever interested in [30], by enabling two-way communication between a user and the technology (*i.e.*, the metascheduler), custom, situation-by-situation based scenarios can be realized. By allowing such direct communication between the two entities, further customizations are possible that fit not only the current situation but are also pertinent to the participants (as opposed to creating a one-fits-all solution). In the end, such an approach offers the benefit of guiding the user through a sea of otherwise complex options. Overall, by enabling the two-way communication, the system is capable of providing valuable insight to the user regarding their job submission that the user is likely not even be aware of.

1.4. Broader Impact

In principle, this dissertation focuses on metascheduling application-level user jobs within grid environments. However, methodologies and conclusions derived surpass the domain where this work has been applied thus far and enable a much broader spectrum of functionality and principles to be developed upon now established realizations. In the context of metascheduling and grid environments, examples of such functionality include: focusing on application-oriented metascheduling alone to improve allocation of jobs across heterogeneous resources, estimation and prediction or job runtime or job wait time in a queue, or support for cost-benefit analysis.

Beyond grid compute environments alone, transitioning toward the realm of cloud computing [31], derivations of this work can enable system wide optimizations that transcend computation resources to include network and storage systems. With such developments, a true notion of Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS) paradigms can be realized where an individual user can interact with the infrastructure in terms relevant to the user and the applied domain as opposed to the infrastructure terms. Such shift enables true focus on the applications, the true driver of science and overall progress, as opposed to the focus on infrastructure and the requirement to adapt the application to the rigidity imposed by the infrastructure.

Once the understanding, automation and, ultimately, control of not only individual but also cumulative resources has been realized, a viable and flexible grid economic model can be developed. Development of an economic model would further enable composition and realization of various grid and cloud computing marketplaces that support notions such as transparent resource sharing, offloading, provisioning and migration. Concrete ideas being development of automated computation and storage bidders that can be relied upon, realizing apples-to-apples comparison of individual clouds, all leading toward semantic provisioning of services based on context (*e.g.*, spatial or temporal locality).

1.5. Overview

The remainder of this dissertation is structured as follows: Chapter 2 presents details about grid computing and grid scheduling approaches. More specifically, the first part of the chapter describes grid computing, associated middleware, grid applications and basic concepts behind grid schedulers. The second part focuses on more specific examples and

instances of work related to the work presented in this dissertation. Current state and developments in the area of grid schedulers is presented. Additionally, some lower level concepts dealing with collection of application-level information and application benchmarking are presented. Chapter 2 concludes with a critique of the currently available tools and technologies and compares those to the hypothesis presented in this dissertation.

Chapter 3 presents the overall approach of this work, dealing with improving user's experience when executing jobs across the grid. The problems with current approaches are summarized and a rationale for the proposed model is presented. The complete architectural overview is presented, illustrating the proposed approach.

Chapter 4 focuses on enabling the first contribution of this dissertation, namely application-oriented metascheduling that is decoupled from the metascheduler itself. The chapter presents a general framework for realizing application-oriented metascheduling accompanied with the analysis and modeling of application performance to realize sought goals.

Chapter 5 presents individual case studies that implement and realize general framework presented and described in Chapter 4. Two different applications are studied and performance results presented, showing direct benefit of the proposed approach. The later part of this chapter focuses on the second contribution of this dissertation, realizing user-oriented metascheduling; it presents real-world examples and an analysis study of how the devised approach can help in alleviating a grid end user from having to deal with low-level infrastructure details and instead effectively realize and utilize application-oriented metascheduling approach presented earlier.

Chapter 6 summarizes and concludes the work, highlighting major contributions. Finally, Chapter 5 provides insights into some future extensions and future potential of presented work. Figure 6 provides a visual overview of the individual chapters summarizing each individual chapter's topic and provides a brief content summary.

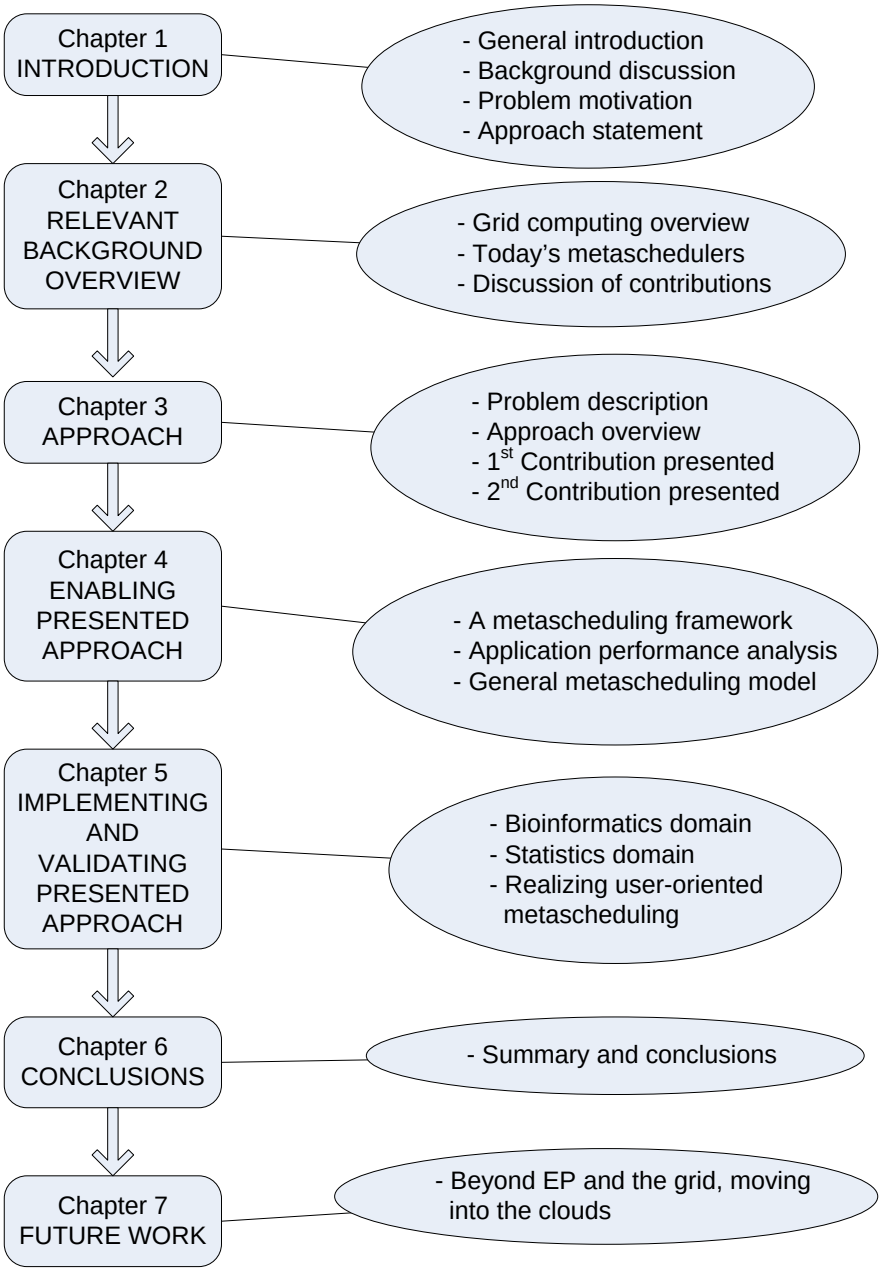


Figure 6. Visual chapter layout and content summary

2. BACKGROUND AND MOTIVATION

This chapter presents general background information regarding the field of grid computing followed by a comprehensive overview of previous work related to the topics of this dissertation. Existing projects in the fields of grid job scheduling and application performance adjustments are analyzed and compared to the work presented herein.

2.1. Glossary of Frequently Used Terms

In order to establish clear understanding of terms most frequently used in this document, a glossary of terms and their corresponding definition is provided in this section. This glossary is organized in a logical order.

- *Grid* - refers to a hardware and software infrastructure composed of multiple resources that do not belong to single administrative domain, use standard and general purpose protocols and interfaces, and deliver non-trivial QoS [6]. Note that terms grid and grid environment are used interchangeably.
- *Cloud* - style of computing in which dynamically scalable and often virtualized resources are provided as a service over the Internet
- *Utility* - a measure of value delivered to the user as a function of job execution time
- *Resource* - although a resource can represent any (physical or logical) entity that exists as part of the grid ecosystem (*e.g.*, compute resource, software application, instrument) [7], in this dissertation a resource refers to a physical compute machine that can deliver meaningful computation

- *Cluster* - collection of individual but connected computers
- *Node* - a portion of a resource or a cluster
- *CPU or processor* - are used interchangeably and represent a physical central processing unit (CPU) on a node that can execute a computer program
- *Core* - represents a single computer processing unit. A CPU may contain multiple, independent cores that are managed by a local operating system
- *Grid application* - computer application that, when provided with input data, performs necessary computation and delivers sought results (general discussion of grid applications is presented in Section 2.3)
- *Job* - an instance of a grid application associated with input data
- *Task* - a job can be decomposed into multiple tasks. Individual tasks operate on portion of the job input but are still associated with the same grid application
- *Job-level information* - information stored at level of a job (*i.e.,* resources a job consumed, performance of job components across those resources)
- *Task-level information* - the same as job information but at the level of a task
- *Job parameterization* - understanding and selection of job parameters (*i.e.,* user controllable and application dependent options that can be changed when submitting a job, such as number of processors employed, algorithm used, and data distribution) that are algorithm, input data, and resource dependent
- *Task parameterization* - understanding and selection of task parameters (*i.e.,* user controllable and application dependent options that can be changed when submitting a task, such as number of threads employed or algorithm used) that are algorithm, input data, and resource dependent

- *Job execution option* - a single parameterization of the job
- *Job plan* – a defined instance of job parameterization
- *Process* - represents an execution instance of a grid application bound to a specific resource. A process is further associated with all the standard operating system details [32]
- *Thread* - represents a light weight process as defined in the scope of operating systems [32]
- *Embarrassingly parallel application* - an application whose input data can be divided into smaller segments so that individual segments can be processes by the application as a set of independent, coarsely grained, and indivisible tasks
- *Embarrassingly parallel job* - an instance of an embarrassingly parallel application
- *Job Manager* - a computer program that coordinates submission and instantiation of jobs on a resource
- *Scheduler* - a computer program controlling submission and management of jobs to individual nodes of a cluster
- *Metascheduler* - a scheduler that submits jobs across different resources in a grid. Also known as a super scheduler or a resource broker.
- *Load Balance* - technique of distributing tasks (*i.e.,* workload) of one job across multiple resources in such a fashion that runtime characteristics of any one task are not different from runtime characteristics of any other task by more than some small delta

- *Load imbalance* - converse of load balance. Runtime characteristics of at least one task are greater than the delta
- *User or end user* - a person using grid resources or grid applications (additional details and user categories are provided in Appendix A)
- *Heterogeneity* - refers to structural and behavioral variations found across grid resources. Structural and behavioral variations refer to physical and logical differences that exist between individual resources
- *Fail-safe capability* - capability of an application or job submission tool to migrate application execution when preset threshold is exceeded and acceptable operation is resumed
- *Excess capacity* - a situation where actual resource capabilities is less then what is achievable or needed for application execution optimum
- *Partial failure* - failure of one or more tasks comprising a job

2.2. Grid Computing

Since the inception of computer science, excluding the few individuals who truly enjoy fiddling with the low-level details required to obtain useful computation from a machine, the majority of people are primarily interested in abstracting details required to operate a computational device to the level where it operates in their familiar domain (*e.g.,* operating systems, compilers, high-level programming languages, Web 2.0). As a step in that direction, grid computing is a technology and an approach that abstracts and hides complexities required to operate a set of resources in a geographically and administratively distributed environments [2]. Often, an analogy of the electric grid is used to describe grid computing [33]. Similar to the power plugs implemented in every

household or office, allowing any compatible appliance to come to life without regard to where the power is coming from or how it is being managed, one would be interested in simply connecting to a world-wide network and in turn gain access to any compute or data demand imaginable. In such scenario, the user should not be concerned with where the demanded resources are located or how are they managed; the user is simply interested in satisfying their utility with as little knowledge as possible. Although such a vision is still somewhat farfetched, the first step toward such a goal is unification and virtualization of available resources. In order for a resource to be capable of joining the grid, and thus being abstracted and brought together under a unified interface, it only needs to possess an interface to the world that enables it to communicate with other such resources [12]. Consequently, grid resources can include an expansive range of devices ranging from a typical personal computer or a supercomputer, a sensor, a satellite, a microscope, an automobile, down to even a single hard drive. These devices may span various geographical and administrative domains, each of which can be independently administered and maintained. Aggregation of available resources that span physical, political, and organizational boundaries enables creation of Virtual Organizations (VO) [2]. Examples of VOs include: researchers at multiple national labs and universities collaborating on a single project to deliver a new medicine, analysts collecting data after a plane crash trying to reconstruct the event, company with multiple branches around the world wanting to maximize utilization of available compute resources by balancing cumulative workloads across individual branches as they experience low loads. As depicted by these examples, VO's can, depending on the needs of the participants, vary greatly in their scope, size, structure, purpose, and duration. One common principle that

all VOs share is that they are created with a desire to bring together people with similar goals in mind and thus enable more effective collaboration of individuals trying to reach a common goal. Major benefits resulting from the creation of such organizations is an ability to transparently share knowledge and technology that might otherwise exist but not be interoperable. Overall, goals behind individual VOs are best understood and summarized through the well-known saying originating from Aristotle's Metaphysica: "the whole is greater than the sum of its parts."

Existence of the grid in general, and thus the above-mentioned VOs, is enabled by middleware. *Middleware* enables ubiquitous access to distributed resources that are shared between multiple organizations through virtualization and aggregation. Middleware is typically implemented in the form of standards allowing general interoperability among participants. In other words, implementing open standards ensures that participants can take advantage of any and all resources that are made available, thus creating and supporting notions of a unified environment.

With such a wide spectrum of functionality that the grid aims to cover, it is necessary and important to develop standards in a range of areas, including security, scheduling, accounting, job management, specific application categories, and so on. Currently, the standards body defining such standards in the area of grid computing is the Open Grid Form (OGF) [34].

The Globus Toolkit (GT) [35] is currently the most accepted middleware framework for enabling the existence of grid infrastructure and the creation of grid applications, although alternative implementations exist, such as UNICORE [36], gLITE [37], Alchemi [38], and Legion [39]. GT offers all the basic technologies required when setting

up a grid environment: Grid Security Infrastructure (GSI) [40] used for user authentication in the system, the Global Resource Allocation Manager (GRAM) [41] used for remote resource job management, Grid Information Services (GIS) [25] used for information discovery and state of resources, and GridFTP [42] used for providing file transfer capabilities in the grid environment. Use of these technologies enables support for users and their jobs. In the context of a computational grid, a *job* is defined as an instance of an application that is ready for execution or is being executed on chosen resources. Because of middleware technologies, individual resources can be abstracted into a unified entity capable of being accessed through a defined interface [2]. For example, individual resources belonging to administratively independent organizations, but joined through a common VO, can publicize their availability. Grid middleware services enable access to and control over those resources' availability and use policies for grid jobs. These policies are primarily imposed and controlled by local administrative guidelines, permitting entire resources to continuously be made available for sharing among VO participants, only portions of one resource be made available, or even to impose preference for local jobs over grid jobs. Resource availability can be publicized as a service through standard protocols and aggregated at a well-known location (*i.e.*, GIS). Higher level tools, such as resource brokers [43] or job submission managers [44], can contact this service to obtain the resource availability information. Subsequently, jobs can be scheduled and executed on those resources. With adoption of a grid computing, a single job instance may span traditional boundaries imposed by any individual machine and thus simultaneously and seamlessly execute across multiple grid resources. Such functionality is made possible through standardization of communication protocols and

existence of mentioned core grid services, leading towards a global and open infrastructure built on top of otherwise heterogeneous foundations.

Building on top of the grid middleware and core grid services that enable interoperability of independent and heterogeneous resources in a standardized way, higher level service, tools and applications can be developed. A layered diagram representing the overall grid computing infrastructure is shown in Figure 7. In this figure, the bottom layer represents hardware resources and connectivity fabric. The goal of the grid is to provide a ubiquitous access to the functionality offered by those resources. Those resources are networked machines: heterogeneous, geographically distributed, and offer variable performance, usability, and availability policies. It is obvious that such resources can support a wide spectrum of functionality. However, in order to support basic interoperability among participant and yet impose minimal requirements on those, from the perspective of grid architecture, fabric resources only need to support two mechanisms: enquiry and resource management. Enquiry mechanisms enables discovery of the resource, its functionality and state, while resource management enables some level of control over the resource (*e.g.*, job submission, job monitoring). Nonetheless, because these resources are controlled by individuals or independent organizations, the resources are free to offer any services as deems beneficial by their owners. Applications installed on those resources, their characteristics, access and control policies are all in the hands of the owner to control and manage.

The next layer up in the grid infrastructure is a layer of software and standards that enables virtualization of individual resources and thus provides ability for more targeted

applications and services to be developed on top of it. This is the middleware discussed in the previous paragraph.

Above the middleware, the next layer up provides needed interfaces and services to abstractly deal with low-level details enabled by the grid middleware. Rather than operating at the level of an individual resource or an individual service, this layer operates at the level of aggregating and coordinating multiple resources. This layer can be seen as a set of tools and services that are capable of and are in charge of simplifying actions such as authentication, resource selection (*i.e.*, job managers, metaschedulers), user and job access options, etc. As opposed to the generality of the lower layers where simplicity was a key virtue, at this layer, tools can provide specialized, complex and targeted functionality.

The top most layer is representative of applications and users. This layer offers the widest range of the functionalities, but it is also dependent on ability of lower level details to deliver and sustain needed services. Grid applications are discussed in detail in the next section, so only some general thoughts are provided here. Although the core idea of grid computing fits into the earlier described analogy of the power grid, there are some differences too. In regard to the power grid, only compatible appliances can be connected to the available plugs. So, for example, one cannot plug a 220V shaver brought from Europe into the standard 110V plug in the United States. Similarly, some applications will not benefit from the availability offered by the grid. As can be observed from the layered diagram presented in Figure 7, there are multiple levels of indirection between an application and the actual physical resource. Therefore, there is considerable overhead involved with executing an application on the grid and the cost associated with this

overhead may prove to be unacceptable for a specific purpose. For example, a calculator application that adds two integers would hardly benefit from sending and performing needed calculation on a grid resource as opposed to the machine it is being invoked on. Although not all computations are well suited for the grid, there are numerous ones that are and some may even be a result of the current situation. An example can be found in an overloaded local resource. As opposed to waiting for ongoing computations to complete, farming out available work to otherwise idle resources may prove beneficial even though, in case of local resource's immediate availability, involved computation would not warrant such a decision.

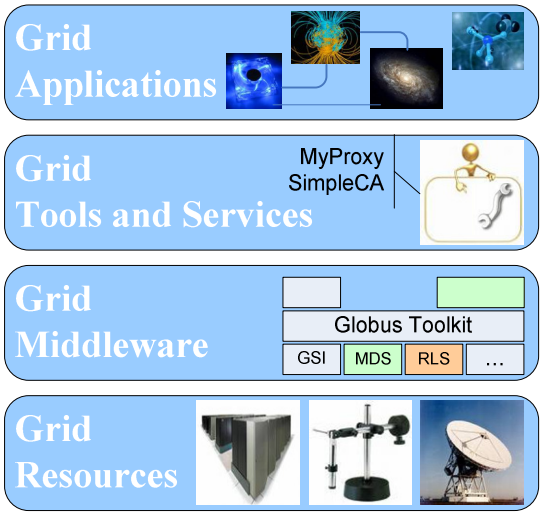


Figure 7. Layered structure of grid computing

2.3. Grid Applications

The top most layer of grid computing infrastructure is composed of grid applications. These applications are built on top of the abstractions and generalizations provided by the lower level layers of the grid environment and they aim at solving specialized problems. Aim of grid applications is to make extensive use of underlying infrastructure layers with the goal of improving given application's performance, availability, utilization, accuracy,

etc. Available applications can be perceived from users as services capable of performing a specific task and delivering needed results without user's need to immerse themselves into the intricacies of grid computing environments. Availability of an infrastructure such as the grid enables existence and delivery of applications to its users that were not necessarily available beforehand. Even if such services were available, grid simplifies associated requirements and thus broadens availability of sought entities (regardless of whether those entities are computation power, access to specialized equipment, or data availability and persistency).

From a high-level, stemming from the primary reasons for using the grid, grid applications can be grouped as follows [45]:

- *Community centric*: these are applications organized around VOs by joining people together for purposes of collaboration in solving a specific problem or working on a specific project.
- *Data-centric*: applications are grid applications that utilize grid resources to store, transfer and deliver necessary data. The reasons and benefits of using the grid for this type of applications can range from the inability of any one system to store generated data (*e.g.*, Large Hadron Collider project [46]) to ubiquitous access to one's data and applications (*e.g.*, online operating system [47]). From the perspective of Web 2.0 and beyond [48], this type of applications are likely to become the most popular way of delivering grid computing functionality to individuals.
- *Computation-centric*: contain the traditional high-performance computing (HPC) applications that can benefit from the additional compute power brought about

through adoption of grid computing. These applications can range from generic Monte Carlo simulations to specific protein analyses. Depending on the application, the way they make use of newly available resources differs and can thus range from simply gaining access to larger numbers of resources to execute across or by executing on exotic hardware without which the computation would initially not even be feasible.

- *Interaction-centric*: are the applications requiring or being enhanced by real-time interaction with a user. Primary example of this class of applications would be a visualization application that uses grid computing resources to render a complex image and displays it on a large screen for the user to manipulate. Realizing support for this type of applications requires a well-defined communication stack, standards and protocols so that the real-time aspects can be delivered.

Despite the many benefits of grid computing, mentioned applications, and application categories, the grid itself does not provide a novel programming paradigm for developing new applications (some efforts have emerged recently to support such ideas [49]). Additionally, no formal methodology exists for porting existing legacy applications to the grid. Grid applications can be constructed by calling on any of the services defined in the lower layers and integrating functionality of those services or building on top of them to deliver higher level services [7]. Nevertheless, most of the applications developed for the grid are based on traditional HPC or distributed computing principles (*e.g.*, [4]). HPC applications are typically developed using a specific programming language and a parallel programming paradigm (*e.g.*, compiler directive-based, threads, message-passing, combination of threads and message-passing) and often times the

programming paradigm chosen decides the application deployment platform. If the application uses a shared-memory programming paradigm then the application can be only deployed on a shared memory system whereas an application developed using the message-passing paradigm can be deployed on both distributed memory and shared memory systems. Furthermore, applications might require specific processor architecture, amount of memory, disk space, etc. to deliver desired performance and scalability.

Following the development process, the application deployment is the process of installing the application on a set of resources. Because of the requirements and even preferences the development process imposed, the deployment process is a non-trivial task. Even if the deployment process is automated, it is necessary to first determine what resources are available and then decide which one is most suitable resource for that particular application. If the process is not automated, deploying an application on the grid requires additional steps that involve user intervention, great insight into the internal structure of the application, and familiarity with the various grid computing technologies and toolkits [13].

Beyond the deployment process, the progressive steps of application execution and job submission may involve many additional steps required from the end user, not necessarily found in a typical application (*e.g.,* execution resource selection, input data locality, load balancing across heterogeneous resources). In order to address this inherent complexity and difficulty of using the grid, several approaches have attempted to simplify grid deployment and configuration by developing technologies such as web portals [50], workflow systems [51], and component assembly [52]. The ultimate goal of such efforts is to enable the adoption of grid technologies and applications to a wider

group of end users who are not familiar with programming languages and the lower level grid infrastructure. The potential impact for improving grid accessibility to such users is significant because such users are typically the ones with the largest problems (*e.g.*, applied science researchers, distributed organizations, and organizations with variable computational requirements).

2.3.1. *Grid Application Classification*

Because different classes of applications impose different requirements onto a metascheduler (as outlined in Section 1.1 and further discussed in Section 2.8.3), the metascheduler (discussed in Section 2.5) needs to support functionality specific to an application. In order for a metascheduler to transcend any single application, individual applications must be classified into application categories; such classification enables a metascheduler to immediately address multiple applications and yet realize desired objectives. Based on the communication pattern implemented by applications, the following grid application categories have been defined [53]:

1. *Sequential applications* – Traditional applications developed to execute on a single node resource. For the applications that require larger resources (*e.g.*, more memory, disk, or faster CPUs), the grid also provides redundancy, fail-safe capability, and excess capacity.
2. *Parametric sweeps* - Multiple copies of sequential jobs using different input datasets or parameters. These applications are often submitted independently by a single user in an effort to reduce overall task execution time. Benefits of using the grid are the same as sequential applications with the addition of multiple instance coordination performed by grid tools and middleware.

3. *Master-Worker applications* – Master-worker or the bag-of-tasks model [4], where a master process distributes work (either statically or dynamically) to a set of worker processes and aggregates the results at the end. Many financial and bioinformatics applications fall into this category, each in constant need of surplus compute resources. The main differences between parametric sweeps and master-worker applications is that the individual tasks do not have to be executing the same code, but a workflow system can be in place with the master-worker model possibly delivering a more complex application functionality by structuring execution of different codes into a meaningful unit. The master process must handle the coordination and task assignment between worker nodes.
4. *All-Worker applications* – Similar to the master-worker model, except that each process involved, including the master, share the workload equally and data is exchanged between individual processes in some pattern (point-to-point or group communication). This class of applications is also known as task parallel.
5. *Loosely coupled parallel applications* – Parallel applications (*e.g.,* coupled fluid flow and wave models) that exchange data occasionally through files during execution (*e.g.,* beginning and ending of an outer iteration). This class of applications is also known as data and/or task parallel.
6. *Tightly coupled parallel applications* – A single Message Passing Interface (MPI) [54] application distributed across multiple systems sharing data during the execution, possibly at frequent intervals, through explicit messages. It requires interoperability between different MPI libraries or an MPI library such as MPICH-G2 [55].

7. *Workflow applications* – A model connecting many individual applications executing at different geographically distributed locations, which are chained together to perform a complex simulation. For an application to be classified as workflow application additional information is needed to identify dependencies with other applications in terms of input and output data streams (also see Section 2.6.9).

2.4. Grid Languages and Technologies

Interaction between users and applications in the grid is done through a set of open standards (*i.e.,* languages), some of which are custom to the field of grid computing that were inherited from existing technologies (*e.g.,* XML, SOAP). In this section, we look at grid languages relevant to presented work. Necessary background and interaction among languages is discusses along with appropriate mappings to grid user categories as defined in Appendix A.

2.4.1. *Resource Specification Language (RSL)*

Developed as part of the Globus Project, Resource Specification Language (RSL) [56] provides a common interchange language to describe resources and jobs to run on them. RSL is represented by various <name, value> pairs and is used by GRAM [41] to perform complex resource descriptions in cooperation with other grid components. RSL represents a resource from the job instance point of view; each of the name-value pairs represents one or more components in resource management system, as can be seen in Figure 8:

```
1. &(rsl_substitution = (TOPDIR "/home/afgane")
2.     (DATADIR $(TOPDIR)"/data")
3.     (EXECDIR $(TOPDIR)/bin) )
4.     (STDDIR $(TOPDIR)/std) )
5. (executable=$(EXECDIR)/xhpl)
6. (arguments=$(DATADIR)/HPL.dat)
7. (directory=$(TOPDIR))
8. (count=4)
9. (jobType=mpi)
10. (stdout=$(STDDIR)/hpl-output)
11. (stderr=$(STDDIR)/hpl-error)
```

Figure 8. Sample RSL document for an MPI type job

The above RSL document specifies the executable and data directories along with the respective files. It declares the job is an MPI type job and should be executed on four nodes. Finally, it specifies the names of standard out and standard error files. As an evolved version of the above RSL, with the advent of GT4, the RSL2 was formed. It is based on XML schema and is used when submitting jobs in service oriented paradigm.

2.4.2. *Job Submission Description Language (JSDL)*

Built on the same principles as RSL, the Job Submission Description Language (JSDL) [57] is a specification of an abstract and independent language used for describing requirements of computational jobs in grid environments. JSDL is a standard defined by the OGF [34] and the Job Submission Description Language Working Group [58]. It contains a vocabulary and normative XML schema, thus accommodating for interoperability among a variety of job management systems and alleviating some of the problems related to the grid heterogeneity. It focuses on providing interfaces to a subset of all available functionalities found in resource management systems. By having a standard language available, a job submission description can enable diverse job management systems to easily communicate and thus complement job description at different stages in the course of the job submission process.

Figure 9 depicts a sample JSDL document used for submitting a BLAST job. It provides a job name and corresponding description along with application and executable used, input and output files, and the required hardware requirements in terms of the number of CPUs, and amount of main memory required to execute the job.

```
1.<?xml version="1.0" encoding="UTF-8"?>
2.<jsdl:JobDefinition xmlns="http://www.example.org/"
3.  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
4.  xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
5.    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
6. <jsdl:JobDescription>
7.  <jsdl:JobIdentification>
8.    <jsdl:JobName>My BLAST invocation</jsdl:JobName>
9.     <jsdl:Description>Sample BLAST invocation.</jsdl:Description>
10.   </jsdl:JobIdentification>
11.     <jsdl:Application>
12.      <jsdl:ApplicationName>BLAST</jsdl:ApplicationName>
13.      <jsdl-posix:POSIXApplication>
14.        <jsdl-posix:Executable>/usr/local/bin/blastall</jsdl-
15.         posix:Executable>
16.        <jsdl-posix:Argument>-p blastp -a2 –d nr</jsdl-
17.         posix:Argument>
18.        <jsdl-posix:Input>input.fas</jsdl-posix:Input>
19.        <jsdl-posix:Output>output1.txt</jsdl-posix:Output>
20.      </jsdl-posix:POSIXApplication>
21.     </jsdl:Application>
22.     <jsdl:Resources>
23.      <jsdl:IndividualPhysicalMemory>
24.       <jsdl:LowerBoundedRange>2097152.0</jsdl:LowerBoundedRange>
25.      </jsdl:IndividualPhysicalMemory>
26.      <jsdl:TotalCPUCount>
27.       <jsdl:Exact>1.0</jsdl:Exact>
28.      </jsdl:TotalCPUCount>
29.     </jsdl:Resources>
30.    </jsdl:JobDescription>
31.   </jsdl:JobDefinition>
```

Figure 9. Sample JSDL document describing a BLAST job

Both, RSL and JSDL are seen as end user tools enabling individual jobs to be described. These languages allow job parameters, ranging from input file names to resource preference, to be defined prior to their submission to a job management tool. The main difference between RSL and JSDL is that JSDL is a generalization of RSL and

has been accepted as a global standard by OGF, while RSL is middleware specific implementation supported by the GT community.

2.4.3. *Resource Description Language (RDL)*

Because, theoretically, any device with network connectivity is able to join the pool of grid resources and thus offer its functionality to grid users through grid tools and services, those grid tools, such as job submission engines and metaschedulers, need access to data describing resources they operate on. Metadata describing available resources would include information about the functionality, the status of a resource, usability and access policies, VO information, usage preferences, as well as any additional data that may be resource specific. Because many applications in grid environment could use such data, it would be beneficial in terms of standardization, overlap, maintainability and evolution for such data to be made available and standardized. Without such data, grid application programmers must create ad-hoc methods to support needed functionality. This may ultimately lead to many data representations requiring numerous conversions and transformations between the formats, automatically reducing interoperability, and decreasing performance.

Currently, there is no such standard available in the area of grid computing, even though discussion regarding the topic has existed. Initially, a standard called Resource Description Framework (RDF) [59] has been developed as part of W3C community. RDF is an XML language used to represent resources in World Wide Web including authorship, licensing, and schedule of a shared resource. Although more specific to web community, goal of RDF was to support notions such as the Semantic Web [60]. Requirement for a language similar to RDF, but targeted specifically at grid resources,

has been stated by two working groups in Open Grid Forum (OGF) [34]. Namely, Job Submission Description Language Working Group (JSDL-WG) has stated in the scope of JSLD a need for a language that describes grid resources. Mentioned language is referred to as Resource Requirements Language (RRL) [57] and is described as a language capable of capturing resource specific information enabling matching of JSDL information against RRL information. The other working group within OGF that has expressed interest in a resource description language is Grid Resource Allocation Agreement Protocol Working Group (GRAAP-WG). Although both working groups have stated requirements and desires for a language able to describe grid resources neither have incorporated it into their respective standard, stating that scope of such language is too large to be incorporated into another language and that it should thus be considered as a standalone specification.

Although no specific standard exists today, tools have been adopted to accomplish the task. Grid resources are in need of communicating and negotiating among themselves automatically without human intervention. Web Service (WS) [61] is the technology that proved itself in this context as the platform and language independent technology that works. WS is a distributed technology that allows creation of loosely coupled client/server applications and the technology is thus appropriate to be used as a building block for grid applications. However, one mayor requirement not currently supported by the standard WS is the need to keep state, which means that separate invocations of the same service need to be able to tell how were they manipulated previously. Web Service Resource Specification (WSRF) [62] specifies how to create stateful services. WSRF is built on top of the existing WS standard. The specification and implementation of WSRF

does not make individual WS keep state, but rather WSRF introduces idea of a WS-Resource [63] which provides separates service implementation from the state. For a client to access state, it is necessary to specify the resource and also the service. Once the service implementation receives the request, it retrieves information from desired resource and performs the operation on it, thus updating the state. Specifications closely related and implied by the WSRF specification are the WS-Notification [64] and WS-Addressing [65] supporting subscription to the change of state of a service and an effective mechanism to specify given WS and the corresponding resource. Adding notification to the existing model simplifies functionality and usability of the grid infrastructure significantly since tracking status of individual resources can otherwise become a complicated problem.

In the context of grid resources, although not developed with this goal in mind, WSRF can be seen as a tool to implement and publish needed information about grid resources by resource owners so provided information can be accessed in an automated way. Through this model, resources can be described in similar terms as used by end users user JSDL to describe their job requirements, thus allowing necessary matching to be, at least partially, standardized and automated.

Figure 10 depicts interactions between described technologies and user categories (described in Appendix A). An observation about user roles can be made from this figure: currently, user categories are clustered and users are forced to use the same technologies and adopt them to fit their individual and independent needs. Later (Section 3.5.1), it will be shown how this is no longer necessary with adoption of technologies and tools presented as part of this dissertation.

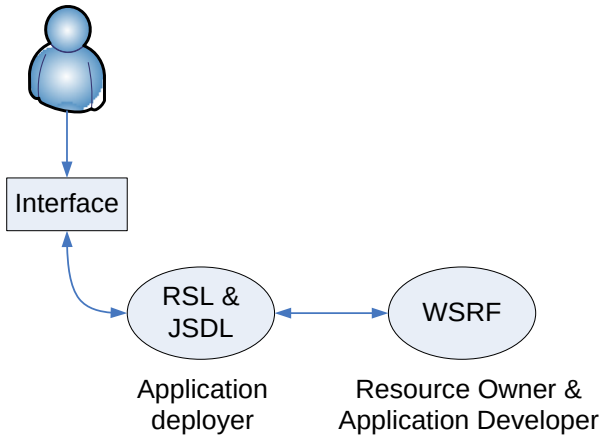


Figure 10. Relationship between end users, WSRF, RSL and JSDL at the basic level showing user's need to directly interact with the low-level infrastructure components (i.e., RSL & JSDL).

2.5. Scheduling Background

Over the years, scheduling has typically been associated with assignment of goods to available resources. Much research has been in the past on scheduling algorithms such as the Job-Shop Scheduling and the Transportation Problem [66]. In the context of parallel compute environments and supercomputers however, scheduling typically refers to selection of jobs in the ready queue for execution and assignment of those to available compute nodes [67]. In the context of the grid, scheduling (often and interchangeably referred to as metascheduling or resource brokering) is defined as the process of making a decision on which resources to execute submitted jobs [68]. This decision process involves searching multiple administrative domains and multiple resources to match requirements imposed by a job to the capabilities offered by available resources. The requirements imposed by the application and corresponding job may require the metascheduler to schedule the job on a single machine or allow scheduling across multiple grid resources. Because of the multitude of possibilities and intermixed with the lack of local resource control metaschedulers must deal with (further elaborate on in

Section 2.5.2), the scheduling process offers a potential for great variation in terms of job execution characteristics but also great difficulty in terms of achieving an optimal solution [14]. Overall, because the grid provides access to a heterogeneous pool of resources, a large potential lies behind grid application scheduling and the associated service it provides to the end user. Therefore, grid job scheduling can be seen as residing at the heart of the grid and thus influencing overall success of the infrastructure. As of writing of this dissertation, infrastructure management (*i.e.,* metascheduling) is still characterized as being not only difficult but also inadequately handled [69]. Before delving into the general approach and solutions provided by this dissertation, the remainder of this section looks at the major components and differences between local resource schedulers and metaschedulers, paving the path for comprehending the remainder of the document.

2.5.1. *Local Resource Managers*

In order to hide complexities of resource usage alongside simplifying maintenance, larger resources (*i.e.,* computer clusters) often employ Local Resource Managers (LRM), which control user access and job submission. These schedulers benefit from a high-level of system privileges and control since they reside directly on a resource, they are tightly coupled with the operating system, and have direct access to the operating system tools. Examples of such high-level control include insight into applications in the queue waiting to be scheduled, instant knowledge of the current system load, status of individual jobs and corresponding error/log messages, including accurate status of individual nodes within a given resource. Because of the given level of control, LRMs are capable of implementing an array of effective scheduling policies and are capable of handling

dynamic priorities, extensive reservations, fair-share capabilities, grouping of jobs and many more. Examples of LRMs are: Portable Batch System (PBS) [70], Sun Grid Engine (SGE) [71], Platform Load Sharing Facility (LSF) [72], and IBM LoadLeveler [73].

2.5.2. *Grid Metaschedulers*

Unlike the local schedulers, metaschedulers (also referred to as grid schedulers or resource brokers), operate at a higher hierarchical level. They cross administrative boundaries and thus must deal with the heterogeneity of underlying systems, dynamic resource availability, lack of access and policy control, lack of available information or dated information on both, resources and submitted jobs, as well as application and resources scalability issues [68]. The efficiency of a local scheduler is mostly measured in terms of throughput and resource consumption; therefore, it can be seen as focusing on the resource owner interests by trying to keep the resource as busy as possible. In the grid realm, this model is modified and grid schedulers are performing their operations with the end user in mind. Rather than monitoring cumulative throughput, metrics such as turnaround time, cost and speedup are more valued [74]. The criterion for application metascheduling involves interpreting resource information and metascheduling from the application's standpoint with the aim of maximizing end user goals. Figure 11 depicts the general interaction diagram between users and grid metaschedulers. As can be seen in the figure, the end user is not in direct contact with available resources, but rather a higher level service is offered that abstracts individual resources and resource interactions. The act of metascheduling can thus be seen as a decision making process of selecting one or more resources across the grid environment that most closely match user request (*e.g.*,

resource with three CPUs and 2 GB main memory) or more generalized requirements (*e.g.*, minimize runtime).

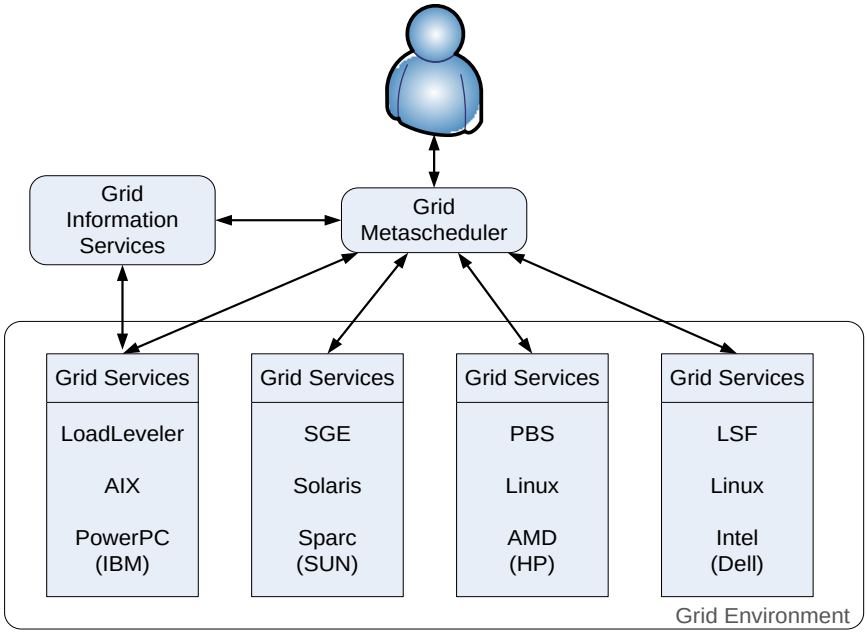


Figure 11. Interactions and relationship between an end users, a grid metascheduler and grid resources

As with any scheduling model, grid application metascheduling is based on information. Rather than focusing on resources alone, the criteria used for resource selection at the application level includes:

- Static and dynamic information available from grid services
- Available application and resource meta-information to improve performance
- Use of real applications performance metrics over theoretical resource performance benchmarks alone
- Restrict domain of the metascheduler
- Use of application specific information (*e.g.*, application preferences from application developers)

Useful and desired information needs to be collected individually and subsequently combined and processed into a meaningful result. Depending on the size of given scheduler's control, level of granularity, and access rights, this information is available from several sources, the main one being Grid Information Services (GIS) [25]. GIS is a collection of services offering information on grid resources and available services. Through a set of query and subscription interfaces, GIS allows users and applications (schedulers) to discover and monitor individual resource belonging to a VO. Other options for information gathering in the grid environment include Ganglia [75], Network Weather Service (NWS) [76] and, partially, MonALISA [77]. Ganglia is a scalable distributed monitoring system used for performance monitoring of grid resources. It uses scalable technologies that impose minimal load on given resource while providing per-node status of individual resources. An observation worth noting is that Ganglia does not use Grid Security Infrastructure (GSI) [40]. Similarly, the NWS is a performance forecasting application used for computation and networking resources. Through a set of distributed sensors and monitors, it gathers a set of readings about the current resource status and uses mathematical models to forecast near-future resource load and performance. Finally, Monitoring Agents using a Large Integrated Services Architecture (MonALISA), geared mainly toward Data Grids [78], is an agent-based system that, through a set of dynamic services, is able to perform information gathering and processing of tasks. It is used to monitor site resources, network and jobs currently executing. The information is mainly used in optimization decisions for large-scale distributed applications.

2.5.3. *Types of Metascheduling*

The broad field of metascheduling can be initially categorized into three subcategories or types, namely system-oriented metascheduling, application-oriented metascheduling, and user-oriented metascheduling (see Figure 4) [67, 79]. The system-oriented metascheduling is representative of the traditional scheduling (described earlier in Section 2.5) where system throughput and utilization are primary goals and metrics [80]. Focus on such metascheduling typically targets environments where resources are provisioned and managed by a single organization (*e.g.*, cloud computing) and the primary goal of an organization is maximizing throughput across its resources. Chapter 5 will present an elaboration on extensibility of current work into such environments.

The application-oriented metascheduling (further defined in Section 3.3) deals with understanding and leveraging the dependency that exists between an application and a resource. The user-oriented metascheduling (further defined in Section 3.3) deals with metascheduling jobs that primarily focus on realizing user's goals (*e.g.*, minimize job runtime, maximize result accuracy). Work presented throughout this document (as outlined in Section 1.3.1 and further discussed in Section 3.3) explores and advances the application- and user-oriented types of metascheduling. These types of metascheduling are realized by raising QoS delivered to a user, namely realizing shorter job turnaround time (see Section 5.1.2 and Section 5.2), and providing detailed insight into job execution alternatives (see Section 5.3). As described in just mentioned sections, such goals are achieved through leveraging benefits of the application-oriented metascheduling.

Each type of metascheduling may be further classified into single objective optimization, multi objective optimization, or tradeoff presentation. This classification is

based on our analysis and categorization of projects described throughout Section 2.6. The choice of the approach (*i.e.*, single objective optimization, multi objective optimization, or tradeoff presentation) implemented by a metascheduler is based on the desired goal of a metascheduler and it is decided upon at the time of metascheduler development. Once the approach for the user-oriented metascheduling has been finalized, specific objectives can be realized by the metascheduler; for example, maximize throughput or minimize runtime and maximize accuracy.

The categorization described and depicted in Figure 4 is only one of many (for an alternative, see [79]), and it focuses on encompassing the most general case, one that effectively enables classification of individual metaschedulers regardless of the purpose for which they were devised. Furthermore, the categorization can be used to serve as a general overview of the field of metascheduling. Figure 4 also depicts viable connections between individual classes.

2.6. Grid Scheduler Approaches and Implementations

With the main topic and contribution of this dissertation being in the area of metascheduling, this section focuses on presenting and analyzing relevant work. Presented projects represent a comprehensive survey of relevant projects, including projects and tools that have lead to the development of hypotheses presented in this document. Also, projects and tools of similar functionality are presented with analysis of differences and contributions provided through this dissertation when compared to those projects.

2.6.1. *Community Scheduler Framework (CSF)*

As the wider community recognized the obvious need for a metascheduler [43], plus the inherent difficulties associated with developing one [43], a Community Scheduler Framework (CSF) [81, 82] was developed to leverage some of development difficulties. CSF is an open-source implementation of a WSRF-compliant metascheduler framework that supports two scheduling algorithms (first-come-first-serve and round-robin) with the main goal being to provide standardized interface to implement grid metaschedulers [83]. CSF is distributed with the GT and it provides a standardized interface and tools for the end users to perform job submission, advance reservation, and an option to define different scheduling policies. It uses GRAM protocol from the GT to provide access to services offered by resources. It also supports custom plug-ins in the form of Resource Manager Adapter Service, which bridge Grid service protocols with individual resource mangers such as PBS, SGE, or LSF and thus provide services not necessarily supported by grid middleware alone (*e.g.*, advance reservation).

2.6.2. *AppLeS Project*

As the pioneer in the field of scheduling from the perspective of the application in the heterogeneous environment, Application-Level Scheduler (AppLeS) project was initiated [28]. The focus of the AppLeS scheduler was to reduce the turnaround time of parallel applications in a given heterogeneous environment by using application specific agents that implemented application dependent schedulers. Each application has a personalized AppLeS agent that used parameterizable application and system specific models to predict application runtime on a given set of resources [84].

AppLeS is composed of four major subsystems, which are controlled by a separate module called the 'Coordinator' [28]:

- *The Resource Selector*: It reiterates over the set of available resources to select a set of viable ones according to a selection criteria (*e.g.,* memory availability, CPU capability, user access rights). It also uses information available from the Heterogeneous Application Template (HAT) system, which provides basic information about the application. This information is provided by the user and it includes information such as input/output file requirements, type of an application, communication patterns as well as amount of information communicated. Set of resources selected by the Resource Selector is called the active set.
- *The Planner*: It generates a system-independent plan of execution that optimizes for the user preferences. User preferences are specified by the user again in terms of the *execution constraints* (*e.g.,* resource access constraints) and *performance objectives* (*e.g.,* minimum execution time).
- *The Performance Estimator*: It uses dynamically available information (*e.g.,* NWS, GIS) to create an estimate of the application performance.
- *The Actuator*: It implements the schedule derived by the planner and interacts with resource management systems.

AppLeS code was designed so that the application scheduler agents were integrated with the application code, which made it difficult to adapt to variable applications. Each agent exploited application specific information and was not general-purpose. This limited applicability of AppLeS.

Based on the AppLeS scheduler, AppLeS Parameter Sweep Template (APST) project [85] enhanced the AppLeS project by trying to develop reusable software that was applicable to different classes of applications. The project continued to solve the two primary goals started by the AppLeS project: (1) to explore and validate the role of adaptive scheduling for the grid, and (2) apply those results to applications in production environments with the goal of improving end user experience through reduced turnaround times. However, the project primarily grew into solving the grid usability problem while using the existing AppLeS scheduler [86]. The APST focused on deployment of parameter sweep applications, which are mostly computationally intensive and have no inter-task communication requirements. Several key principles were used when developing APST: (1) ubiquitous deployment, which abstracts resource discovery, job submission and job monitoring issues for various middleware services, (2) opportunistic execution, which uses all the information available to improve application performance but requires only out-of-the-box ones (*e.g.,* ssh, scp), (3) lightweight software, which implies installation requirements are necessary only on user's local machine, (4) automation of user processes, which tries to conform the APST to have the same look and feel as the user is accustomed to when running given applications and is thus not required to learn new technologies, (5) simple interface, which uses XML technology for job specification to allows for portability, and finally, (6) resilience, which implements fault-detection and restart mechanism to cope with middleware instability.

2.6.3. *GrADS Project*

Because of the complexity of use for most of the grid middleware, which averts many potential grid users, Grid Application Development Software project (GrADS) [87]

tries to address nearly every step of application development and execution in the grid environment. The project provides advances starting with integration of grid-enabled libraries, application compilation, scheduling, staging of binaries and data, application launching, and monitoring of application execution. In this discussion, we focus on the scheduler developed as part of the GrADS project because it is most relevant to the topic of this dissertation.

The scheduler component of GrADS project was inherited from the AppLeS project and was modified and extended to try to accomplish key goals of the GrADS project as a whole [88]. The modified scheduler uses specialized compiler and grid-enabled libraries derived from the GrADS project to record and retrieve application characteristics [29]. As stated earlier (Section 2.6.2), AppLeS code was built using poor cohesion so, as part of the GrADS project, the implementation was separated to support the search procedure and the application-specific components as independent code modules. The application specific component collects information about the application itself and is the only component that is application dependent. It consists of two modules: *performance model* (analytic metric for predicting application turnaround time on a given set of resources) and the *mapper* (performs mapping from logical names to physical resources) [88]. At the same time, the search procedure is completely independent and it performs initial resource selection based on the information available from the NWS and GIS with no application-specific components. It generates a pool of available resources that might be of interest to applications in general. At the next stage, the two modules are evaluated together and resource set with predicted minimum application turnaround time is selected.

This scheduler was primarily focused around scheduling loosely synchronous applications on resources supporting high-performance networks. It was not designed to use dedicated machines controlled by local schedulers and thus does not support activities such as advance reservation.

2.6.4. *Condor-G*

Condor [89] is a framework and a resource manager for coarse-grained, compute intensive applications. It provides functionalities of a typical job manager system such as job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management [90]. Rather than being yet another job manager, Condor is a scavenger of CPU cycles and allows for numerous individual resources to be joined into Condor pools, thus aggregating and provisioning otherwise idle resources. It monitors registered resources and combines high-throughput computing and opportunistic computing by scheduling jobs onto otherwise wasted CPU power. Three points distinguish Condor from other distributed job submission technologies (*e.g.*, CORBA, RMI, Locus, Grapevine): (1) *ClassAds* [91, 92] used to expressively match resource requests to available resources, (2) *job checkpointing and migration* allowing certain jobs to move from one resource to another, thus supporting fault-tolerance, and finally (3) *remote system calls,* which builds on the idea of a sandbox and relives the user from having to manually transfer input and output files to and from execution resources.

Condor-G [44] is a culmination of the technologies employed by the GT and the Condor project. It combines security and resource-management protocols from the GT to allow for cross-domain communication and resource-management methods for inter-domain manipulation found in Condor, thus allowing distributed resources to be viewed

as local ones. Condor-G can be used as an interface on top of the GT and it provides the user with efficient insight into job queues, job lifecycle logs, failure handling, and management of input and output files otherwise not directly supported by the GT. Condor-G does not directly support selection of resources for user jobs but relies on user-supplied information. If combined with the techniques of Gliding In [90], it can provide the user with the nearly full capabilities of Condor, the corresponding Matchmaker [91], and thus a personalized, distributed Condor pool. As a final note on Condor-G, it should be noted that Condor-G does not implement a metascheduler; it provides a uniform access to various underlying schedulers (*e.g.,* Condor), but as a standalone application it only provides a standardized interface for job submission and fault tolerance.

Many of the Condor and Condor-G techniques depend on the two components: *ClassAds* and the *Matchmaker*. *ClassAds* are a schema-free set of uniquely named expressions that provide a mapping from the attribute names to the given expressions. They are used to describe jobs and resources and then provide an effective mechanism for evaluating two *ClassAds* against each other. *ClassAds* are largely a Condor representation of RSL and JSDL. At the same time, the *Matchmaker* is based on the idea of *ClassAds* and is the mechanism performing the comparison and evaluation of user requirements and resource owner specifications. The process of matchmaking requires four steps: (1) through the model of classified advertisement, resources and agents advertise themselves, (2) matchmaking takes place next, it involves creation of pairs that satisfy requirements and constraints, (3) next, the matchmaker introduces the two parties satisfying the match, and finally, (4) the process of claiming takes place where the agent and the resource

establish contact and possibly exchange any further requirements and conditions before commencing the job.

2.6.5. *Nimrod/G*

Nimrod [93], inspired by the Condor Project, is a tool that works with large parametric experiment applications. Parametric experiments are applications requiring numerous runs with different parameter values, providing an opportunity to run the application in parallel. Nimrod provides a resource manager interface in a distributed environment for execution of such applications and thus parallelizes application execution. The application's code does not need to be modified in any way with Nimrod providing functionality such as fault-tolerance and result aggregation for the submitted set of parametric values. The user is required to submit a task plan for execution using the format of the Nimrod-specific declarative parametric modeling language of given parameters and Nimrod performs the rest.

Nimrod handles small-scale resources where they belong to the same administrative domain and are mostly homogeneous in terms of resource managers, CPU, user policies and access cost. Nimrod/G [94], on the other hand, was created to maintain functionality of Nimrod but in a heterogeneous and dynamic grid environment. It uses GT protocols and services to support remote resource discovery and job submission. Even though Nimrod/G is composed of several components (*i.e.,* Client and User Station, Parametric Engine, Scheduler, Dispatcher, and Job-Wrapper), we focus here on *Parametric Engine* and *Scheduler* components as being most relevant.

Parametric engine is an agent acting as the central component in charge of parameterization of the experiment, job creation, job status monitoring, interacting with

clients, scheduler and the dispatcher. It provides the sandbox for the user job in terms of interpreting the declarative task language, maintaining the state of the job as a whole and handing expansion and contraction of job size due to the errors and failures.

The *scheduler* component is the key contribution of Nimrod/G because it introduces economical aspect into resource selection. Beyond only using the information available through GIS, it attains cost associated with resources through services similar to GIS (not functional yet). When evaluating resources, it uses one of the three scheduling algorithms: time minimization, cost minimization, and cost-time optimization [30, 95]. Time minimization schedules with the goal of minimizing the time but staying in the assigned budget limits. Cost minimization algorithm produces results by the given deadline, but focuses on minimizing the cost. Cost-time optimization algorithm stays within budget and deadline limits and tries to minimize turnaround time when possible. The optimization procedure is done through a system of resource cost sorting and exploiting all the resources with comparable cost and performance. If the deadline and/or cost restraints cannot be satisfactorily completed, the negotiation phase is entered where either the advance reservation is employed or the user is required to change the deadline and/or cost [96].

2.6.6. *Gridbus Broker*

Gridbus Broker [97] steps into the realm of data grids [98] by not only trying to find computationally adequate resource for user's job, but also considering the discovery of suitable data sources and making optimal matching between the two entities with respect to the data locality and capabilities of computational resources. The broker also handles

the process of file staging, job submission and monitoring, as well as combining and presentation of the results.

The scheduling algorithm of Gridbus broker, rooted in the event-based Round-Robin approach, extends the scheduler component from Nimrod/G and the economical aspects employed there. As just mentioned, rather than optimizing user parameters for computational jobs only, the broker considers the cost of accessing remote data repositories and tries to optimize for associated file transfers. The idea of scheduling computation on resources close to the resources holding the required data comes from [99]. It is observed that once computational economy [100] aspects are introduced into the scheduling of applications, there will be more tradeoffs users and resource owners will be willing to make because of the associated cost. Gridbus broker tries to minimize this. The broker also extends Nimrod/G's parametric language by supporting dynamic parameters in form of regular expressions allowing the user to leave parameter evaluation to the runtime environment. In order to select the appropriate compute resource, the scheduling algorithm considers capability and performance of a given resource, bandwidth available from the compute resource to the data resource and the cost of the data transfer.

2.6.7. *Michigan Advanced Resource Scheduler (MARS)*

From the University of Michigan, as part of a the Michigan Advanced Resource Scheduler (MARS) project [101], comes a scheduler that supports job prioritization and on-demand task scheduling. Even though the software for this project is not available for download, it is an example of an on-demand scheduler that supports critical-priority resource selection and job launching. The scheduler uses resource utilization prediction

based on a low-pass filter [102] and passes the information to either of the two supported scheduling algorithms: the minimum complete time (MCT) algorithm [103, 104] or the evolutionary algorithm.

The MCT algorithm is based on the idea of tasks being passed to available resources and estimated complete time is recorded. The scheduler selects the resource with the minimum execution time. The two version of the algorithm exist, the *online*, in which execution times are computed at run time, and the *offline*, where the execution times are computed a priori. Even though MARS prefers the online version of MCT, adoption of the offline version would allow the information to be collected over time and be provided as the contribution to the scheduling algorithm giving insight into actual application performance. This subject is further discussed in the Approach section of this document.

The evolutionary algorithm seems to be most effective when dealing with a large number of tasks. It uses a genetic algorithm (GA) to optimize resource assignment and order of task submission. Currently, it uses user-provided parameters dealing with maximum/average time to wait when evaluating resources, but it also allows users to specify their own fitness function and extension of supplied parameters.

2.6.8. *GridWay*

GridWay [105] is a framework that works on top of an existing GT installation and uses GT services for most of its operations. It is considered to be part of the grid ecosystem [106] because it provides a slim middleware layer on top of the GT. The goal of the project is to support the “submit and forget” ideology where the user is abstracted from the grid middleware details by supporting easier and more efficient job execution on the grid. GridWay allows for unattended, reliable, and efficient execution of a range of

applications through command line and programmable interfaces. GridWay provides implementation of Distributed Resource Management Application API (DRMAA) standard [107] and thus supports API level interaction with the given scheduler and job management infrastructure, significantly simplifying the process of grid application development [108].

The GridWay scheduling algorithm supports dynamic and adoptive scheduling with migration support as well as job and resource prioritization policies [109]. Supported job prioritization policies include fixed, fair-share, deadline and waiting-time [110]. Fixed job prioritization policy allows specifying fixed priority for each job or user group and thus distinguishing individuals and corresponding jobs within the grid. Share policy allows a ratio to be established for either individual user or user group and enable job submitted to be submitted at a more or less frequent rate (*e.g.*, target 2:5 ratio for job submissions). Waiting-time policy disables starvation of low priority jobs in linear time fashion. Finally, deadline policy specifies support for submission of jobs by given deadline through job priority increase.

At the same time, a built-in support for resource prioritization exists that consists of fixed property policy, rank policy, usage policy, and failure rate policy. Fixed policy enables groups of resources to be assigned priority values and thus support priority scheduling on those resources. Rank policy allows specification of rank for individual resources. Usage policy is used in conjunction with historical resource execution to include statistics over a specified period of time and or for the last job submission individually. The information obtained is used to predict runtime of new user job based

on previous execution, transfer, and queue wait times. Failure rate policy provided support to prevent thrashing of resource(s) that keeps failing.

One of the interesting contributions from GridWay, and supported by the provided scheduler, is the model of a grid-aware and self-adapting application. Requirements for converting an application into a grid-aware one require it to be aware of the dynamic environment it executes in. This is achieved through specification of a *requirement expression* (automatically specified at runtime as application specification requirements that must be met by execution resources), *ranking expression* (required to dynamically assign each resource a rank used to prioritize it), and a *performance profile* (used to keep up with application performance activity to detect performance slowdown and initiate migration). Once the application conforms to this model, it is able to register performance degradation, communicate its list of preferred resources to the scheduler and request a migration. Even though it is not necessary to modify application source code to use GridWay scheduler, by adopting the grid-application model, one is able to achieve significant performance gains and thus get the most out of resources [111].

GridWay represents the current state-of-the-art job management tool where the entire job submission process is automatically handled on user's behalf (*i.e.*, resource selection, input data transfer, resource acquisition, job monitoring, job cleanup, and result retrieval). Beyond being used as a comprehensive scheduler and a job manager, GridWay can also be used as a job submission tool where listed job management procedures are automatically handled by GridWay but controlled from another tool. This is promoted through the support for programmatic interface (*i.e.*, DRMAA) enabling custom-built

tools (*e.g.*, metaschedulers) to reuse low-level functionality delivered by GridWay and focus on developing higher level logic.

2.6.9. *Workflows and Multi-objective Scheduling*

With complex analyses of various data, computations performed on the grid, and HPC resources in general, are often constructed as workflows [112]. A workflow can be seen as a formalization of the scientific analysis where individual analysis components that need to be executed are well structured and accompanied with the required input data at each step. Often times, a workflow is composed of several distinct stages where each stage exhibits a certain level of parallelism but then results need to be aggregated between such workflow stages allowing the overall workflow execution to continue. Depending on the formulation and dependencies that exist between individual components, workflows can be characterized by a balanced structure or an unbalanced structure. Figure 12 depicts two examples of the possible workflow structures.

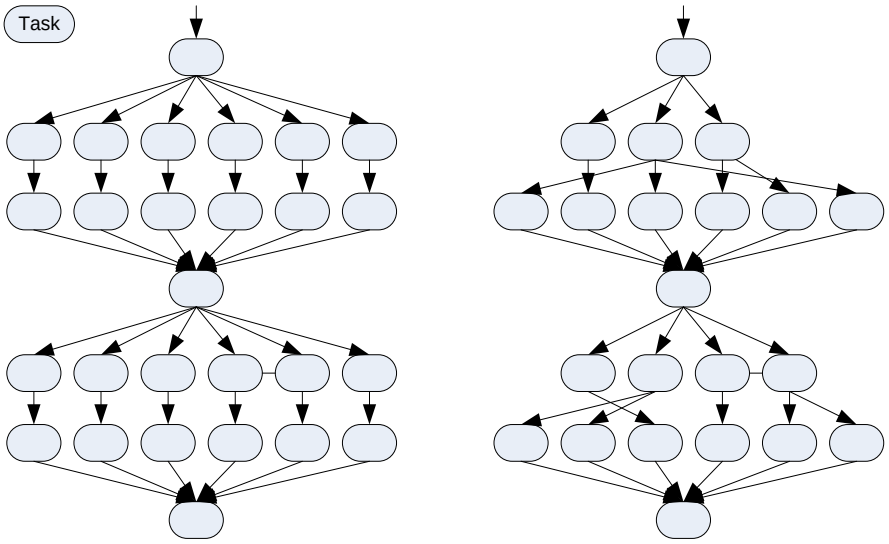


Figure 12. a) Balanced structure workflow, and b) unbalanced structure workflow

A typical goal of grid job scheduling is minimizing turnaround time of a job. However, rather than minimizing such a single objective (*i.e.*, application runtime), it

may be appealing to minimize multiple such objectives (*e.g.*, time and cost or time and accuracy). However, realizing such objectives often calls for conflicting actions to be performed. For example, minimizing runtime of a job may require acquisition of all of the available resources. However, such action will cause increase in job cost. Similarly, minimizing cost would call for using as few of the cheapest resources as possible. It is obvious that such decision is in conflict with the goal of minimizing job runtime. In such a case, neither of the objectives can be minimized as they would if they were the sole objective but rather a compromise needs to be reached. This is the idea behind multi-objective scheduling [113, 114]. In multi-objective scheduling, it is not possible to find a single, optimal solution that maximizes/minimizes all of the included factors. Instead, a set of alternatives exist providing tradeoffs with respect to the multiple objectives.

In the context of grid workflow scheduling, several projects have considered applying multi-objective scheduling techniques to computational workflows [115, 116, 117]. In [115], a workflow planning method is proposed that investigates the tradeoffs between multiple objectives (namely, cost and time) when a workflow is scheduled across grid resources. The proposed solution performs workflow planning that generates a set of alternative tradeoff solutions within user specified budget and runtime constraints. Although the general idea resembles work presented in this dissertation, realization of the approach falls short because the work focuses on implementing and comparing performance of three well-known algorithms for workflow execution planning and many of the details (discussed in next section) required to perform fully automated and application-oriented scheduling are not considered nor incorporated.

Work performed in [116] presents a formulation of the workflow planning problem as sequential cooperative games. In this case, the workflow is defined as containing a number of homogeneous and concurrent activities whose structure should be manipulated in order to deliver a more optimal execution organization. Therefore, the aim of this work is toward structural reorganization and composition of a workflow whose mapping will yield a time and cost optimal solution. However, mapping of the individual workflow tasks to heterogeneous grid resources is not considered.

Beyond looking at time and cost as the only two optimization objectives, work done by Wieczorek et al. [117] presents a solution where a choice of two objectives can easily be chosen immediately preceding the scheduling process. Typical multi-criteria scheduling approaches require the user to specify weights for either of the two constraints, thus imposing preference as to the optimality of the solution. On the other hand, work introduced by Wieczorek et al. provides a general bi-criteria scheduling heuristic that allows for a range of criteria and weights to be specified when planning a workflow execution.

2.6.10. *Critique*

Grid computing is a popular and a likely candidate to become the next-generation high performance distributed computing platform. However, the goal of providing ubiquitous access to distributed, HPC resources that are shared between multiple organizations through virtualization and aggregation is only as efficient as its overall perception by end users. This means that access to such distributed and dynamic infrastructure should be brought to the level where an individual user is not only comfortable in their environment but their demands and requirements are met. In order

for this to be achieved, the system should be customizable from user's perspective and also have a deep understanding of the existing relationships and dependencies between user jobs (*i.e.,* applications) and resources across the infrastructure so that the user demands can be more adequately met.

Projects focusing on scheduling of application jobs across the grid, as presented throughout Section 2.6, represent a thorough overview of the directions and accomplishments in this field. Conclusions that can be drawn from these experiences point in the direction that application-oriented scheduling approach [28, 88] to grid job scheduling delivers a higher level of performance regarding application execution characteristics, such as higher accuracy, reduced runtime, when compared to simpler and more generalized approaches [109]. Consequently, the approach adopted throughout this dissertation embraces the conclusions behind application-oriented scheduling and build on the lessons learnt. Notions such as application level scheduling [28], decoupling of the application and the scheduling action [88], simplicity of use [118] and economic impacts [119] are all considered as tools and features that have been proven as viable approaches and are supported throughout the design of here presented work. A major novelty of this work is the generality with which it incorporates and embraces relevant features, in turn enabling the concepts behind application-oriented scheduling to be realized. In order to enable such functionality, the presented work does not employ any single functionality, or a set of functionalities, but rather undertakes a meta-approach enabling a variable number of features to be incorporated on as needed basis (details are provided and elaborated on in the Chapter 3 where overall Approach is described). As a result, existing and upcoming technologies and tools that generate data pertinent to application-oriented

scheduling can be effectively incorporated into the overall architecture enabling the proposed work to transcend current trends.

In spite of many attempts (as discussed in preceding sections) that have been made at solving the infrastructure management and access problem (*i.e.,* scheduling), to date, there is still an existing and a recognized barrier between the core infrastructure and the users [69]. This is not to say such projects have failed; on the contrary, much progress has been made but, as the case often is with software and intangible tools in general, the demand and requirements imposed by users tend to surpass available functionality [120]. In particular, when it comes to the grid job scheduling and user perception, current projects tend to focus on automating much of the process for the user with the goal of absolutely simplifying or abstracting the use of the underlying infrastructure (*e.g.,* assume runtime minimization is the only objective for all users). While overall such an approach is welcomed, it implies reduction in the control a user has over their environment. In order to improve existing behavior and the approach to the grid job scheduling, the aim would be to automate the tedious and demanding tasks, which a user does not care about or cannot perform effectively because of the problem scope. At the same time, no assumptions of user's intentions should be made (*e.g.,* minimize job runtime). Instead, the user should be guided through the process of job submission in terms applicable to the user. Such an approach combines the advances of automation a computer is capable of, but it also permits the user to influence the direction and final destination of the process. In other words, it allows for user-level customization of the default system behavior. Adoption of such an approach permits a high-level of flexibility for the user and can thus satisfy more of the individualistic demands imposed by various users. At that point, the

system becomes flexible, as opposed to being predetermined or rigid, and can thus evolve with the evolution of the principles guiding the general progress.

The approach presented differs significantly from the existing approaches in that application-oriented scheduling is supported from the 'outside' (*i.e.*, results from other tools can be readily used to guide the scheduling process) and that a user-level customization is not only supported but encouraged. In order to make these goals feasible, rather than providing a concrete implementation and a solution, presented approach offers a higher level architecture that, when implemented, automatically enabled desired functionality.

2.7. Application Performance Modeling and Monitoring

Because a significant part of the work discussed in this dissertation is devoted to application specific parameters and trying to collect and interrelate that information, this section explains current projects that deal with these topics and how such information can be leveraged to improve application performance characteristics in grid environments.

2.7.1. *Using Historical Information to Predict Application Run Times*

Because of the limited amount of information available to schedulers in the grid, additional tools are needed to complement already existing resource selection tools. A prominent approach seems to come from the area of tracking historical application information and applying it to predict future runtimes of the same and similar applications. Downey [121] and Gibbons [122] both used the idea of applying templates to individual jobs to classify corresponding applications as similar. Templates can be seen as application descriptors from the scheduler's perspective and can include information such as username submitting the job, time of job submission, number of

CPUs requested for execution, queue selection and so on. Downey used prediction modeling to predict how long will the job request wait in the queue before starting its execution, while Gibbons explored the capabilities of improving scheduling algorithms by predicting application runtimes. Although both approaches seemed promising, the results did not show beneficial because the prediction times often exceeded job execution times. Through a new study, which was built on similar concepts, Smith [123] managed to achieve desired results of predicting application execution times by using more carefully selected set of templates. While developing the templates, Smith manipulated choice of application characteristics incorporated into creating the set of templates and added a choice of analysis and selection algorithms used to find the best template set.

Some of the major contributions found in Smith's work, seen as modifications from the previous approaches, include answers to two major questions: (1) what defines similar applications, and (2) how are the predictions generated? Answer to the first question is found in the extension of the previous idea of creating categories and thus templates to which an application is assigned. The idea is extended through the notion of a continuous parameter for job options, such as the number of requested nodes, rather than exclusively using discrete parameters. The set of pieces composing a template is a variable number that is obtained by executing one of the two algorithms (greedy algorithm or a genetic algorithm search). These algorithms are each fed existing resource workloads and they proceed to consider varying template sets while keeping only template sets selected by either *mean* or *linear regression* methods. The *mean* approach considers the mean time of all applications in a given category while the *linear regression* approach considers additional parameters, such as the number of nodes, and

tries to compute coefficients *a* and *b* for the equation R = *a*N + *b* (where R is the runtime and N is the number of nodes). Regardless of the method employed, the template set is selected as valid only if the prediction falls within 90 percent of the confidence interval (confidence interval is an interval centered on the run-time prediction within which the actual run-time is expected to appear some percentage of time).

The above described approach has produced results with prediction errors in the 50 percent range, which is a significant improvement over previous approaches, and confirms the point that historical application run-time information can be effectively used to predict future execution times.

2.7.2. *Prophesy Performance Database*

Based on the idea and an observation that resources used to execute given jobs influence the performance of the application for the given run, a Prophesy project [124] was started with a goal of capturing such information and provide insight into application and resource dependencies. Prophesy infrastructure is based on a relational database used to record application performance data that is later used to develop models for application execution on available resources. Derived models can be used to discover most efficient implementation of individual functions for a given system, to relate input data to application performance, or to choose the most appropriate resource for execution. Information collection is based on community effort and is supported at the API level and through a web-based interface [125].

There are three major components making up Prophesy infrastructure, namely data collection, data analysis, and relational databases. Databases employed consist of *template database*, used to store application templates and identify appropriate model

optimization technique, *performance database*, used to store performance information dictated by user selected granularity level, and *systems database*, focusing on application execution parameter in terms of inputs, runtime, resources and instructions on generating the executable [126]. Data collection component's goal is automatic code instrumentation at the low level of blocks, functions, and procedures and automatic storage of this data in performance database. Granularity of code instrumentation can be adjusted by end user. Once collected, the data is analyzed through data analysis component aimed at developing prediction models for different sets of resources allowing users to investigate other options and possibilities within their application, such as performance on varying kernels, interactions with different applications and across differing resources. Models are developed from historical application data and resource information by applying one of two primary optimization methods: curve fitting and parameterization method. In order to generate the model, curve fitting uses one of optimization techniques available in Matlab, such as least squares, and uses application computational complexity information and historical application runtime data as available in the Prophesy database. Limitations of curve fitting model limit the applicability of the generated model to application scalability vs. resource configuration comparison. While curve fitting is a fully automated technique, parameterization method requires a combination of manual analysis and resource performance statistics. With assumption that only small, function critical code segments need to be analyzed, analytical equations are developed where generated equations, and thus the model, are a function of input variables. This, combined with resource information available in the Prophesy database, allows an application to be modeled across different resource configurations [127].

Contributions and applicability of Prophesy infrastructure have been verified through an array of examples and test cases with real-world applications (*e.g.,* Parallel Multiblock Lattice Boltzmann Application [128], 3D Parallel Volume Rendering Application [129]). In conclusion, the key objective of the Prophesy infrastructure is identification of the best implementation of individual functions for a given resource. This is accomplished using historical, unprocessed data to generate application models and analysis for variable resource configurations to establish speedup, scalability, communication requirements as well as application coherency.

2.7.3. *GridBench*

Somewhat similar to the Prophesy infrastructure, GridBench [16, 26] is another project dealing with matching of resources and applications through application performance monitoring. Unlike Prophesy, which focuses on applications and uses historical application performance data to predict and process various application configurations, GridBench focuses on providing a core set of benchmarks that characterize grid resources. Use of such benchmarks, as is commonly done in comparison of individual resources even without grid concepts, allows prediction of performance and scalability of compositions of applications on desired systems. The proposed framework supports collecting, archiving, and publishing of collected data. The goal of GridBench is to provide a collection of kernels representative of applications and application categories that can be used to benchmark and characterize components that affect performance of applications and resources, allowing comparisons to be made.

GridBench framework provides five scenarios where it can be applied: benchmarking of grid resources in isolation and capturing elements such as CPU speed or

communication speeds; benchmarking of VOs that encompass multiple grid resources; benchmarking performance of middleware; benchmarking of resources by a representative application, or a subset of one, where given resource can be characterized using more application-specific details; and finally, as a test tool for developing tools to validate assumptions made during the development. These benchmarking functionalities are realized through creation of three levels of benchmarks: micro-benchmarks (capture basic performance such as CPU performance, I/O speed, memory), micro-kernels (mathematically well understood synthetic based codes developed to stress test individual aspects of grid performance), and application benchmarks (derived from applications deployed on the grid consisting of many micro-kernels composed into a workflow representative of a given application).

Implementation of GridBench framework is a composition of the following components: RSL compiler, which creates job submission documents from benchmarks; orchestrator, which manages all other components; benchmark components or benchmark executables; monitoring component in charge of information collection; archive database; information provider for information publishing; benchmark definition and benchmark browser in charge of submission and retrieval of benchmarks, respectively. Such layered implementation allows partial components not only to be developed independently, but also supports modular deployment and addition of outside components. GridBench is a promising tool whose applicability is obvious in the community dealing with application performance storage and modeling.

2.7.4. *GrapBench*

Positioned between the Prophesy benchmark suite and GridBench, GrapBench [130] is a tool that focuses on application benchmarks while incorporating the problem size and the resource size into the benchmark results. Through an XML based definition of a desired benchmark, which includes executable specification, set of problem sizes, pre-execution requirements, and a set of available resources, GrapBench enables targeted and controlled execution of application benchmarks. Beyond only executing the benchmarks, GrapBench archives them in a local repository for later retrieval and analysis. Lastly, collected results can be visualized through a GUI enabling performance analysis with a focus on application performance and scalability or benchmarking of grid site as a whole. The analysis component includes ability to predict application's behavior across various input data sizes and resources by extrapolation. However, such an approach assumes consistent application scalability and relational performance of an application across various resources. By enabling a user to compose an application benchmark, GrapBench allows for flexibility regarding the type of benchmarks used and thus provides a benchmarking framework that can be customized on as needed basis. By enabling variable input data to be included into the benchmarks, impact of input data on application performance characteristics can be studied.

2.7.5. *BioPerf*

Building on the notions of application-level benchmarking and information such benchmarks can deliver regarding relationships between an application and a resource, BioPerf [131] is a tools that focuses on benchmarking bioinformatics applications. With the exponential growth of genomic data caused by recent improvements in sequencing

technologies, the amount of data to be searched through has grown significantly and has thus led to higher demand for HPC resources in the bioinformatics community [132]. To that extent, in order to provide users a general approach to benchmarking frequently used applications across available resources, BioPerf is a benchmark suite of representative bioinformatics applications and sample input data sets that can easily be invoked and thus provide application-specific benchmark of a selected resource. Through availability of such a tool, newly arriving architectures can be quickly and consistently benchmarked resulting in immediate comparison to other available resources. BioPerf includes codes and input data sets for ten popular bioinformatics applications, one of which is BLAST, and thus represents a useful tool in the context of majority of experimental work performed as part of this dissertation. One major drawback of BioPerf, as relevant to this work, is that it does not allow easy modification of application invocation parameters and thus does not allow for benchmarking of influence that various parameters have on application runtime characteristics.

2.7.6. *STAPL*

The underlying differences in hardware found in a typical grid causes application performance to vary from one resource to another. For many applications, the underlying algorithm comes in many flavors but performs the same functionality (*e.g.*, sorting, matrix multiplication). Selecting which algorithm to use, depending on the system specifications, can significantly improve the execution time of the application or reduce application requirements in terms of communication cost and/or compute requirements [14]. All of such choices should, however, be hidden from the end user because the user

is primarily concerned with the correct execution of the application as s whole rather than such low-level details.

Standard Template Adaptive Parallel Library (STAPL) [133] is a framework for parallel C++ code which provides a parallel programmer with a shared object view of the data space. Through the internal mechanism, STAPL ensures automatic translation of one space to another for objects distributed across a possibly distributed address space, thus providing the user with a unified data space. As part of the STAPL, a framework has been developed for performing an adaptive algorithm selection used to select the most appropriate parallel algorithm for the given architecture [134]. The algorithm selection is based on the information about the underlying system architecture and is done at runtime. STAPL framework focuses on selection between multiple equivalent parallel algorithms and tries to optimize on the granularity of data distribution and inter-process communication.

When STAPL is installed, the framework collects statically available information from the standard header files and system calls about the underlying system and the given environment (*e.g.*, CPU speed, number of CPUs, memory size). This information is stored in a data repository along with algorithm performance information obtained from installation benchmark runs. Machine learning tools are used next to analyze this data and store it for retrieval at application runtime. The machine learning is decomposed into two categories: the *general empirical model*, which uses information from the previous runs to predict optimal parameter values for the future runs, and *analytical models*, which are manually generated models (usually by the application developer) that provide insight into a given algorithm to a greater detail (*e.g.*, communication formulae) and can possibly

result in greater accuracy during algorithm selection. Currently, STAPL framework supports only the empirical model. As far as analyzing collected data, the framework uses three learner methods, the most promising, decision tree learner [135], a back propagation neural network [136] with two internal layers, and a Bayes naïve classifier [136]. Selection of which learner method to use for algorithm selection is done on per-algorithm basis and it is based on the examined accuracy of the learner's ability to model the problem (this is evaluated upon the installation and completion of the algorithm benchmarks).

2.7.7. *Application Skeletons*

All the approaches attempting to predict application behavior discussed so far do so based on information available about the current resource status and/or experiences from previous runs of an application. Another approach with a similar goal in mind is prediction of application execution time based on application performance skeletons. An application performance skeleton is a short running program that mimics the execution time and cost of the real application [137]. The basic idea behind performance skeletons is that skeletons execute operations representative of the real application in terms of CPU usage, communication patterns and cost, memory access patterns, and the similar but execute up to 1,000 times faster than the real application.

From the University of Houston, comes a framework and the initial implementation of the application that analyzes applications and automatically generates code for the performance skeletons ready to be executed on any given resource [137]. Current version of the application deals with MPI applications implemented in C programming language only. The process of skeleton generation consists of three steps. First, the application

execution trace is recorded registering CPU usage and message exchanges. Then the signature of the application is created by analyzing the trace, manipulating it, and compressing it. Finally, the performance skeleton code is generated that can be executed as the real application but will run by a factor *K* times faster. Testing has shown an average time a skeleton needs to run is approximately one second and a reliable execution signature of the application can be obtained, thus showing plausible acceptance into the grid application scheduling.

2.7.8. *Critique*

In the general context of the grid, it has been shown that individual heterogeneous systems provide heterogeneous performance features that map more suitably to one class of applications as opposed to another (*e.g.*, [16, 17]). It has also been shown that performance of any individual application is heavily dependent on the underlying system characteristics including parameters selected during the job submission process [138]. Such observations can be used to recognize existence of a relationship between an application and a resource. Not observing and understanding such relationships can result in excessive job runtimes, wasteful user allocation time, decreased resource utilization, and reduced resource throughput. It is thus desirable to detect existing dependencies and relationships between individual applications, resources, and job parameters so that the observed characteristics can be exploited during future job allocations.

To that extent, there is a need to capture and provide application- and resource-specific data so it can be used during the scheduling process and a match between application requirements, resource capabilities, and user requests can effectively be made. Generic and standardized benchmark suites, such as LINPACK [139] and SPEC

(Standard Performance Evaluation Corporation) [140], have been created with the idea of benchmarking individual workstations and providing meaningful comparisons between performances of those. Such approach worked well in more localized and traditional settings where major units of performance were resource speed, throughput and bandwidth. In the context of the grid, rather than focusing on a number of jobs a resource or a group of resources can process, the focus is on the individual user and job turnaround time; this is a characteristic of a service oriented environment. Because standardized benchmarks focus on benchmarking a resource as opposed to an application, they often fail in a number of ways to capture intrinsic details and requirements imposed by individual applications as they are executed across various resources [130].

Individualized solutions and projects that focus on application-specific benchmarks and performance analysis, as described and elaborated on throughout Section 2.7, provide an excellent direction for capturing necessary application- and resource-specific data that can be used during metascheduling. However, such solutions provide standalone instances targeted at either specific class of applications or a specific characteristic across applications. Because of such individualistic and uncoordinated approach, a general tool, such as a metascheduler, that attempts to consume application- and resource-specific data to deliver application-oriented services must aggregate desired data from multiple sources. Superficially, an approach that attempts to manually combine data from such multiple directions complicates its development and functionality because of the differing formats the data is stored in and inconsistency of the data collected by individual application performance profiling tools. More substantially, by only using the data from individual and unrelated projects on an ad-hoc basis, there is a need to repeatedly perform

analysis of application execution characteristics by each instance of a tool consuming available data (*i.e.,* each metascheduler being developed or even just deployed). In other words, each tool that tries to analyze and understand behavior or performance characteristics of an application must do so individually and irrespective of whether such analysis has already been performed by another tool. This is because the application analysis information is tied and stored locally and individually by the user of the data (*i.e.,* the metascheduler or corresponding analysis tool) as opposed to being tied and stored with the application itself and thus distributed with the application itself. With the availability of only individualized application performance collection tools, there is no standardized set of tools that a VO can adopt and publish. Instead, if a VO could publicize availability and support of application-specific data in a standardized format, thus ensuring availability of necessary data to metaschedulers, such metaschedulers could become more generally deployed. Moreover, because such a standardized set of data would be available to the metaschedulers at the VO level, the metaschedulers could immediately offer support for the concepts behind application-oriented metascheduling at the level of a VO.

In conclusion, although current approaches present viable solutions for application-level and application- and resource-specific performance data collection, the solutions they provide are defined in terms of actions they perform and data they collect. Because such tools are developed on a stand-alone basis with no direct ties between them, consumers (*i.e.,* metaschedulers) of such tools use them on case-by-case basis. Such an approach results in tools consuming the data to be forced to aggregate desired data from multiple, independent sources and create custom solutions that can be applied in

scheduling of application jobs. More importantly, it is likely that because individual instances of tools consuming available data have to compute the data on their own, the available data is being analyzed repeatedly by individual tools and tool instances. This obviously results in waste of effort and computational power.

To alleviate such repeated efforts and enable a higher level of service to relevant tools, there is a need to enable capturing, collecting and sharing of pertinent data on a more generalized scale (*e.g.*, at the level of a VO). Moreover, rather than focusing on storing only low-level application performance data, it would be beneficial if analysis results and conclusions about application performance characteristics, which can be easily understood and immediately consumed, could also be collected and distributed (*e.g.*, this application scales efficiently up to 128 processors and the processor assignment number must be a power of 2). Such improvements enable generalized approach to application-oriented scheduling while minimizing required effort and are part of the topics covered by this dissertation.

2.8. Embarrassingly Parallel (EP) Application Domain

The overall approach and discussion of this work, as outlined in Chapter 1 and elaborated in Chapter 3, presents this research as applicable across the field of grid metascheduling; topics dealing with application- and user-oriented grid metascheduling are detailed and explored (see Section 3.5 and Section 3.6). Within this scope, the focus of presented work is on metascheduling Embarrassingly Parallel (EP) class of applications. This section builds on the classification of grid applications presented in Section 2.3.1 and provided background details regarding the EP class of applications (Section 2.8.2). In addition to the general discussion about EP class of applications,

observations regarding metascheduling considerations for the class of EP applications are presented (Section 2.8.3). Conclusions of the considerations and analysis presented in this section are incorporated into the work described throughout Chapter 4.

2.8.1. *Selection of the EP Application Domain*

Based on the grid application classification categorization presented in Section 2.3.1, with the same general approach to resource allocation, the EP class of applications encompasses categories two through four [14]; namely, applications whose parallel execution is supported by the application model without any communication between individual instances of the application. This class of applications was selected as the focus of this dissertation. The selection was made based on the following three reasons:

1. From the early days of the grid computing, EP class of applications has been thought of as the 'killer' application for the grid [96]. This means that the grid offers an extremely suitable platform for realizing the type of execution environment this class of applications requires.
2. A recent study [141] shows that the EP class of applications is the most widely deployed class of applications on existing national and international grids. Therefore, this dissertation's focus and advances within this class of applications offers the widest benefit to the global community.
3. Once the metascheduling model for the EP class of applications is understood, other application categories can be modeled within the EP class. Initial discussion on how to accomplish this is provided in Section 7.1.4.

2.8.2. *EP Applications*

EP applications are characterized by independent, coarsely grained, and indivisible tasks. The goal of EP applications is to introduce parallelism into application execution without any application code modifications or associated cost. This parallelism is realized through multiple invocations of the same application, where each instance is invoked using a different input data set (see Figure 13). The number of tasks being instantiated can range greatly, and can include only a few instances or several hundred instances with each instance executing from several seconds to minutes to many hours. The end-result is speedup of application's execution that is limited only by the size of input data and the number of resources available. Suitability of the EP class of applications for grid environments is a result of the ability of EP applications to consume large numbers of resources as well as impose minimal requirements and dependencies on network status and offer high tolerance for partial failure.

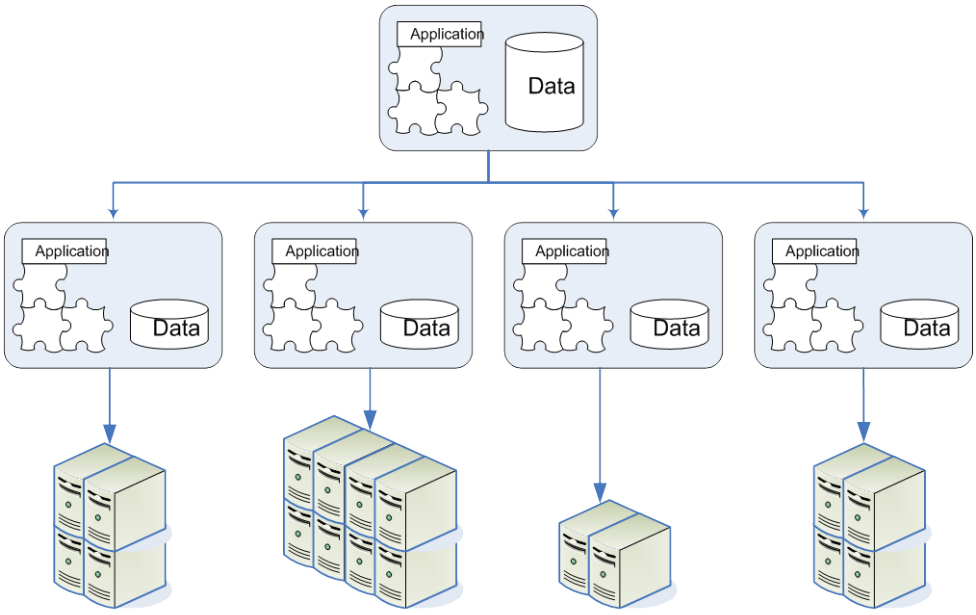


Figure 13. Execution model for embarrassingly parallel applications

2.8.3. *EP Application Metascheduling Considerations*

Although executions of individual instances of EP application jobs are considered independent and should therefore be easy to schedule, systematic execution of EP jobs involves considerable technical difficulties when executed in grid environments [14]. The challenges include dynamic resource availability, initial data distribution, global load balancing, and failure handling. If such considerations and dependencies are not acknowledged or understood, many of the potential benefits may simply be invalidated.

Understanding an application's relationship to potentially numerous grid resources may seem as a farfetched goal requiring disproportionate effort for the benefit received. If, however, the process of metascheduling is decomposed, the process can be classified into several main components, lending themselves as a set of considerations for deepening the understanding of the application-resource relationship. Furthermore, depending on the desired results (*e.g.*, quick metascheduler development, high accuracy of metascheduling actions), the level of understanding of the existing relationship can range from rudimentary, where just a small number of the most obvious application characteristics are recognized, up to a well-understood and well-developed mathematical model for an application's behavior that can later be mapped to individual resources. With the aim of providing a decomposition of the scheduling problem, the following can be seen as the most general set of considerations that should be kept in mind when metascheduling and executing the EP class of applications across a grid environment:

- Job-level coordination
- Task parameterization
- Load balancing

- Task failure
- Cost

Job-level coordination – it is beneficial to consider a job in its entirety prior to its execution rather than simply submitting jobs on first-come-first-serve (FCFS) basis. Global overview and comprehension of the job with respect to input file size, input file format and characteristics, data locality, available resources, and user requirements is desirable. Through recognition of these characteristics, the scheduling objectives, and thus the scheduling algorithm, can be adjusted more adequately. By considering a job as a unit, a job plan can be developed a priori eliminating many uncertainties as job execution gets under way (*e.g.,* job budget, input file distribution, data accuracy, expected task failure rate).

Figure 14 points to the effect job planning may have on overall job runtime characteristics. In the figure, the left most three bars (colored in red) represent runtimes of tasks resulting from the job plan submission; remainder of the bars represent runtimes of the tasks created without a job plan. For the job plan option, a BLAST job was submitted across three heterogeneous grid resources so that the job workload (1,024 queries) was distributed across resources to match the resources' relative performance. Technical job resource details are available in Appendix C where the following resource configurations were used: Ferrum resource with 5 nodes, Everest resource with 10 nodes, and Cheaha 1 resource with 15 nodes. In order to obtain resource performance, a benchmark of resource performance for BLAST was needed; this value was obtained from AppDB. The same input data set was then submitted across the same resources in a FCFS basis. The input file was divided into 120 chunks using the UNIX *split* utility and

submitted to available resources. As a resource completed processing an assigned chunk, another chunk was automatically submitted to the given resource (after multiple experiments with a variable number of chunks, submitting 120 chunks resulted in shortest job runtime so results for 120 chunks are shown). Results in the figure show the benefit of the job plan resulting in significantly lower load imbalance across tasks. Such job execution and has lead to approximately 30% shorter turnaround time for the job.

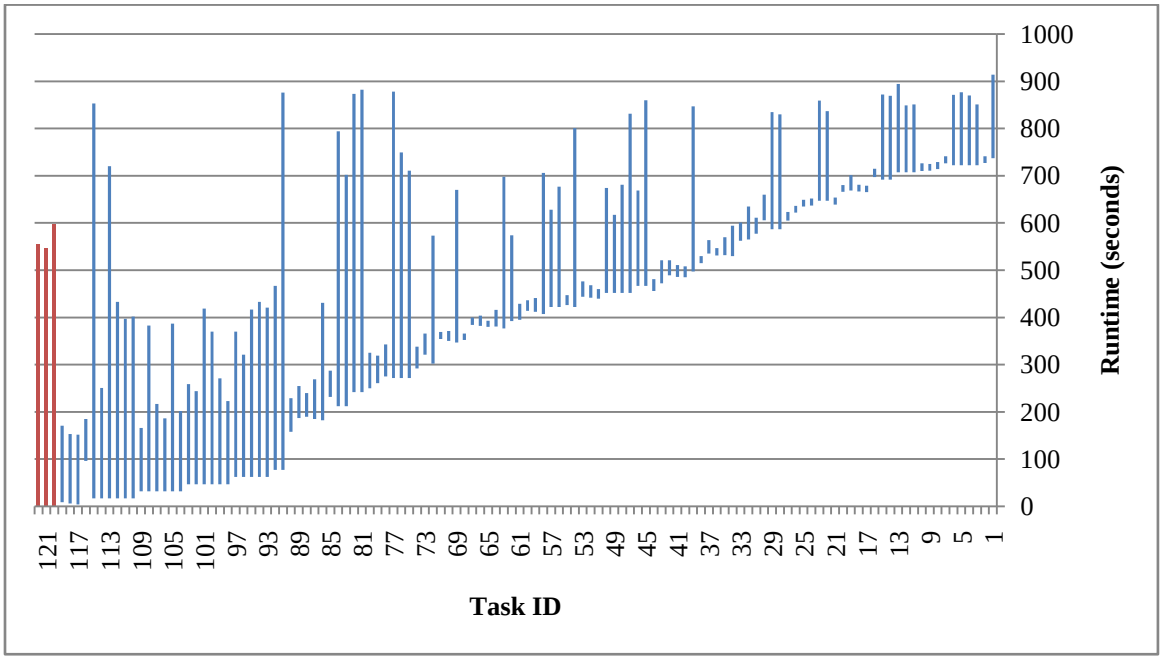


Figure 14. Runtimes of tasks distributed across three resources when using a job plan (left most three bars - colored in red) and when using first-come-first-serve approach with 120 tasks (remainder of bars). Overall, job planning leads to 30% shorter turnaround time. Jobs executed 1,024 query BLAST search against the 1.6 GB nr database.

Task parameterization – once created, each task can be seen as an individual job. This is especially true from the resource perspective. Depending on the characteristics of the execution resource, parameterization of a task should be independently handled and optimized for the resource at hand. Based on the amount of data available for current application and resource, in addition to the level of understanding of an application

performance model, this process could require variable amount of effort. Effects of task parameterization can turn a poorly behaving resource into a competitive one or vice verse.

An example is shown in Figure 15 where a properly parameterized task (Parameter set 2) executing on an old Sun SPARC machine is comparable in performance to an improperly parameterized task (Parameter set 1) on a much newer Intel Xeon based resource. The difference in the Parameter set 1 and Parameter set 2 is the number of threads that were employed to process given BLAST job. The Parameter set 2 exploited resource scheduling policy to realize presented runtime reduction. Scheduling policies of given resources assigned a single user job request to a single compute node, irrespective of the number of processing cores available on the node. Recognizing the number of cores available on a node and starting that number of threads on given node leads to shown runtime reduction.

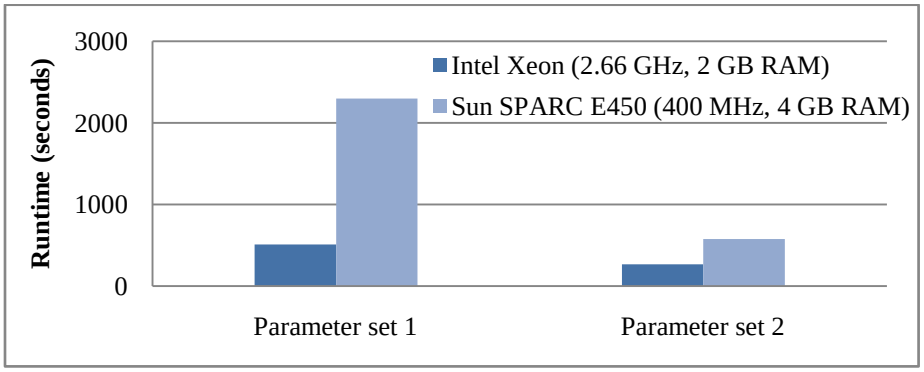


Figure 15. Significance of task-level parameterization - improperly parameterized task can result in a resource behaving poorly. Parameter set 1 used basic invocation of BLAST where only a single thread was instantiated to process the input. Parameter set 2 matched the number of threads to the number of cores available on the resource.

Load balancing – although each task of an EP application executes independently, minimizing load imbalance across tasks results in maximization of resource utilization

and thus minimization of job runtime (*i.e.*, maximization of job performance). At the job level, by aiming at achieving load balance leads to task dependencies, thus making efficient scheduling of this class of applications significantly more difficult. This is further complicated by the task-level parameterization discussed above. An approach to managing this issue can be seen in the job plan management that is aware of individual task-level optimizations and can thus guide the computation toward set goals. Other possible directions for achieving load balance in grid applications include adaptation of techniques used in parallel and distributed environments (*e.g.*, [142, 143, 144]).

Figure 16 presents an example of the effect load imbalance may have on runtime of a job. Corresponding BLAST job input file was divided into a number of chunks using the UNIX *split* utility. Each of the chunks was submitted to an available resource node. Because of the structure of the job input file (discussed further in Section 4.2.1), tasks exhibited shown runtime characteristics. Not understanding and leveraging effects of job input data resulted in shown load imbalance, poor resource utilization, and excessive job runtime.

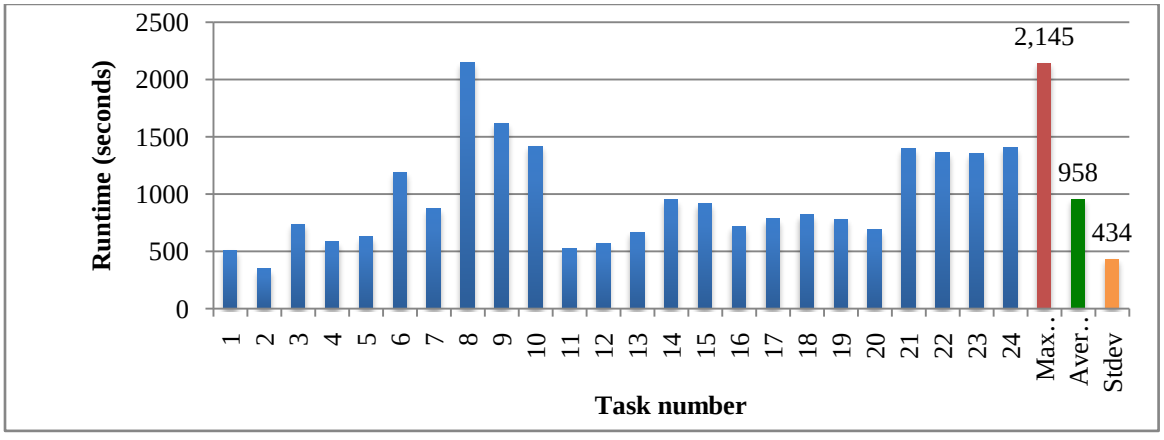


Figure 16. Variation in runtimes of a set of BLAST tasks caused by load imbalance between the tasks. Tasks performed a search against the 1.6 GB nr database using 4,096 query input file. Load imbalance was caused by disproportionate workload assignment to individual nodes in terms of query lengths.

Task failure – superficially, execution of EP class of applications with no task communication seems like a perfect platform for easy handling of partial task failures. However, under the goals discussed throughout this section, task failure introduces a significant issue. The aim of developing a job execution plan is to provide additional insight into job execution and try to meet user objectives. Once a task fails, current job plan is invalidated and, depending on the failure situation, could completely disrupt outlook of successful job completion.

In order to cope with task failure, the following considerations apply. As a general observation, tasks executing on grid resources have the highest failure rate during their initialization and startup process. This can be attributed to resource miss-configurations, task miss-configuration, policies, events local to a resource, as well as any of other host of variables. Consequently, a much higher probability of success can be assigned to a task that has reported as being under execution. Alternatively, multiple submissions of a task can be initiated, albeit resulting in wastage of resources. In addition, a timeout mechanism may be implemented that checks on a status of a task and either cancels it (in case of multiple copies of the same task exist) or spawns another copy of the task (in case of a non-responsive, single task instance). If balance between computation time and number of tasks is well tuned, dynamic task distribution can offer highest tolerance for task failure [145]. However, other issues, such as consistent resource availability, may pose alternate problems. While there is no safe-guard against task failure, observations and models such as these can be incorporated into the job planning process leading to a more robust execution.

Cost – considering the general direction of grid computing toward enterprise and cloud computing where economic aspects are being increasingly prevalent [74], cost associated with job execution is becoming a concern. Since job cost is obtained from cumulative task cost, utilization realized by each task is directly proportional to the overall job cost. From the perspective of job metascheduling, it is thus important to generate an efficient job execution plan that satisfies user's requirements while yielding profit to the resource owner.

Adopting the job plan model just discussed leads toward desired estimations regarding job execution control and alternatives. Figure 17 (also shown as Figure 3), indicates the variation in job cost as caused by different job plans. The three cases, shown, represent the outcome of manipulating the internal characteristics of the input data files in addition to the distribution size of data chunks that were passed to individual resources. As can be seen, realizing an effective job plan that meets resource capability leads to considerable reduction in job cost and thus increases user's QoS.

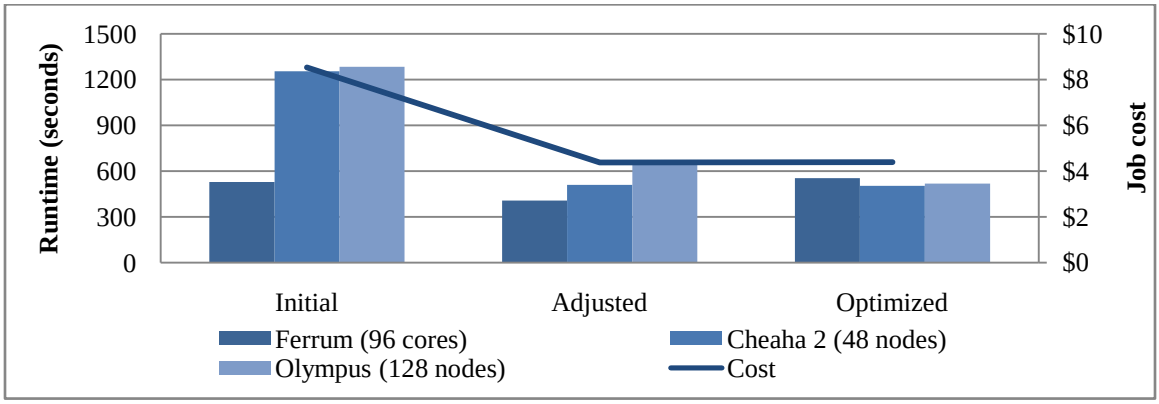


Figure 17. Difference in job runtime, load balance and cost between a naïve job submission and an optimized job submission of a BLAST job. Jobs performed the search against the 1.6 GB nr database using 4,096 query input file.

In the context of enterprise and cloud computing, user requirements are likely to become the major driving force behind job metascheduling policies. Unlike the more traditional cluster computing environments where resource utilization was the main objective, service oriented architecture promoted by the grid infrastructure is advocating a user centric orientation. Here, the quality of a system is realized by user satisfaction and measured through user utility. These notions are realized through the QoS requirements imposed by the users and agreed upon by the resource provider through the Service Level Agreements (SLAs). With future advancements of pervasive computing, a user is likely to require detailed insight into their consumption of computational power further complicated with imposed requirements on alternative job execution plans. Although current efforts in metascheduling EP applications primarily focus on runtime minimization, cost may take on different forms in future systems (*e.g.*, result accuracy, system responsiveness, power consumption, availability) and will become a primary metascheduling consideration. Therefore, considerations such as the ones listed in preceding paragraphs will become more important and prevalent and will need to be cumulatively manipulated.

2.9. Other Related Work

Besides the concepts and projects most relevant to the topic of this dissertation and covered throughout Chapter 2, this section provides additional information about applications and tools that are either tangentially relevant or used in presented work.

2.9.1. *Bioinformatics Application Domain*

The sequence alignment process enables identification of regions of similarity among biomolecular sequences (*i.e.*, DNA, RNA, or amino acid sequences), where high

sequence similarity often implies significant functional, structural, and evolutionary information between genes. Finding such similarities enables derivation of inferences about gene functionality and ancestry. Sequence alignment has grown to one of the most important aspects of today's biological research [146]. The alignment process consists of comparing multiple sequence queries by searching for series of matching individual characters or character patterns across the sequences.

One of the most widely spread algorithms for sequence alignment is the Basic Local Alignment Search Tool (BLAST) [15, 147]. Although there exist other sequence alignment algorithms (*e.g.,* FASTA [148], SSEARCH [20], HMMer [21]), BLAST has gained popularity because of its emphasis on execution speed. At the same time, the advent of high throughput sequencing technologies and large scale projects, such as the Human Genome Project [149], have led to exponential growth of search target databases and thus exponential search times [132, 146].

In order to cope with the prolonged search times, parallel computing techniques have been used to help BLAST jobs gain speedup on searches by distributing search jobs over a cluster of computers. There are two main methodologies for parallelizing BLAST searches, namely *database segmentation* and *query segmentation* (depicted in Figure 18). Database segmentation methodology (employed by mpiBLAST [22] and TurboBLAST [150]) distributes a portion of the sequence database to each cluster node. Thus, each cluster node only needs to search a given query set against its portion of the sequence database. Alternatively, query segmentation (employed by [25, 151]) distributes portions of the user submitted queries to available cluster nodes. Each compute node has access to the whole database and performs the search only on its portion of the query set, thus

speeding up the overall search effort because of the developed parallelism. Query segmentation represents an instance of the EP class of applications discussed in Section 2.3.1 and is the focus of this research work.

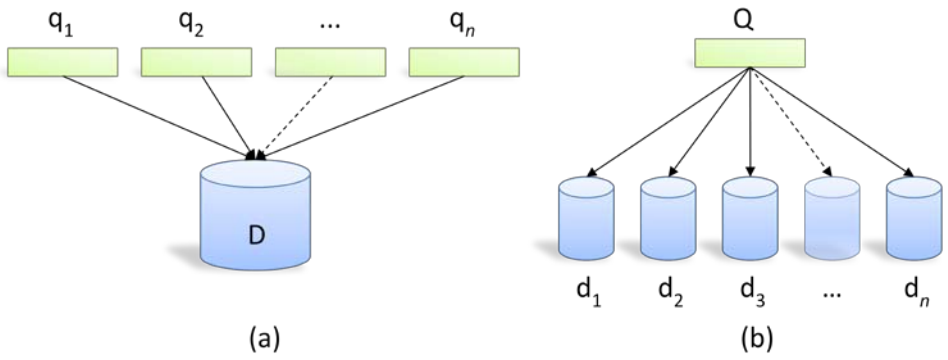


Figure 18. Two models for parallelizing BLAST: (a) query splitting and (b) database splitting

Nevertheless, executing alignment searches presents a challenge in terms of required resources because neither individual workstations nor individual clusters are capable of realizing needed performance. For example, a small-scale search with approximately 1,000 search queries against a popular 1.6 GB *nr* database on a decent workstation (dual CPU, dual core Intel Xeon 3 GHz, 4 GB RAM) takes approximately 8 hours to complete using one thread per CPU. Because of the speed at which sequencing machines perform analysis, it is not uncommon to generate on the order of 250,000 queries in a single experiment[2]. Performing a search with that amount of data presents itself with a challengingly high computation cost. Resorting to publicly available resources such as NCBI [152] is still not enough because they impose a one cumulative CPU-hour limitation per user job [153].

[2] Based on personal communication, Clemson University, Clemson, SC, February 25, 2008.

In order to realize the benefits of increased execution speed offered by the parallel approaches and gain access to needed resources, scientists are aiming at using grid computing technologies (*e.g.*, [154, 155, 156]), which offer needed computational capabilities to user's fingertips. However, already discussed heterogeneity of available resources introduces additional complexities when executing such jobs across the grid. With a goal of improving runtime characteristics of BLAST jobs across grid environments, BLAST was chosen as an application to work with during this research.

2.9.2. *Statistical Genomics Domain*

Two different applications from two different domains were considered in the validation stage of presented research. Besides BLAST application discussed in the previous section, the second application belongs to statistical genomic field and was selected from UAB's Department of Biostatistics, Section on Statistical Genetics (SSG)[3]. The application represents an application of statistical methods to understand gene expressions and how those relate to expressed genetic traits. Statistical methods are used to determine distribution of traits in an attempt to build a mapping between general genetic traits and an individual's expression of those traits; for example, which genes control the expression of diabetes? The analysis is performed on real experimental results with variables being regions of genome and those may vary in length from 10,000 to 100,000.

The execution of the application is based on random data generated for simulated analysis with the goal of understanding behavior of statistical methods onto real data analysis. The application needs to be executed repeatedly, as new methods are studied. The data analysis is implemented in R [157] and takes as input a set of parameters. The

[3] http://www.ssg.uab.edu

number of parameters varies from one experiment to the next and each experiment needs to be performed a certain number of times. Because of the limitations of computational resources, the total number of iterations performed has typically been less than 10,000; however, to achieve desired accuracy 100,000 iterations should be performed.

From the grid computing perspective, SSG analysis code[4] represents an opportunistic platform that allows distribution of individual parameters as well as individual iterations across multiple resources. Figure 19 depicts the computational model of relevant code. The application has two inputs, a set of parameters and a number of iterations that should be executed on each individual parameter. Each of the parameters is independent of all the others and can thus be spawned to a separate resource. Moreover, individual iterations that are performed on each parameter can be split into smaller segments and executed in parallel across multiple resources. As presented in Section 4.1.1, selected application belongs to class III category of EP applications, and, due the potential impact of adopting grid computing into the computational pipeline of the SSG group that would deliver sought performance, it was used as the application of choice.

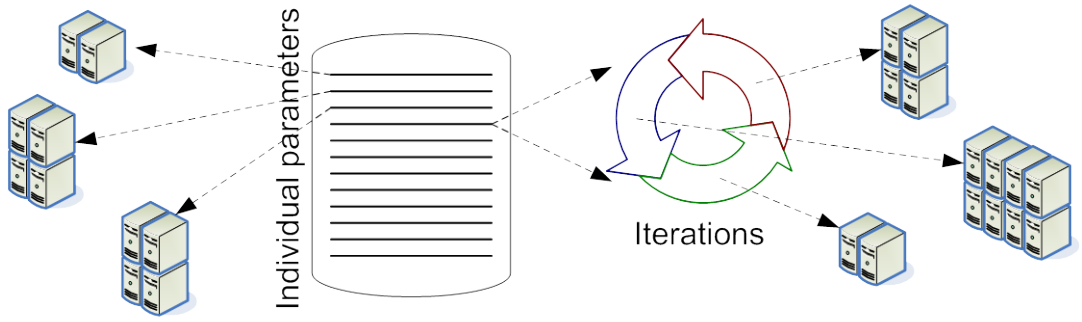


Figure 19. Computational model for statistical SSG analysis code indicating viable levels of parallelism for application execution.

[4] http://projects.uabgrid.uab.edu/r-group

2.9.3. *Simulating Grid Resources*

Application testing, and particularly scheduling of applications on underlying resources, requires a testbed to perform required operations and check for validity and success of the newly generated application and/or scheduling algorithm. Beyond only requiring a testbed, there is a requirement to compare given scheduling algorithm, in particular, to other versions of the algorithm or other algorithms altogether. The implication of this requirement is that there is a need for repeatable and controllable environment where experiments and evaluations can be observed. Because the grid spans multiple administrative domains and has a wide range of resources participating, this may be hard or impossible to achieve. Researchers, in particular the ones dealing with scheduling algorithms, thus need a framework for deterministic modeling and simulation to test developed algorithms.

As a solution to some of these problems, use of simulation methods rather than real world testbeds alleviates and enables the necessary manipulation and control of the environment. Beyond control over individual resources, use of simulations enables researchers to work with varying and otherwise unobtainable resources, allowing testing of application and algorithm scalability. Several tools were developed with the goal of simulating the grid, namely Bricks [158], MicroGrid [159, 160], Simgrid [161], and GridSim [162]. The Bricks project focuses on simulating client-server model as s global computing multi-user system. It operates under a centralized system controlling performance and providing service rates. The MicroGrid operates as a GT emulator and expects applications and the scheduler to be constructed using the GT. Because of the characteristics of emulated environments, evaluation of large-scale grid scenarios and

configurations may take considerable amount of processing power and time. The Simgrid tool allows for modeling of time-shared resources or resources restricted to a single user. Simgrid is targeted for users developing schedulers that minimize application execution time alone and offers support only for the C programming language. Much like Simgrid, the GridSim toolkit provides support for schedulers focusing on minimizing application execution time but also supports space-shared resources and a multi-user environment. As such, it supports scheduling algorithms that focus on cost minimization and deadline scheduling based on economic models.

GridSim offers a versatile set of choices in creating virtual grid with built in support for the following options:

- Modeling of heterogeneous resources
- Space and time shared modeling of resources
- Specifying resource capabilities and time zones
- Specify network speed between resources
- Weekday/weekend/holiday rate based on resource workload
- Application modeling on given resources allowing simulation of a range of parallel application models (*e.g.*, compute to I/O intensive)
- Support for static and dynamic schedulers
- Statistics tools recording selected operations for later analysis

Interaction with GridSim is done by extending a set of Java classes provided by the toolkit and writing the code to specify grid parameters including resources and applications. Upon correct environment setup, the scheduler component needs to be implemented on top of the toolkit to provide the desired functionality and focus on

particular goal (*e.g.*, resource, application or cost). Because of its versatility, GridSim has been accepted as a de facto standard for simulating grid environments.

2.9.4. *Automating Service Descriptions*

Some of the ideals of the grid computing can be seen as realizing automation and consistency across this heterogeneous environment. Much of such work can only be accomplished through development and adoption of standards and standard protocols. Automated service description and publication has never been attempted for grid applications, but there has been much work at defining and describing software components in the field of Software Engineering [163]. By using precise component definitions, one is able to advertise individual component functionality and use other such components to compose new systems. This has been realized through the Open Software Description (OSD) [164] language standard and advanced to where available components are published on the web and used as needed by software systems without any human intervention.

By supporting a method for capturing the core purpose of the application, requirements, and options, the end user is provided with specific information that describes the application. After successful installation, the second most important feature enabling application use is the interface that the application provides to its users. With respect to grid applications, the most appropriate way to interact with an application is through a web-based interface that requires no local installation of the application. A web-based interaction may also provide special tools and knowledge to access available resources [50, 165]. By providing a default standardized interface to the given application automatically, a resource owner may reuse the interface rather than implementing their

own. The benefit for the end user is that the interface stays constant across different providers. An additional benefit is a reduction of possible errors during interface generation caused by possible misunderstanding or lack of application knowledge.

The Pasteur Institute Software Environment (PISE) [166] is an example of previous work where the notion of interface description has been adopted in practice. PISE is a transformation tool that receives as input a PISE-DTD compliant XML document and interprets the document to create any of the suite of interfaces ranging from HTML to CGI [166]. A scalable core is also provided by PISE that can be extended to add additional interface interpreters as needed. PISE currently contains a database of over 200 XML documents corresponding to interfaces for various applications that are primarily focused on bioinformatics. By leveraging initiatives and technologies such as those provided by PISE, many of the accidental complexities associated with grid application deployment can be removed.

2.9.5. *Cloud Computing*

Cloud computing [31] is an emerging technological infrastructure that makes use of existing and centralized hardware resources to incorporate and deliver functionality such as Software-as-a-Service (SaaS) and Infrastructure-as-a-Service (IaaS) directly to the consumer over the Internet. Unlike grid computing, cloud computing is characterized by localized administrative control of possibly heterogeneous and geographically distributed resources. Users gain access to cloud computing resources on as-need basis, utilizing required resources to complete a job at hand and releasing consumed resource upon the job completion. Individual clouds often provide specific functionality and thus target a specific niche of users (*e.g.*, Amazon EC2: general computing, Google App Engine: web

portal development, Microsoft Azure: enterprise application development for the web). Aggregating such variable functionalities and providing a single access point atop multiple clouds can be seen as the true potential and realization of the grid.

Work presented in this dissertation immediately applies to the concept of cloud computing primarily from the perspective of the resource owner but with a considerable potential for the end user. Concretely, performing distribution and assignment of user jobs to resources in an application- and resource-specific fashion by a resource owner, as enabled by work presented in this dissertation, would enable not only increased utilization of resources for a cloud provider but would also result in shorter turnaround time for the end user. Such developments lead toward higher job throughput and more satisfied consumers. From the end users' perspective, the true potential is realized upon generation of interoperable aggregation of multiple cloud computing providers where service comparison and competition is present. Such environment can result in increased level of support and QoS as guided by user demands.

3. APPROACH

This chapter describes the overall approach and contributions of this dissertation; rationale for the approach is presented followed by the general requirements, directions, and contributions. Rationale of the approach builds on the analysis and conclusions presented in Section 2.8.3 where complexities regarding execution of EP class of applications are presented. The goal of this chapter is to provide a high-level architecture of the framework developed as part of this dissertation and describe the interaction between the different components of the framework. As will be empirically shown in Chapter 4 and Chapter 5, adoption of the framework described in this chapter enables delivery of higher user utility through incorporation of resource-application dependencies into metascheduling and altering the interaction between a user and the scheduler.

3.1. Complexity of Grid Job Metascheduling

At its core, a metascheduler receives an abstract resource specification (typically from a user) and translates it into a more concrete definition. It does this in order to find a match for the job submission request. In the process of searching for an available match, multiple domains and resources might be traversed until an available and compatible resource has been found. Once a resource, or a combination of resources, is located, the specification is passed onto the Local Resource Manager discussed in Section 2.5.1 for job execution [167]. However, because grid environments consist of heterogeneous resources, the execution characteristics of a job depend on the resources selected for execution. Therefore, the resource capability available across individual and

heterogeneous resources, suit certain applications and application classes better than others. Furthermore, the selection of job invocation parameters that match resource capabilities, application requirements, and user desires can significantly alter job execution characteristics [14]. Consequently, the act of resource selection for job execution, or metascheduling, involves the making of multiple, interdependent decisions. As a result of the dynamic resource state across grid environments, together with the number of combinations and options for each individual job, manual methods employed by users are inadequate to reach desired potential. Automated methods that do not understand implied dependencies and available options are also inadequate to meet users' demands and satisfy user's utility. Hence, there is a need to deliver automated but system- and application-aware solutions that can leverage capabilities of underlying resources while meeting user demands.

3.2. Rationale

As an example of scheduling in the context of providing and supporting interaction between a user and the scheduler, a quote from Lee and Snavely [18] seems appropriate:

> "To understand what we mean by a "dialogue" between the user and scheduler, it will be helpful to examine the very first supercomputer "scheduler," the scheme of *Tennis Court Scheduling*. This was the first scheduler on San Diego Supercomputer Center's Touchstone Delta (W. Pfeiffer pers. comm. 2004; a contemporary Delta system, belonging to the Concurrent Supercomputing Consortium is documented by Messina 1993). This "scheduler" was in fact a set of human system operators who were responsible for managing phoned-in job requests from users. While on one level, Tennis Court scheduling is the height of unsophistication, and clearly cannot scale to the large and busy systems and grids of today, we claim that it can still provide an important perspective from which to judge software-based schedulers. Human beings possess a creativity, flexibility and nuance of analysis that outclass any proposed scheduling

> algorithm, not just in terms of bin-packing algorithmics [*sic*], but also in terms of the total user interface. One can imagine a lengthy negotiation between user and operator over job parameters and schedule availability to achieve the most satisfying result for all parties concerned. Operators, knowing the habits, personalities and relative importance of their users could assess the urgency of each job and act accordingly: backfilling, rearranging already scheduled jobs and perhaps even stopping already running jobs."

For example, assume a grid resource can be consumed at a cost of 10 cents, for one hour of application runtime on a single processing core. Assuming all else is constant, if a user submits a job that uses only one processing core and their job executes for 10 hours, they have spent $1 for their job. Alternatively, if supported by the application, the job could be split into 10 tasks and have each task execute on a separate processing core in parallel. The outcome is that the user now had to wait only one hour and still spent the same amount, namely $1. Delivering such alternative job execution options to the user, enables the user to make a more targeted decision given their current utility. Through support for such a model, the user becomes informed of their options and, in turn, can better meet infrastructure capabilities to their ever-changing desires.

It is in this context that the basic drawback of approaches of currently available schedulers is realized. Typically, a user is prompted for job parameters, which, once received, are considered for the lifetime of the job. At the same time, intuition, in addition to previous research [18], show how poorly end users evaluate the performance of their jobs, even if incentives are given. The job submission requirements may include parameters such as expected runtime, total number of CPUs to be used for the job, and minimum amount of memory required by the application considering current input data. At the same time, users, just like computers, operate on a schedule, it being daily work hours, conference deadline, or experiment timeline. Consequently, their jobs have a value

and, as indicated in [18, 168], it changes over time and from one job to the next. This complex dependency and observation is also shown in Figure 20 where, depending on the time of the day, user's value for the job completion changes. By providing a set of job execution alternatives to the user at the time of job submission, the user can select the option that suits their current utility the most.

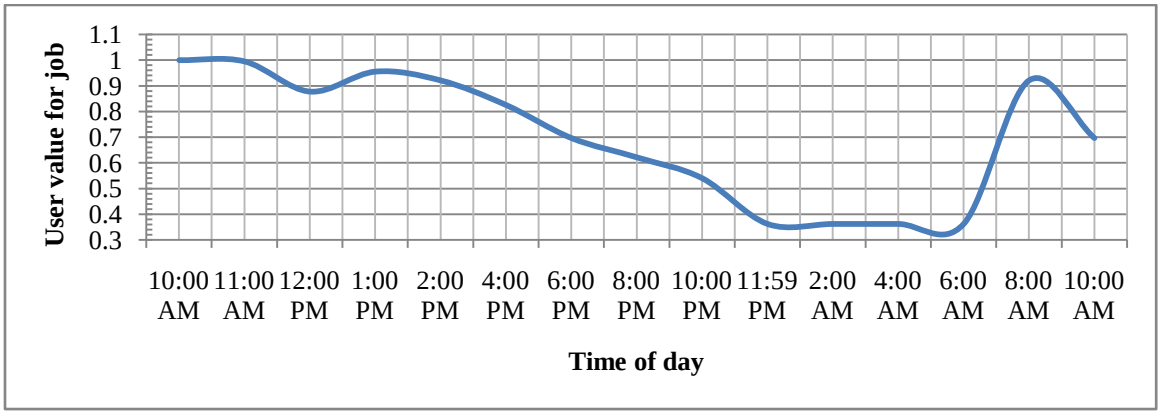


Figure 20. Value for completed job from user's perspective and associated change through a typical day

3.3. Approach Overview

In line with the examples from previous section, as users move into the grid computing environment, they have expectations and goals that exceed capabilities of other technologies that they have been dealing with previously. Examples of such capabilities include increased performance, increased resource availability, or improved resource utilization. Any of these capabilities are inherently provided by automating the process of job submission and job management. Grid job metaschedulers lie at the heart of this process and elude users from having to deal with the low-level grid details. However, there are still many required components of the job submission process that are not supported by metaschedulers. This is primarily because the metaschedulers are not aware of application requirements, resource capabilities, and user’s preferences. In order

to achieve desired functionality, users are expected to know much of required information and are required to choose among available options involved in the job submission process. As a result, users are overwhelmed by requirements and, combined with the dynamic state of grid environments, they cannot realize all the capabilities offered [14]. The result is mediocre performance exhibited by resources and unmatched expectations from users. Simultaneously, grid computing is advancing into the commercial sector, which results in incorporation of economic aspects (*e.g.,* cloud computing); however, such aspects are currently only marginally supported in the grid [100]. In this context, higher level QoS requirements and user-based adaptability will become crucial in users' daily routines. Such development of the commercial aspects within grid and cloud environments has the potential of resulting in higher requirements on automation and insight into various aspects of user presence on the grid (*e.g.,* cost analysis, performance tradeoffs, resource usage, resource utilization, etc.).

In general, the main goal of the metascheduling process can be seen as maximization of user utility. User utility can be explained as a function of variance between the requested selection and the delivered service. The value associated with this variance depends on user’s goals and requirements and is both user and job dependent, plus variable at any point in time. The requested criterion is generally expressed through some sort of QoS (*e.g.,* completion time, total cost, minimum number of average CPUs used) agreement to which both parties (*i.e.,* end user and resource provider) agree. Upon lack of fulfillment of the agreement, penalties can be put in effect, as mandated by the contract. User utility can be seen as a support tool to match and compare user QoS to economics and services available in a grid environment. User utility can also be described as an

optimization problem where an objective is minimized or maximized (*e.g.*. minimize runtime, maximize accuracy or minimize cost). For a metascheduler to optimize the user's objectives, certain application and resource parameters and parameter values need to be considered. In case of runtime minimization, typical parameters would include consumption of as many resources as input data demands. When multiple resources are available, parameters may include selecting the ones that realize the best runtime characteristics for selected application and parameterizing the given job to use as many nodes within one system as the application scales to. On the other hand, if user objective is minimization of job cost, the parameters chosen for job submission are likely to change; for example, use the number of nodes on a system that result in greatest application efficiency or select resources that require minimum data transfer.

Yet another example of user objective can be maximizing result accuracy. One way to achieve this may be to select one implementation of selected application over another. Beyond considering one objective, a user may be interested in minimizing or maximizing two objectives simultaneously; for example, minimize runtime and cost, or minimize runtime and maximize accuracy. These objectives are often conflicting, because maximizing result accuracy typically requires longer runtimes, which is in direct conflict with minimizing the runtime. Regardless of the objective, in order to realize this kind of metascheduling, the metascheduler needs to be able to make decisions and choices regarding the influence of individual parameters and parameter values on job execution characteristics, and match those to meet user's utility. The act of metascheduling application jobs in a fashion where user goals are guiding the metascheduling process is defined as *user-oriented metascheduling*. Furthermore, parameters that affect application

execution characteristics and are used to meet user's utility are application-specific. This can be concluded from observation that application A is likely to have varying performance characteristics on heterogeneous Resource 1 and Resource 2 and also that applications A and B are likely not to execute in the same fashion on any one resource, which can be summarized as *application-resource dependency*. Recognizing application-resource dependency and metascheduling jobs that leverage such dependency can further be defined as *application-oriented metascheduling*.

Combining concepts behind user goals to maximize their utility and the requirement for the metaschedulers to understand application-resource dependencies in order to deliver and meet user demands, it can be concluded that advancements in the areas of infrastructure adaptation and customization on an individual user basis are needed. In order to support such a goal, an infrastructure needs to be in place that is capable of automating the customization process and warrant its functionality. This dissertation addresses these goals by devising a high-level metascheduling architecture that enables grid metaschedulers to support notions of application-oriented metascheduling and user-level customization. More specifically, *it is possible to deliver higher user utility by incorporating resource-application dependencies into metascheduling and altering the interaction between a user and the scheduler*. Presented architecture focuses on application-oriented metascheduling from the user's perspective and it does so by providing two general contributions:

1. *Application Information Services (AIS)* are a suite of core grid services to be deployed at the level of a Virtual Organization (VO) in support of application-

specific data collection and distribution. This promotes application- and resource-specific information generalization and sharing.

2. *Focus on the individual user* by supporting two-way communication between the user and a scheduler. This feature embodies the functionality captured and offered by AIS, and presents it to the user delivering desired level of customization.

The first advancement is driven by the realization that application-oriented metascheduling enables more targeted metascheduling actions and in turn delivers a higher level of QoS to the user (as described in Section 2.6.2 and Section 2.6.3 and because such an approach enables leveraging of resource heterogeneity to maximize application and resource compatibility). In order for metaschedulers to perform application-oriented metascheduling however, they need to have access to application- and resource-specific information (*e.g.*, application requirements and preferences, resource suitability for particular application, application constraints). AIS meets this demand by providing a general suite of application-oriented services that capture and enable collection and dissemination (at the VO-level) of application- and resource-specific data that can be effectively used during metascheduling actions. With the availability of such information, especially when the information is collected at the level of a VO, general metaschedulers can rely on the overall availability of required information. Such reliance on information availability enables advanced application-oriented metaschedulers to be developed. These metaschedulers can transcend any one application, or any one VO, and yet deliver desired QoS. Developed metaschedulers can consume available data to provide application-oriented metascheduling implementations that minimize/maximize user objectives or simply streamline job execution.

As an example of such an advanced metascheduler, the second major contribution of this dissertation is a metascheduler relying on information available in AIS and enabling a user to consider tradeoffs regarding their job submission prior to the actual submission (*i.e.,* delivering customization to individual users for each individual job). This idea is embodied through the notion of two-way interaction between a user and the scheduler where the user initiates a job submission and then the scheduler analyzes application-specific information available through AIS and combines that with current resource availability to calculate a range of job execution alternatives. Generated job execution alternatives are then presented back to the user in the form of *job execution space* (*i.e.,* a number of alternative job execution options that are mapped onto conflicting objectives) allowing the user to make the final decision as to which alternative to execute (*e.g.,* one that provides highest utility to the user at the given time).

Figure 21 presents an example of job execution space where a set of job execution options is presented to the user. These options represent a set of concrete and application-oriented job invocation alternatives, as carefully selected from a large number of possible alternatives by the metascheduler. Through the presentation of job execution space to the user, the user becomes savvy of various job execution alternatives currently available to them and is able to make targeted decisions relating job execution tradeoffs. This can be seen as a generalization and an improvement of multi-objective optimization because, in case of the optimization, the metascheduler must automatically decide on the weights of individual objectives and choose one final value, irrespective of current desires guiding user's utility. On the other hand, by presenting the job execution space to the user, the

user can quickly interpret and analyze various options, consider tradeoffs and make a decision that suits the moment the most.

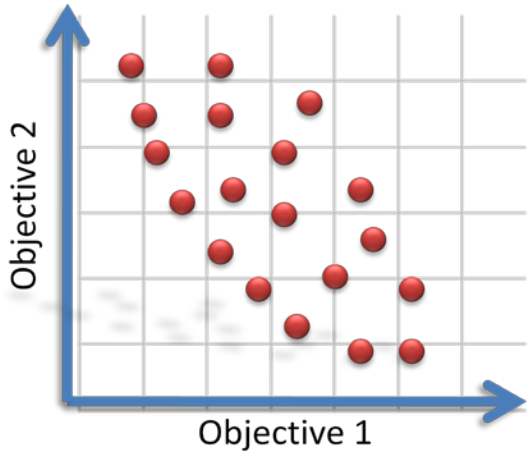


Figure 21. A representation of job execution space allowing the user to choose among available job execution options after having considered associated tradeoffs. Each dot represents a job execution option (i.e., fully parameterized job plan) mapped onto the two objectives.

Figure 22 depicts a component-level general architecture for a metascheduler supporting contributions devised as part of this dissertation. From the figure, embedded support for AIS and flow of information between the metascheduler and a user can be seen, as described in the previous paragraph. The unique aspects of the framework are: (1) it enables application-oriented scheduling while application specific information and the scheduling algorithm are highly decoupled; and (2) presentation of the job execution space to the user, which provides the user with a complete insight into viable job execution alternatives. Based on the first aspect, generation of job execution options is not only entirely automated but also application-oriented, thus hiding the low-level infrastructure management details and enabling high-level of QoS for the user. In addition, information in AIS can be used irrespective of the job execution space to enable

goals set by a metascheduler implementation (*e.g.*, runtime minimization, runtime minimization and accuracy maximization).

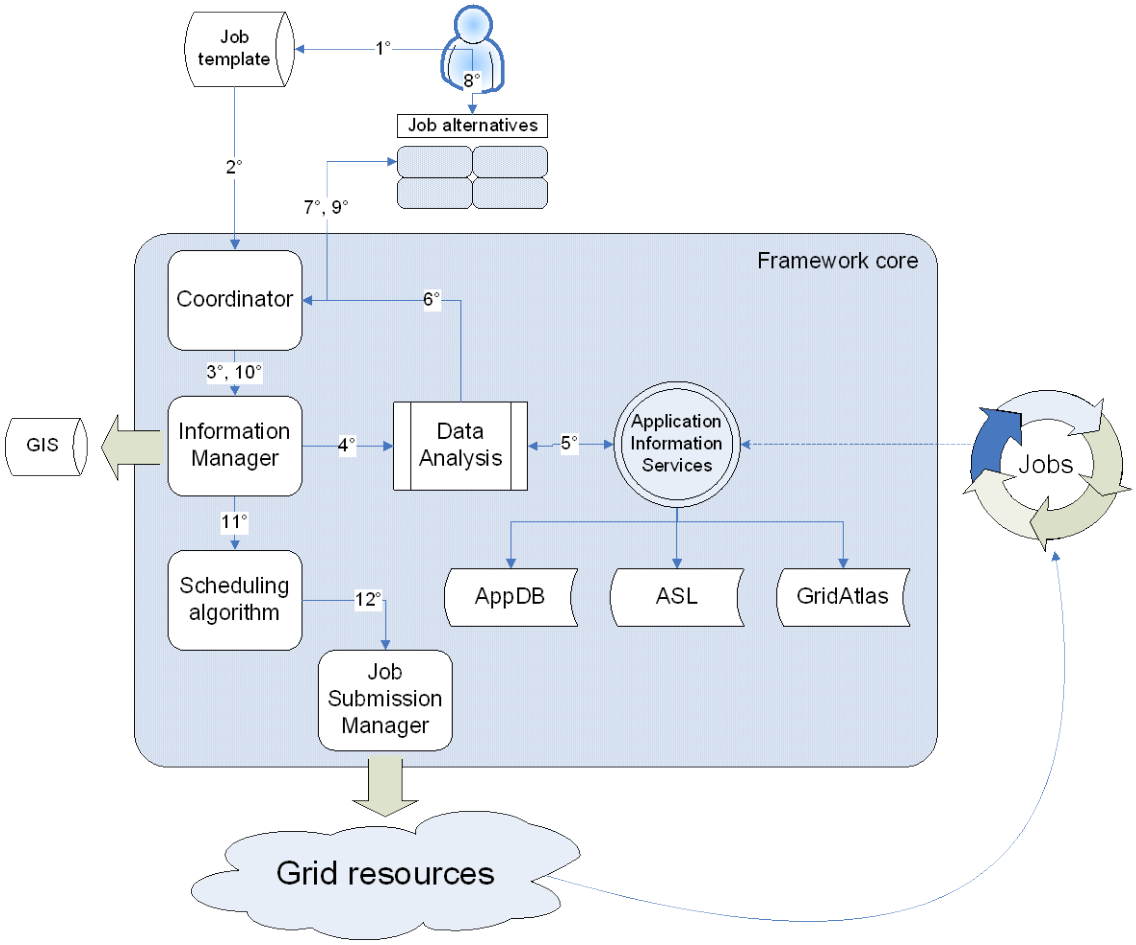


Figure 22. Architecture (showing individual components and interactions between those) for a metascheduler supporting contributions devised as part of this dissertation.

In Figure 22, the user initiates the framework through the job submission process (step 1). The job submission process consists only of the desired application that the user wishes to execute and associated input. Once the system received the request (step 2), the Coordinator module contacts the Information Manager (step 3), which assembles information from Grid Information Service (GIS) and AIS (steps 4 and 5). Once information is collected and analyzed to include the resource as well as application-specific information to derive effective mappings between the two sets of entities, the

Coordinator (step 6) manipulates derived results to provide the user with a concise and relevant set of job execution alternatives (step 7). After observing the alternatives presented, the user decides on the most suitable and desired job alternative (step 8) and passes the information back to the Coordinator (step 9). Selection is passed to the Scheduling Algorithm (steps 10 and 11), which implements application specific job submission parameters (*e.g.,* input data reorganization) and submits the job to individual resources through a job submission manager (step 12). As jobs execute, application-specific information is submitted to the AIS for future consumption and analysis, enhancing future metascheduling actions (in terms of understanding application-resource relationship).

3.4. Requirements for Described Approach

Two requirements guided the design of the present research: (1) general applicability of information collection mechanisms; and (2) delivering higher QoS for the end user. The challenge of the first requirement is a necessity to devise a solution capable of capturing the requirements and vastly differing capabilities of individual applications. The solution employed allows for a custom level of information collection not only for individual VOs but also for individual applications. Furthermore, there is support to permanently associate collected information with individual applications, thus delivering the ability for the information to move alongside the application and, in turn, be available where and when needed. The second requirement aims to deliver capabilities of the infrastructure to the user, in understandable terms. This is realized by translating the infrastructure terms (*e.g.,* individual resources, number of processors) into terms directly relevant to the user (*e.g.,* runtime, result accuracy).

A non-exhaustive list of applications and tools that could rely and benefit from adoption of application-specific information collection tool, namely AIS, is as follows:

- *Application discovery service* provides a record of available applications, as registered by participating resources. Such service represents largely static information and enables discovery of relevant applications, their approved use and provides instructions on how to invoke selected application.
- *Job submission tool* can contact AIS to obtain details about the installation properties of an application on a given resource. Such tool can thus obtain information about application installation path and location of required input data without requiring the user or resource provider to supply such information on as needed basis. The discovery of application invocation options and arguments can be obtained from AIS offering an ability to automatically generate custom (*i.e.*, application-specific) job submission interfaces.
- *Metascheduler* can perform resource selection based on available historical application runtime data and then optimize selection of job parameters for the selected execution resource. It can also query individual components within AIS to obtain specific application's performance analysis results on per-resource level and thus provide application- and resource-specific scheduling without needing to analysis at the time of job submission.
- *Performance optimizer* can be seen as a more specialized and static version of a metascheduler with a focus on long-term analysis of historical application runtime data to determine 'best' implementation of a given algorithm for a given architecture, influence of input data structure on application runtime, appropriate

granularity when distributing a job, etc. Derived analysis results can be uploaded to AIS for sharing among VO participants.

- *Accounting services* can deliver information about use and utilization of individual applications or resources. With further adoption of cloud computing and grid economics [169], insight into past application execution will become a necessity for bookkeeping purposes. Users can use AIS to obtain their resource usage history and compare various providers or resources while resource providers can use it to establish pricing policies.

Although presented sample use cases of the data available through AIS differ greatly, they all fall under the model of requiring application specific-data to enable or improve undertaken tasks. More generally, the model employed can be described as having one or more consumers requiring data from one or more producers. Such model has been described and accepted by the wider grid community as a plausible information delivery system for distributed environments [25]. Unlike the more general Grid Information Services (GIS) [25] that offer general and current resource and service availability and status information, AIS offers more detailed, application-oriented and application-specific information and includes historical in addition to current data. However, the overall motivation, structure, and architecture of AIS mimic that of GIS due to GIS's proven record.

Dynamic resources that appear and disappear in conjunction with hardware and software that evolves over time characterize a grid environment. Because of such an environment, the system needs to operate under the constraints of data aging, data inconsistency, and data incompleteness. To cope with these constraints and to capture

required information, the information collection mechanism has been divided into three distinct but related units. These units operate at different levels of information collection and cumulatively cover the entire lifecycle of an application. Moreover, results of data analysis based on data collected by an individual unit can be stored in another unit making repeated data analysis unnecessary. Such an approach fosters information sharing and knowledge buildup. By supporting the notion of VO-wide application information collection, many of the ambiguities regarding naming conventions and data aging can be safely dealt with, further enhancing embedded functionality.

Through general support of application- and resource-specific information at the level of a VO, tools can emerge that rely on availability of such application-oriented information. This further promotes concepts behind simplifying use of the infrastructure from the user's perspective. In the context of this dissertation, the abstraction of the infrastructure has been raised to the level of an individual job. The developed metascheduler consumes application-oriented data in terms of resource and application suitability to deliver fully abstracted manifestations of job execution options to the user. Each job execution option hides low-level details that otherwise involve resource selection, specification of resource configuration and data distribution for a job, in case multiple resources are simultaneously used for current job execution.

With these basic assumptions and general guidelines, application- and resource-specific information collection can safely cope with many of otherwise cumbersome details. Availability of such data then enables and promotes application-level operations, including application-oriented metascheduling, which can greatly simplify user

interaction with the infrastructure and thus raise the level of QoS experienced by the user (*i.e.,* realize user-oriented metascheduling).

3.5. Application-oriented Metascheduling

Application-oriented metascheduling is realized through understanding and leveraging application-specific information. To enable collection of such information, Application Information Services (AIS) were designed. AIS is designed as a framework composed of multiple services that provide grid users or application tools with the ability to store and retrieve relevant information regarding individual application's execution requirements and preferences. Stored data includes application purpose, options and preferences including application invocation properties across resources, and application runtime data from previous executions. By collecting such information in a specific format and making it available in a known location through well-defined interfaces, users and tools can rely upon availability of such information and develop higher level logic to make use of needed information. AIS are realized through a composition of lower-level services, each targeting a specific portion of application’s lifecycle.

By providing a targeted set of services, AIS enables the entire lifespan of an application to be captured, including application description, invocation details, performance information and later behavior analysis. By establishing and providing such a well-defined mechanism for data collection and retrieval, higher level tools can be developed that can rely on availability of needed data. Reliance on established data availability sources further enables such tools to transcend individual VOs, or even grids, leading to a more global tool development and a wider user base. AIS are realized through the composition of lower-level services, each targeting a specific portion of

application's lifecycle. The following is the list of self-contained services that together realize AIS:

1. *Application Specification Language (ASL)* [170] is a specification that can be used by application developers and end users to describe persistent information about an application. The ASL allows application-specific information to be captured describing its functionality, options, and known intricacies (*e.g.*, minimum required memory, scalability, preferred architecture). Availability of such information enables job schedulers, end users, future application developers and deployers to obtain job invocation routines, execution instructions and comments, known performance tradeoffs, user suggestions and caveats.
2. *Historical application performance database (AppDB)* [171] offers collection, storage, and retrieval capabilities of previous application execution instances on the grid and thus offers a historical perspective of application execution characteristics. Availability of such information enables application- and resource-specific performance analysis to be performed and made available for upcoming job submission delivering desired functionality.
3. *GridAtlas* [172] is a tool that hides and automates the process of keeping track of installation properties of any one application across resources and allows uninterrupted job submission by appropriate tools. Such functionality is offered in direct contrast to manual effort otherwise required.

Figure 23 depicts the overall architecture of AIS; only major communication links are shown in the figure with the focus on information providers. Namely, ASL documents are created one per application and then aggregated in an ASL repository for

sharing and updating at the VO level. Similarly, individual resources report relevant properties to the GridAtlas Aggregator service. Lastly, as jobs execute on grid resources, runtime information is recorded in AppDB. Such information can be provided by a metascheduler or a job manager (as depicted in the figure), by the application itself (by modifying the application and embedding desired reference calls), by a local resource manager (through uploading the local accounting database), or by an application wrapper that reports pertinent data to AppDB as the application executes. Interaction with AIS, and thus information retrieval, is realized by contacting a desired service directly on as needed basis. Within AIS itself, information continuously propagates and builds on itself (*e.g.*, performance data stored in AppDB is analyzed and results summarized in the relevant ASL document). However, this act of information propagation and analysis should be implemented at the VO level and on as needed basis. In general, and as shown in the figure, individual services comprising AIS operate as standalone tools and can be used as such, but when all of the services are aggregated, the complete functionality of AIS is realized.

To cope with scalability and preserve desired functionality, we suggest supporting AIS at the level of an individual VO. Such an approach not only limits the size of aggregated information but also limits the type of information stored. This is because, as a result of the nature of VOs, it is likely that applications being executed on VO's resources are limited and similar in nature, lending itself to collection of more targeted and useful information. In addition, supporting AIS at the VO level allows the participants to rely on a certain level of service and they can thus build their tools without having to worry about exceptions or incompatibilities with the underlying infrastructure.

Also, with the VO-wide AIS support, real-world experience and information sharing is encouraged, allowing speedier advances in overall science.

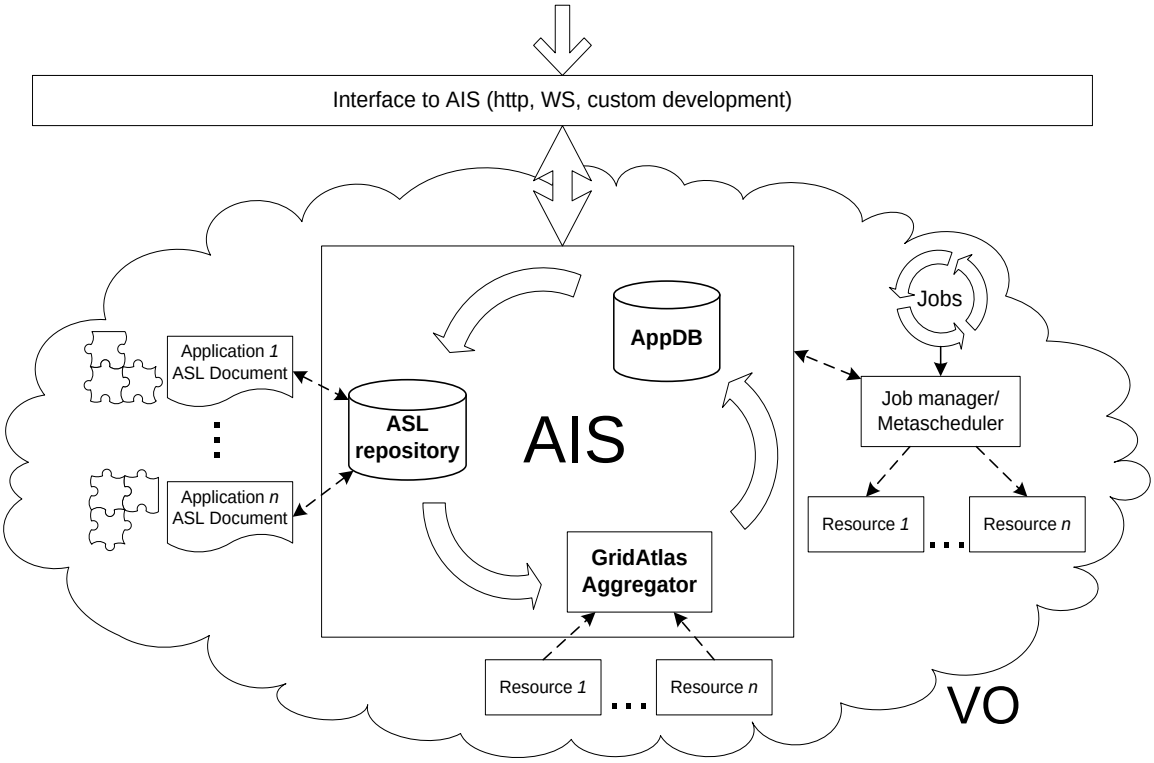


Figure 23. A high-level architecture of Application Information Services (AIS) deployed at the level of a VO with only major communication links shown.

Overall, we argue that AIS is a missing set of core services that otherwise composes the grid middleware. The de facto standard for grid middleware, the Globus Toolkit described in Section 2.2, is described as supporting three key elements necessary for computing in grid environments. These elements have been depicted as three pillars and include Resource Management dealing with grid resource allocation, Information Services dealing with provisioning of information about grid resources, and Data Management dealing with access and management of data in grid environments [35]. We believe that with the global trend of transitioning toward cloud computing environments, economic implications of such environments will require effective and controlled

execution of user jobs in distributed heterogeneous environments. In order to understand and leverage relationships that exists between a resource and an application (*e.g.,* [14], [43]) AIS should become the fourth pillar necessary for computing in grid environments (Figure 24).

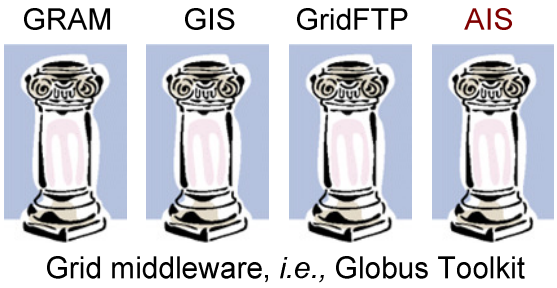


Figure 24. Fourth pillar of grid computing

The following subsections describe the individual services that comprise AIS, indicating the roles each of the services plays in achieving application-oriented metascheduling and execution in grid environments.

3.5.1. *Application Specification Language (ASL)*

In order to enable execution of jobs in heterogeneous grid environments, an application initially needs to be deployed across a set of resources and only then can it be used. It is furthermore important to differentiate the application development process from the application deployment process. Typically, high-performance computing (HPC) applications are developed using a specific programming language and a parallel programming paradigm (*e.g.,* compiler directive-based, threads, message-passing, combination of threads and message-passing) and often times the programming paradigm chosen decides the application deployment platform. If the application uses a shared-memory programming paradigm then the application can be only deployed on a shared memory system whereas an application developed using the message-passing paradigm

can be deployed on both distributed memory and shared memory systems. Furthermore, applications might require specific processor architecture, memory capacity, disk space, etc. to deliver the desired performance and scalability. Following the development process, application deployment is the process of installing the application on a set of resources. Because of the requirements and preferences imposed by the development process, the deployment process is a non-trivial task. If the deployment process is automated (as often the case is in cloud environments), it is necessary to first determine what resources are available and then decide which is the most suitable resource for that particular application.

While the task of application deployment on a grid is quite challenging, the task of using multiple applications deployed is not simpler (*e.g.*, [18]). Users have a variety of applications to choose from for performing a specific task. Many applications exist that provide the same or similar functionality, yet often there are subtle differences among those, such as programming paradigms used (*e.g.*, shared memory, distributed memory), algorithms employed (*e.g.*, multiple search or sort methods), resource requirements (*e.g.*, memory size, number of processors), numerical accuracy, scalability, and performance. Although these differences are interesting to a domain-expert, for an end user who is interested in getting a problem solved with a particular QoS requirement (for example, find the most accurate solution for this problem, or find the fastest solution to this problem) there are too many options to choose from. In order to make the best possible application selection that matches users QoS requirement, a typical user would require significant domain expertise, HPC expertise, and application expertise.

In order to ease the process of application deployment as well as execution, it would be beneficial if following an application development, an application developer could describe application's requirements and dependencies using some sort of application descriptors. As the application is executed and execution patterns are observed, it would be especially beneficial to add hints and analysis results about various performance implications and space/time tradeoffs. For commercial software packages, information about licensing and subscription could also be provided by the descriptors. A resource broker could take advantage of such descriptors during resource selection and job scheduling, in order to perform application-specific scheduling. For example, how would a metascheduler invoke a Matlab application in batch mode with five otherwise heterogeneous and independently administered resources currently available? It is likely that the user would have to specify the number of processors they believe is the required number to complete the job in a reasonable amount of time and then the scheduler would perform its tasks. However, how does the user, and in turn the scheduler, 'know' that the specified number of processors is the number of processors that will yield desired turnaround time? Is that number the same across all five heterogeneous resources? Can the job be divided across five resources or is it limited to any one? Which of the five resources should be given a preference, one with faster interconnect, faster processor, larger amount of memory or bigger number of compute nodes? What about the licensing – can the requested application be deployed at runtime (in the form of a virtual machine) or do the licensing restrictions prohibit that? And what about individual toolboxes required to execute the application (*e.g.*, parallel computing toolbox, statistics toolbox)? Beyond licensing considerations, if the application was to be deployed at runtime, what

are the required libraries and packages that need to be available on a resource for the application to be successfully deployed?

To address these goals of user accessibility and application requirements, there is a need to standardize and simplify the process of application deployment and application use on the grid. As a step in that direction, ASL represents a descriptor, along the lines of the Job Submission Description Language (JSDL) [57], that is used to describe details of a given application. ASL allows an individual application to be represented in heterogeneous world of grid computing by capturing its requirements, functionality and options. By standardizing communication protocols through ASL, tools access and process an ASL document extracting needed information about an application. For example, application descriptions are made available for immediate use or for further advancements among applications, application deployers, automated interface generators, job schedulers, and application-specific on-demand help provisioning tools. ASL can also be used to describe how an application is compared and/or combined with other matching services and software, thus also supporting concepts behind workflow systems. This capability enables individual tools to be automatically combined where the output data of one application (*i.e.*, stage in workflow) immediately feeds as input to another application. With availability of ASL and input/output data formats supported by an application, such functionality can be fully automated.

ASL is defined as an XML language with a well-formed structure. A single ASL document consists of only four distinct yet related sections, each used to describe a different component of application's lifecycle. Here, these sections are enumerated and only briefly described. In Appendix B, full details on individual sections and structure of

those is provided. In order to ease composition of ASL documents, a domain specific language that was developed part of this dissertation. Complete details regarding the creation of ASL documents and use of the domain-specific tool are available in [53, 173]. Following are the distinct four sections of an ASL document:

- *Application name and description* contains the most basic information about an application and acts as the application identification component
- *Installation requirements* section of an ASL document contains a set of required elements that describe the installation requirements and the installation procedure
- *Job invocation requirements* section focuses on providing a user with the information needed to execute the application. Starting with the executable name, it also provides the available switches and minimum hardware requirements, as well as allows the developer to specify the number of input and output files with examples of their respective formats.
- *Hints* section contains instructions and comments, mostly in natural language, providing additional application information that was not otherwise captured by other sections of the document. This section of the document provides a standardized method for storing and circulating users' experiences and lessons learned as they invoke the application.

As can be observed from the above description, there should be one ASL document created per application. Strictly speaking, ASL documents alone do not represent a service that can be simply included into AIS. Rather, individual files are self-contained application descriptors. Therefore, in the context of a VO, it is beneficial to provide an ASL repository service that would allow collection and sharing of individual descriptor

documents. Because of the VO focus, it can be expected that related applications would be deployed and used by VO's users further fostering the idea of discovery and sharing by publicizing information available in individual ASL documents. Once created, ASL documents can be modified or appended to. With such structure, ASL documents can be modified during the lifetime of an application by multiple parties. By collecting application-specific information in a single document (*i.e.,* ASL), knowledge about the application can build on itself.

3.5.2. *Historical Application Performance Database (AppDB)*

AppDB focuses on storing static, high-level application and job information over an extended period, allowing ample amount of information to be collected. This, in turn, facilitates creation of an application- and resource-specific knowledge base. Collected information can be searched or mined, resulting in the observation of relationships that exist between applications, resources, job parameters, input files, and input file properties. Recognizing such relationships and dependencies allows inherent performance tradeoffs to be observed, lending itself to improved job scheduling techniques and thus improved experience for both, end users and resource owners. Similar to the Matlab example from previous section, suppose a user would like to execute a BLAST[5] search against *nr* database with 10,000 mostly short query sequences. Suppose the same five resources considered earlier in the Matlab example are available. When the user needs to provide a job specification or the metascheduler needs to perform its scheduling decisions, the same questions as to the number of processors employed can be asked. In addition, how does the average length of the query impact application runtime? Also, it has been shown that different versions of BLAST offer different

[5] A public service instance is available at http://blast.ncbi.nlm.nih.gov/

runtime performance [17]; therefore, depending on the configuration of available resources, at what granularity should the user's input file be divided to achieve maximum load balance across resources? Should the granularity be measured in terms of file size or number of queries? Depending on the amount of available memory for individual resources and the size of the search database, how will the runtime be affected; should message-passing implementation of BLAST (*i.e.,* data distribution version) be employed or will the sequential one be more appropriate (assuming task distribution can be performed)? All these are questions that are not only application dependent, but also resource dependent. Moreover, they can significantly influence runtime characteristics of a user job [14] and should thus be considered when submitting a job. Before such questions can be fully answered, there is a need to capture relevant information and make it available on as needed basis, which is the aim of AppDB.

Unlike Ganglia [174], which is the most popular and accepted cluster monitoring tool, AppDB is unique in that it offers a historical view of application execution characteristics at the job level (as opposed to the resource level). Because a single grid application execution may span multiple resources, job-level view offers global insight into job execution properties. Such an approach allows users (or middleware tools operating at the same level) to focus on tuning performance of a job as a unit, as opposed to viewing a job as a set of disconnected tasks executing on various resources and trying to monitor performance of those individual resources. This view can assist in identifying application performance problems by noticing irregularities in resource behavior and tracking the problem down to the job specification or even resource configuration. Note that the definition and intent of AppDB is not to be used as a low-level application

performance tuning tool, which can be performed by tools such as Tau [159], Vampir [175], Prophesy [125] or NWS [76]. Those projects focus on low-level inter-task communication or task computation performance details and require code application tuning. Instead, AppDB focuses on higher level relationship that exists between an application and resources where such observations can be controlled without editing application code but rather changing resource selection or job invocation parameters.

Information that AppDB stores can be divided into two levels: job-level and task-level. At the job-level, information that is relevant and thus stored includes job-level components including application name, user, and execution time. The lower level of detail stored focuses around tasks (a job may be composed of multiple tasks that execute across multiple resources). This level of information collection includes collection of data such as task parameters, task submission information, input files statistics, and individual task statistics. The set of task statistics and/or parameters collected can be customized for each application, each job, or each user thus facilitating greater system flexibility. Access to the current implementation of AppDB is realized through two interfaces, a web-service API that enables users to incorporate and/or instrument existing code and fully automate interactions with AppDB. Additionally, a web-based front-end is provided to accommodate for manual, but more user-friendly interaction.

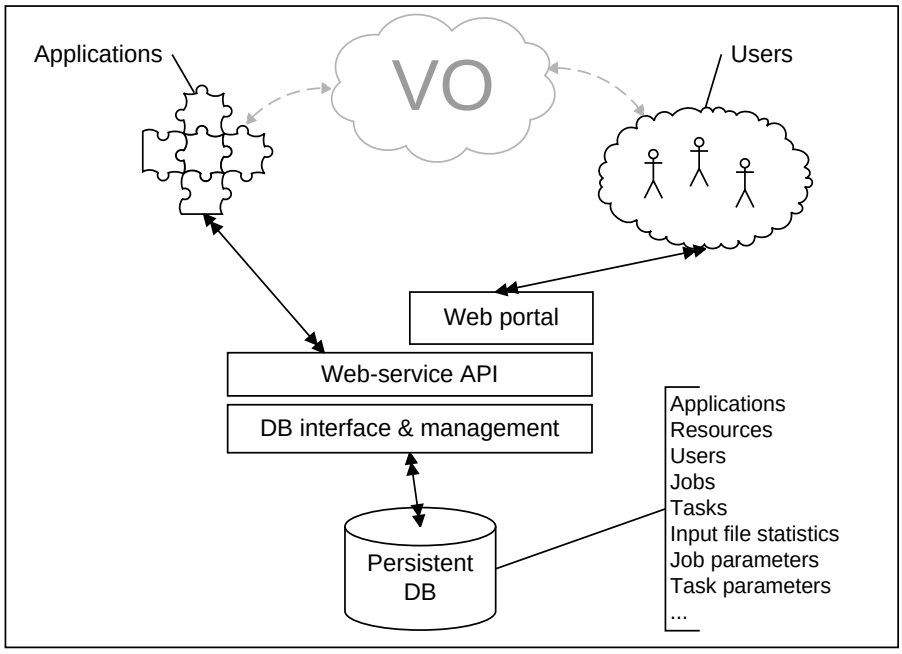


Figure 25. High-level architecture of AppDB

High-level architecture of AppDB is provided in Figure 25, showing access interfaces and interaction modes for AppDB. As can be seen from the figure, a single AppDB can exist per VO with many data providers or consumers connecting to it. Such a setup promotes information sharing and leads to a more generalized and rapidly evolving application information knowledge base. The term *Applications* depicted in the figure can refer to any application that is capable of automated communication. Examples of such applications can include metaschedulers that query AppDB about historical performance of applications whose jobs are queued up, *smart-applications* that can adjust their execution configuration during runtime in response to changing resource availability, or analyzers that study application-specific data over time to derive analytical models of the application. A snapshot of AppDB web interface is provided in Figure 26 showing a set of sample runtimes for several jobs and task details for one job.

Figure 26.Snapshot of AppDB web interface showing a list of jobs and task details for one job.

3.5.3. *GridAtlas*

With multiple instances of an application deployed across grid resources, one particular component that is likely to differ across those resources is the installation path of any one instance. Suppose a user submits a job through a job manager (*e.g.*, Globus Toolkit [35] or GridWay [109]) and requests the job use BLAST application across three independent grid resources. In order to invoke this application (assuming the user has required permissions on selected resources), the job manager needs to know which version of the application to invoke, where the application is installed on each of the resources, as well as if and where requested database is available on those resources. If the database is not readily available, the job manager should know where on the resource it can upload the database for the duration of the job. In order to resolve these requirements, the job manager is likely to request the appropriate answers from the user

(in the form of an RSL document for the Globus Toolkit or a Job Template file in case of GridWay). The user will then have to manually provide required data. This can be accomplished by manually accessing each of the resources and retrieving the data; alternatively, the user could define an environmental variable on all of the resources and instruct the job manager of variable's existence. Yet another option would be for the system administrators of relevant resources to define a system wide environment variable that exists and is the same across all resources within a VO (*e.g.*, [176]), or the system administrators can agree to install the application in the same location, support the same pool of applications and associated input data. These are all tedious solutions that are not scalable and are error prone. They also require conformity to a common set of policies, which stands against the core ideology of grid, namely site autonomy. A more automated method would be preferred that would hide such low-level complexities and not leave those for the end user to deal with and manage.

To that extent, GridAtlas is a tool that hides and automates described process by keeping track of resource and application instance details. At its core, GridAtlas is a simple lookup tool that matches a request to a previously stored value. Existence of such a tool enables a job submission manager to, upon user job submission, contact GridAtlas and obtain necessary information to complement user provided information regarding job details. The job submission can thus continue without requiring the user to specify resources that the application is installed on or where is it installed on those resources.

Because information stored by GridAtlas is resource-specific, it is important that the providers of the information be close to the instance of GridAtlas keeping the information. At the same time, because there can potentially be many providers, it is

necessary to aggregate available information and present it in an easily accessible and well-known location. To comply with these requirements, we have designed the architecture of GridAtlas with two service levels: individual GridAtlas Daemons (GAD), located directly on the information providers and VO-wide GridAtlas Aggregators (GAA) that collect information from multiple GADs (see Figure 27). The information in GADs is populated by the system administrator in charge of the local resource. The amount of data stored in a GAD is relatively small and we have thus not found a need to automate the process. By allowing manual population of the data, the system administrator can provide information only about the supported applications, thus enabling a high-level of control over what gets publicized. The GAA represents a VO-wide service that aggregates information from multiple GADs and provides a single access point for VO users or tools. Because aggregated information depends on the information providers to provide and update any information, GAAs can serve information through one of two methods: (1) act as a name-serving entity that stores contact information about known GADs and direct the query to the source directly, or (2) create a local copy of the information reported by registered GADs and serve it immediately following a request. There is an obvious tradeoff between the speed of query and accuracy of the data. The choice of mechanism used can be made at the VO level depending on VO's characteristics and requirements. Overall, interaction with GridAtlas is performed through a set of well-defined web-service APIs or through a web interface. In case data serving GAA (as well as AppDB and ASL repositories) where the size of a VO or the data being aggregated becomes substantial, well known replication and mirroring technologies typically associated with large database installations and directory

services (*e.g.,* Active Directory, LDAP) can be implemented to handle failure and scalability issues.

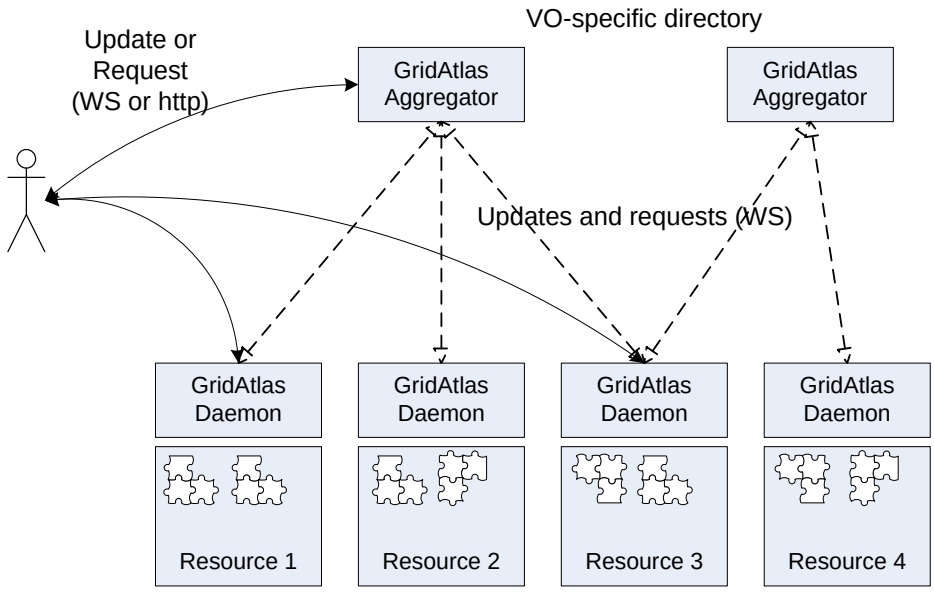


Figure 27. Architecture and interaction modes of GridAtlas

3.5.4. *Security Considerations in AIS*

Typically, access and use policies for individual resources available in a VO are set at the VO level and then refined at the level of individual resource owner, with the Grid Security Infrastructure (GSI) [40] typically used to implement security mechanisms. Within AIS, support is provided for the information provider to specify who is allowed to access and view published information. This is done for each piece of information made available. Depending on the implementation of individual AIS service, such support can be implemented as simple flag making information either private or public, implementing a security mechanism similar to the UNIX file system security with read and write permissions, or providing full support for the GSI identity credentials and requiring mutual authentication. Regardless of the selected security mechanism, required

permission or credential data is stored at the same level as the actual data and can thus be automatically propagated to the aggregator services. As such, the permissions easily traverse individual resources and VOs, and allow a spectrum of access policies. In the current implementation of individual services within AIS, the data security is realized as being either public or private with no additional granularity at various levels of a VO.

3.5.5. *Composition of Services*

In this section, we discuss how and at what stage are the individual services that comprise AIS integrated into the application lifecycle, how these services interact with other tools and thus enable execution of applications across the grid and cloud environments, in addition to possible performance implications. Overall, AIS is realized as a composition of above described services; operating together, these services satisfy the requirement to capture the entire lifecycle of an application. Although these services can be run individually and be accessed as stand-alone tools, in order to support the notion of a well-defined environment for application-oriented scheduling and execution, deploying all the services in unity as comprehensive AIS is deemed beneficial. If all the services are deployed within a VO, the general benefits of AIS can be advertised, and participants can rely on the established and defined set of tools and services. Because of the nature and purpose of these services, the location and multiplicity of them varies but the following can be used as general guidelines within a VO:

- *ASL* is an application level descriptor and there should be one ASL document per application. Simultaneously, there should be a single ASL repository service per VO to enable collection and easy sharing of available ASL documents.

- *AppDB* can be configured as a single instance within a VO allowing multiple information consumers and providers (*e.g.*, job submission managers, applications) to easily report or retrieve captured data from a well known location.
- The complete functionality of *GridAtlas* requires a GridAtlas daemon service instance to exist on each participating resource accompanied with a single GridAtlas aggregator per VO.

3.5.6. *AIS Implementation*

Individual services within AIS were developed in Java as Web Services [10] and thus support service and application interoperability. Metascheduling tools utilizing AIS interact with those services on as needed basis and extract data that is then operated on. Chapter 4 points at the type of data that is stored in AIS and how to make use of it. Data storage for both AppDB and GridAtlas has been implemented on top of MySQL database [177] using Hibernate [178]. Use of Hibernate provides immediate support for a range of relational database implementations without requiring any changes to developed software. The Web Services and associated web-frontends for both services were implemented using Apache Tomcat web server [179] and Axis [180]. Developed tools are freely available for download from lab's website[6]. Additional technical implementation details about individual services are available in previously published works: ASL [173], AppDB [171], and GridAtlas [181].

3.5.7. *AIS Usage Scenario*

In this section, a general usage scenario of AIS is presented depicting ability and usefulness of combining individual services into the AIS framework:

[6] http://www.cis.uab.edu/ccl/

1. Upon completing development of a new application, a developer composes an ASL document describing the application and uploads it to an ASL repository. The provided ASL document (see Appendix B for an example) provides a description of the application, installation requirements and procedure, and usage options.
2. The newly developed application is deployed by respective local system administrators on a number of resources. The deployment process is aided by the content of the ASL document where installation requirements and installation process were described. In future, it can be envisioned that automated application installation and deployment methods (*e.g.,* GridRPM) may arise that could use available information directly from an ASL document, further advancing the idea of the application-specific environments.
3. System administrators register their installation details with the local GridAtlas Daemon (GAD) instance that resides on the installation resource. Local GridAtlas instances then register new data automatically with the VO's GridAtlas Aggregator (GAA) service, making it easily available to the remainder of the VO. Figure 28 shows this interaction and propagation of data between GAD and GAA though an event diagram.

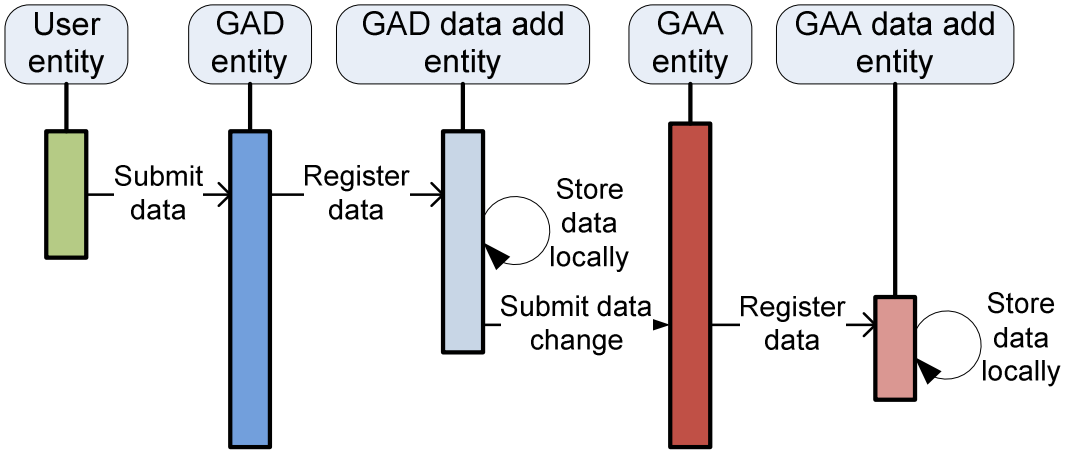


Figure 28. An event diagram for registration or update of data incorporating data propagation from GAD to GAA

4. The user submits a request to run the new application by simply selecting the application and providing input data. A sample job specification for BLAST application is provided in Figure 29 where the user simply specifies desired application, input, and output files while the rest is left up to the metascheduler and AIS. In order to realize job submission captured in this figure, the metascheduler queries ASL repository to discover invocation requirements and application preferences. AppDB is queried to discover any historical performance data or models (in current scenario, because this was a newly installed application, AppDB data will not yet be available so approaches introduced in Section 2.7.8 and described in Section 4.2.2 should be used). Next, the metascheduler queries GIS and GridAtlas Aggregator service to discover resources where selected application and needed data is available. Metascheduler invokes its resource selection algorithm to perform resource selection and develops a job execution plan.

```
APPLICATION  = BLAST
ARGUMENTS   = -p blastp –d nr -i input.fas_${TASK_ID} -o results.out_${TASK_ID}
STDIN_FILE  = /dev/null
STDOUT_FILE = out.${JOB_ID}
STDERR_FILE = err.${JOB_ID}
INPUT_FILES = input.fas
OUTPUT_FILES = results.out_${TASK_ID}
```

Figure 29. Sample job specification provided by a user at the time of job submission to AIS-integrated metascheduler.

5. Devised job execution plan is passed to a job manager (JM), which queries GridAtlas Aggregator service once again to discover application installation path including any other input parameters pertinent to the given application, job and resource. Finally, the job is submitted without any user intervention in an application-specific manner using a metascheduler. The overall process of job submission and integration of a general purpose GridWay metascheduler into the AIS framework are shown through an event diagram in Figure 30. The figure points at the behind-the-scenes interactions that take place between user's job submission and job instantiation on selected resource. Resource selection and binding of application- and resource-specific data take place on user's behalf, thus streamlining the job submission process. As a comment, depicted wrapper on top of GridWay is provided as part of GridAtlas software distribution package.

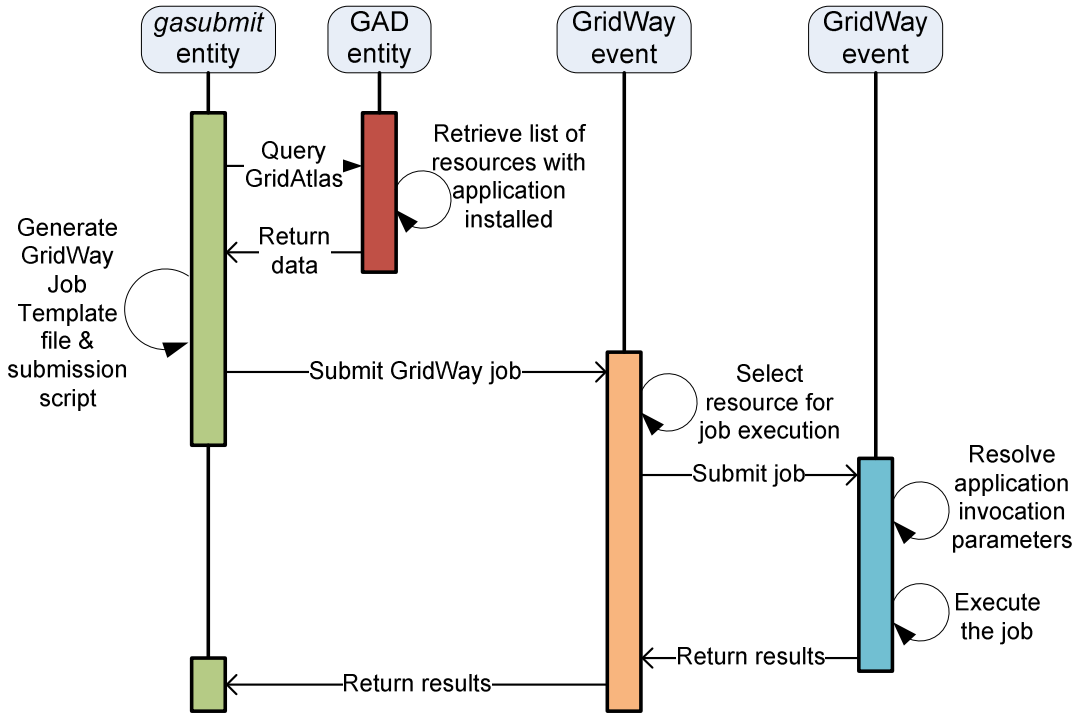


Figure 30. An event diagram for GridAtlas wrapper on top of GridWay metascheduler. User job submission is streamlined by extracting application- and resource-specific information automatically on user's behalf from GridAtlas service.

6. As the job executes (or at least when it completes), performance data is recorded in AppDB (*e.g.,* performance data that is analyzed in next chapter) where it can be used in future application invocations to improve resource selection process. Relevant data reporting can be done by the metascheduler, the job manager, the application itself, or a separate monitoring tool.
7. As the application performance data becomes available in AppDB (after repeated application executions), application performance analysis can be performed on collected and available data. Analysis can incorporate and focus on anything from the job execution cost, application scalability on a given system, input data impact or development of an analytical model.

8. The output of the analysis can be used in conjunction with user suggestions (*e.g.*, [182]) to update the Hints section of the ASL document and thus enable future dissemination of derived data, which can also be used to support development of future versions and improvements to the application.

Figure 31 captures the described data progression and general workflow of integrating individual AIS services. As can be concluded from the provided scenario and illustrated through the figure, the information collected and provided through AIS represents a cyclical paradigm. It is a constant cycle of application execution followed by adjustment. The adjustment does not necessarily need to refer to the executing application adjustment (*i.e.*, application source code); instead, it can refer to the adjustment to the metascheduler itself or just the job scheduling process. Whichever the case, because of the application-specific data availability delivered through AIS services, the updates can be more specific and targeted. Consequently, modifications to an application can be guided by user suggestions or analysis results, thus contributing desired application functionality.

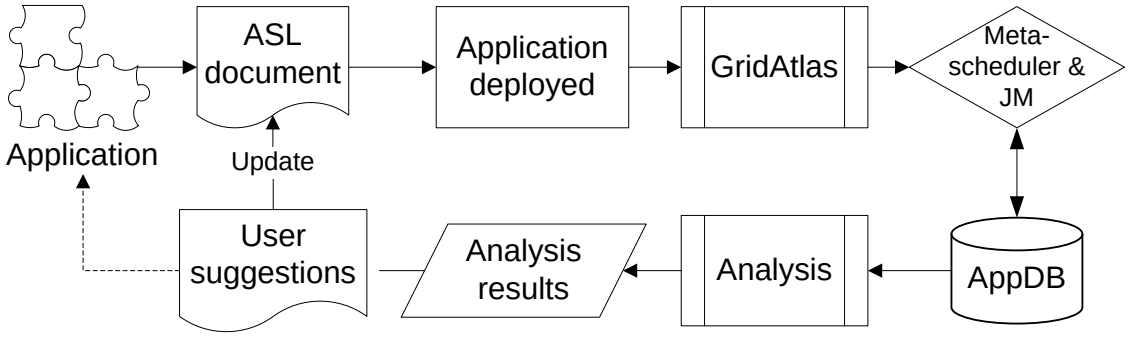


Figure 31. Integration of AIS into the application execution control flow.

3.6. User-oriented Metascheduling

Building on the concepts developed and established in the previous section, this section focuses on applying delivered functionality in a user-oriented fashion. Therefore, described approach completes the quest of fundamentally changing the interaction mode between the scheduler and a user. As described in Section 2.5 on related work, most of the advanced previous work in grid job scheduling area aims at automatically optimizing execution of user's jobs in terms of execution time, or cost, or both (*e.g.*, [30]). Component presented in this section focuses on exploring and presenting a set of tradeoffs and concrete values regarding one's job before job execution begins. Through this model, the user is exposed to and is interacting at the true service level. Rather than keeping track of how many resources were selected for execution, how many processors are employed on each resource or how much data is being transferred between hosts, a user is presented with discrete, quantitative metrics – for example, job execution time and associated cost. To accommodate such interaction mode, the *user-oriented metascheduling* approach developed here analyzes and maps information available through the AIS onto concrete infrastructure metrics of direct benefit to the user.

3.6.1. *Interacting with a User*

In the context of metascheduling, access to grid resources is typically handled through a job submission interface, such as a web-based portal [50] or a command line interface, where the user is requested to supply job execution parameters. Parameters requested depend on the scheduling engine employed behind the submission interface and can range from as few as the application name and job input files to as many as an individual application supports (*e.g.*, requirements for number of processors employed,

amount of memory needed, speed of data transfer, etc.). As discussed in Section 2.5 and Section 3.2, typical metascheduler accepts user input and acts on it. However, such an approach assumes the user knows available alternatives and major factors that enable the manipulation of such alternatives. In order to alleviate the user from having to make these assumptions, work presented thus far in this dissertation can be applied in a way that enables focus on an individual user and their current needs. Rather than providing a generic job submission interface and requiring a user to provide information that they think is most suitable for effective job execution across currently available resources (*e.g.*, number of processors to execute a job across), the aim of this part of the dissertation is to build on information collected and delivered by AIS to turn the roles around. In such context, *the user is provided with job execution options* without requiring much job-specific information. Through such an approach, the user becomes aware of the alternatives and accepts the technology rather than being bogged down with the low-level, operating details.

This is realized through an effective presentation of the job execution options generated as part of OptionView application (fully described in Section 5.3) where the user is capable of observing a set of tradeoffs and then makes the final decision regarding their job submission. Consequently, this approach has the direct function of informing the user of choices currently available to them, thus allowing them to examine tradeoffs regarding their current situation. Calculating and composing such a set of tradeoffs in an application-specific fashion (based on information available in AIS), embodies notions behind application-oriented while realizing user-oriented metascheduling. This is because the interaction between a user and the scheduler is no longer at the low, infrastructure

details level, but it is rather at the user level where individual job execution options are presented to the user in applicable terms, thereby allowing the user to analyze available tradeoffs and make an informed decision regarding their job execution.

Overall, the focus of this approach is on an individual job and an individual user. For the service-oriented model adopted by the grid, the proposed approach represents an exciting solution because the user is abstracted from architectural details while enjoying maximum system flexibility and support. To deepen the understanding and significance of proposed hypothesis, an analogy to the airline industry is made here: when a customer wants to purchase a flight ticket, a popular option is to visit one of the online flight search engines and provide simple trip information (*i.e.,* departure/destination and travel dates). The system then analyzes many possible routes, companies, layovers, among others. resulting in a self-contained list of travel alternatives. Such alternatives offer the user a choice regarding ticket cost, departure time, flight duration, layover location, etc. In the end, the user can make informed, custom decisions that meet their current needs. For example, if a person is traveling on business and has set meeting times and a tight schedule, they may choose the shortest flight or one that lands at a particular time, irrespective of the cost. On the other hand, if a person is traveling for leisure with their family, they are likely to choose the cheapest alternative and not be so much concerned with travel times. Solution proposed in this part of this dissertation achieves the same for grid job submission, where the user is informed of their job execution options without having to poses detailed knowledge about grid environments or applications.

3.7. Realizing Described Approach

In its entirety, the described approach can be generalized into a process required to deliver an application-oriented metascheduling environment to the end user in a user-oriented fashion. In order to provide a solution that can be applied in a variety of scenarios and for a variety of applications, there is a need to decouple application-specific components from the general purpose components (Section 3.5). In addition, there is a need to enable flexibility regarding delivery format of user-oriented scheduling, as described in Section 3.6 (*i.e.*, single/multi objective optimization, tradeoff presentation).

With the availability of information within AIS (Section 3.5) and a specific goal set for the metascheduler, a general purpose metascheduler can be customized with the addition of an application-specific wrapper, a plug-in, or a metascheduler adapter to deliver desired functionality. Examples of such a wrapper are typically custom scripts or small programs that are capable of interpreting application-specific information available in AIS (*e.g.*, divide and distribute the job input data based on the data distribution model supported by the application). Development of such wrappers represents an efficient approach to tiered development of an application because it also enables code and software reuse. Such an approach is well accepted and supported in the software engineering community through availability of software development patterns such as the adapter pattern or the façade pattern [183]. Furthermore, the application-specific wrapper implements desired type of user-oriented scheduling or processing of metascheduling results (*i.e.*, single/multi objective optimization, tradeoff presentation) and thus provide full support for application- and user-oriented metascheduling. Figure 32 depicts a schematic model of how this was achieved in the presented approach.

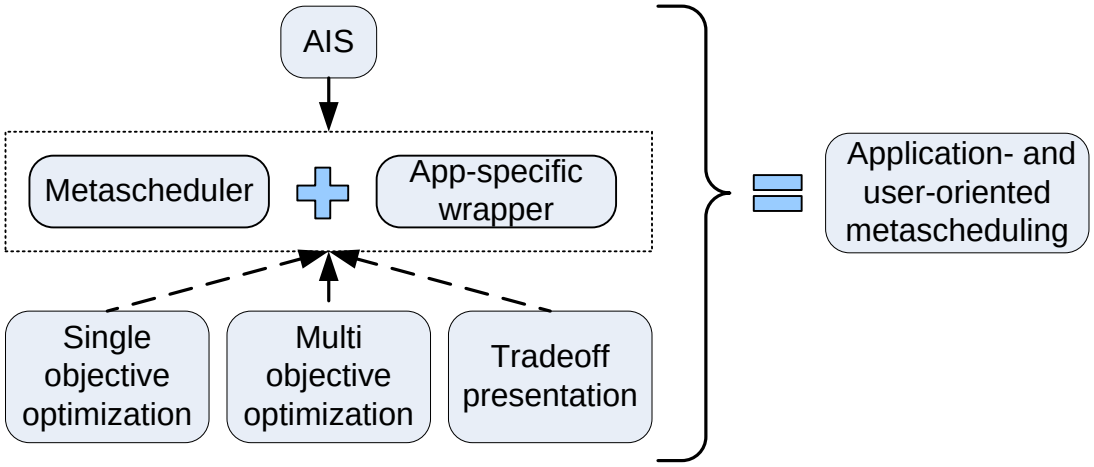


Figure 32. A general model of support mechanisms for delivering application- and user-oriented metascheduling solutions that can range in level and type of user support.

In this dissertation above model and the approach described in this chapter has been implemented, realized, and validated through development of AIS, development of a general-purpose metascheduler model, development of two application-specific wrappers, and realization of user-oriented metascheduling. Based on the requirements described in Section 3.4 and summarized at the beginning of this section, application-oriented metascheduling is realized development of through AIS (architectural and implementation details of AIS are provided in Section 3.5). User-oriented metascheduling is enabled by extending capabilities offered by the application-oriented metascheduling, as described in Section 3.6 and detailed in Section 5.3. Model of a general purpose metascheduler that understands notions of variable resource performance, the act of resource selection, and the act of data distribution across resources is detailed in the Chapter 4 (Section 4.3). Lastly, implementation and validation details for the two application wrappers from two different domains are presented in Chapter 5 (Section 5.1 and Section 5.2).

4. PERFORMANCE ANALYSIS AND MODELING

Chapter 3 presented rationale (based on the motivation presented in Section 2.8.3) as well as the overall approach of work accomplished in this dissertation. Focus on application- and user-oriented metascheduling was highlighted accompanied by an overview of the general solution. This chapter focuses on enabling application-oriented metascheduling through analysis of application performance. The aim of this chapter, and the outcome of this dissertation, is to explain how to obtain and subsequently utilize application-specific information in order to enable application-oriented metascheduling. Initially, an approach for metascheduling EP class of applications is presented in the form of an EP application execution model and a metascheduler framework (Section 4.1). An example of application performance analysis is presented in Section 4.2, which serves as an example in future application performance analyses. Lastly, in Section 4.3, metascheduling models that support the approach proposed by the metascheduling framework are provided. These models are instantiated and implemented in Chapter 5 where additional validation and results of application-oriented metascheduling are presented.

4.1. EP Application Metascheduling

With the overall aim of providing a solution for application-oriented metascheduling, this section presents a general purpose metascheduling approach for EP class of applications. A finer taxonomy of EP class of applications is presented (Section 4.1.1) followed by an EP application execution model (Section 4.1.2); derived results are

captured in a general purpose metascheduling framework for EP class of applications presented in Section 4.1.3.

4.1.1. *EP Application Taxonomy*

With the aim of enabling and providing insight into the metascheduling procedure of EP class of applications (relevant considerations were described in Section 2.8), this section presents a finer taxonomy of the EP class of applications. Taxonomy is presented in order to enable deeper and more targeted discussion with respect to the purpose of grid job metascheduling and metascheduler development. This taxonomy captures major characteristics regarding data distribution of an EP class of applications. Presented taxonomy is divided into three classes, as follows:

1. *Class I – Embarrassingly parallel with homogeneous tasks*: this type of EP applications is characterized by having input data set of approximately equivalent size (hence, tasks experience homogeneous execution times, within a small delta, on a homogeneous resource). Input data size can be defined relative to the application, with examples being physical file size, number of lines in the input file, or number of iterations per task. The scheduling approach for this type of applications largely folds into analyzing resource performance for the given application, followed by resource selection and assignment that maximizes overall job objective(s). Resource selection and task instance assignment (*i.e.*, online scheduling) can be effectively done even during job's execution, as required by a change in resource availability or by resource failure. A formal representation of this type of application is provided in Equation (1) where J refers to the job as a

whole and t_i refers to an individual task. Function $size(t_i)$ used in this Equation refers to the amount of data assigned to task t_i.

$$J = \{t_1, t_2, \dots, t_n\}\ and\ size(t_i) \cong size(t_j) \forall i, j \in 1..n \quad (1)$$

2. *Class II – Embarrassingly parallel with heterogeneous tasks*: this type of applications is characterized by having input data set that are heterogeneous in size or content when compared to one another. Although the scheduling approach for this type of application is largely the same as the previous one, there is a need to perform subtle load balancing between task instances. This may be achieved through dynamic task assignment or resource selection at the job level by creating a job execution plan that simultaneously considers and accounts for task heterogeneity. Realizing job load balance and thus achieving effective scheduling for this category requires deeper understanding of application execution patterns, associated input data influence on application's execution time, and application-resource relationship. Such understanding increases scheduling potential because application-intrinsic properties can be more closely matched to resources' capabilities, thus improving overall job characteristics. Dealing with failed task instances or changes in resource availability within this category increases the difficulty of achieving load balance. Therefore, metascheduling application jobs within this category is more difficult than the homogeneous task category. More formally, this category can be represented as follows:

$$J = \{t_1, t_2, \dots, t_n\}\ and\ size(t_i)\ is\ variable\ \forall i \in 1..n \quad (2)$$

3. *Class III – Embarrassingly parallel with moldable data*: this type of EP applications is characterized by having a single, large input data set that can be

freely divided and distributed. As result, a variable number of execution tasks, each with possibly heterogeneous input data size, may be created. This category of EP applications is similar to the traditional moldable jobs [184] in that it can be divided into a variable number of tasks. However, the distinction made here deals with the data accompanying each task. Unlike the traditional moldable jobs where data is not explicitly discussed and can thus be assumed to be homogeneously divided among tasks, in the current case, the initial input data can be divided in any fashion supported by the application. As a result, the input data division process is generally application specific and must comply with application's requirements regarding location of data splitting; examples for textual input are: split at end of any line, split at end of any paragraph, or split at a special character. For Monte Carlo methods or Genetic Algorithm applications, an example of data division can be an iteration number. Evidently, scheduling for this type of applications requires an application-specific module for the data division. From the scheduling perspective, the major difference with this class of applications is that rather than matching resource's capabilities to tasks' requirements (as the case is with the previous two classes), tasks' requirements can be created to meet resource capabilities. In order to do so, resource capabilities in terms of a given application need to be understood. Overall, in grid environments with heterogeneous resources, this model offers a shift in scheduling approach and a large potential regarding job execution options; through creation of a job execution plan, maximization of resource utilization can be achieved with significant reduction or even elimination of load imbalance. As the case is with

the heterogeneous tasks though, resource failure may require changes to the job plan. This increases the difficulty of minimizing load imbalance. In practice however, factors such as job input data size or application memory requirements often restrict the number of tasks that should effectively be generated. It is one of the goals of job scheduling, and the described approach, to derive such limits.

Although class III of EP applications is sufficient to model either of the two earlier classes (similarly, class II can be used to represent class I), presented taxonomy is necessary because it transitions into taxonomy of metaschedulers and the type of metascheduling individual metaschedulers are capable of performing. Additionally, because different scheduling approaches, techniques, and considerations apply to individual application classes, future implementations of metaschedulers that will target specific class of applications can focus on the specifics of an individual class. Parallel to the ideas of component frameworks [52], if a metascheduler is designed to handle only Class I type of applications, there is no need to build in a large, complex and heavy weight structure needed to support either of the other two classes if such functionality will never be used. Therefore, we are convinced that there is a need to define the above specified taxonomy and thus divide the generic class of EP applications more adequately.

4.1.2. *EP Application Execution Model*

Aiming at providing an applicable methodology and a framework for metascheduling EP class of applications that recognize considerations of Section 2.8.3, this dissertation focuses on metascheduling the Class III category of EP applications. Such focus provides the most general solution, the most widely applicable case, and maximum potential. In this context, we define accompanying job metascheduling as the

problem of selecting available resources for execution, distributing the input data to meet resources' capabilities, and assigning generated task instances to selected resources.

An instance of an EP application is referred to as an EP job J and it is represented by a set of tasks t_i that work toward a common goal: $J = \{t_1, t_2, \ldots, t_n\}$. Because of the heterogeneity of grid resources, individual tasks t_i comprising the job J are likely to exhibit heterogeneous runtime characteristics. Figure 33 presents an example where a 10,000 query input file was divided into 17 evenly sized chunks and then individual chunks were submitted across four different grid resources for execution. As can be seen in the figure, runtimes of individual tasks assigned to any one resource are approximately the same. However, runtimes of tasks assigned to different resources vary greatly. In the EP application execution model, the job is considered complete only after the longest running task has completed. Therefore, in order to achieve maximum performance for a job, load imbalance across tasks needs to be minimized while resource utilization is maximized. To achieve such job execution characteristics, factors that affect runtime characteristics of a task need to be understood (*e.g.,* from Figure 33, why is 2.8 GHz Intel Xeon only 25% faster than the 3.2 GHz resource although it has twice the number of processing cores and a slightly slower processor?).

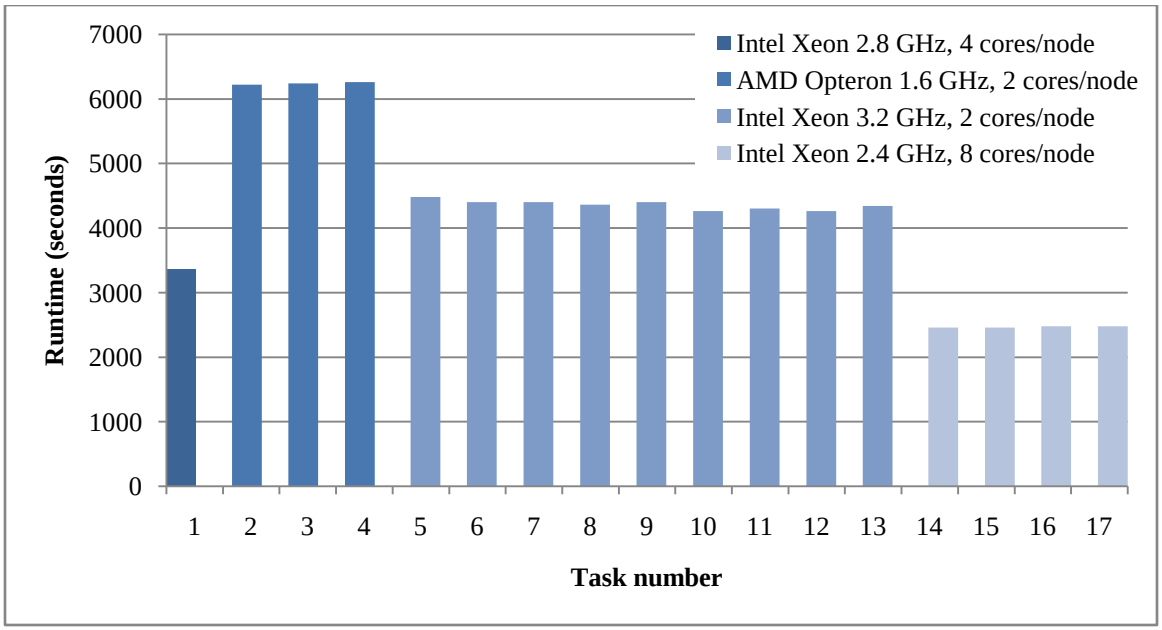


Figure 33. Heterogeneity of task runtimes for a job that is executed across heterogeneous resources. Tasks assigned to individual resources exhibit comparable runtimes but runtimes of tasks assigned to different resources vary significantly indicating the impact a resource can have on task's (and in turn, job's) runtime.

The following are factors affecting runtime characteristics of a task t_i [14]:

- d - the task input data
- r - task execution resource
- p - task invocation parameters

As a result, the runtime characteristics C of a task t_i are a function of the three factors:

$$C(t_i) = f(d, r, p) \quad (3)$$

Understanding and controlling how these factors cumulatively affect task runtime characteristics is an example of task parameterization introduced in Section 2.8.3. Controlling individual tasks that comprise a job leads to a fine level of control of the job and thus the ability to realize desired objective from the perspective of a job. More specifically, task parameterization leads to job parameterization. Task parameterization is

defined as understanding and selecting the task parameters (*i.e.*, user controllable and application dependent options that can be changed when submitting a task, such as the number of threads employed or algorithm used) that are algorithm, input data, and resource dependent. Job parameterization is then defined as coordination and control of individual tasks (and relevant factors) in such a fashion that a desired objective is realized (*e.g.*, minimize runtime, maximize accuracy).

For the objective of job runtime minimization, the aim of job parameterization is to minimize the load imbalance across tasks comprising the job. Achieving such a goal requires the metascheduler to coordinate resource capabilities, match those to application's observed potential, realize appropriate data distribution, and finalize the process through individual task parameterization. The results of a job metascheduling action is a job plan comprising of a set of heterogeneous tasks whose interactions and execution characteristics are simultaneously understood, balanced, and coordinated (see Figure 34 for an illustration).

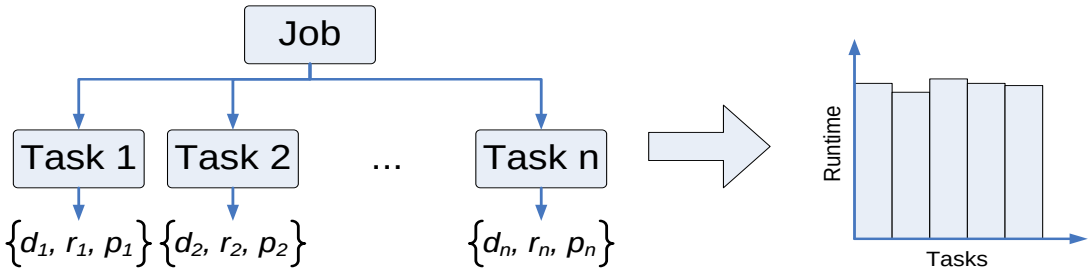


Figure 34. Illustration of the job parameterization process aiming at minimizing load imbalance

The process of job parameterization focused on runtime minimization objective is realized through minimization of load imbalance while maximizing resource utilization. This process can be formalized as follows: given a job J with a single input D of size

$size(D)$, we may create n tasks $t_i \in J$ and $J = \{t_1, t_2, \ldots, t_n\}$ such that $size(D) = \sum_{i=1}^{n} size(t_i)$ holds ($size(t_i)$ is defined as size of the input assigned to a task t_i). Let R be the set of available resources. Each resource $r_j, 1 \leq j \leq |R|$, has capacity $c(r_j)$. Although capacity of individual resources is considered application, data, and parameter dependent, in this discussion it is considered constant. Let E (where $E \subseteq R$) be a set of resources selected for executing job J. Function $resourceUtilization\ (J)$ then defines a policy stating that full capacity of each resource in E is consumed by the job J (*e.g.*, if x CPUs are available on r_j, all x CPUs are consumed by the task assigned to the given resource). Furthermore, two functions are assumed existent, $estTime(t, r)$, which provides an estimate of runtime for a task t on a resource r, and $estCost(t, r)$, which provides an estimate of cost for executing task t on resource r. $loadImbalance\ (J)$ is a defined as $standardDeviation\big(estTime(t_1, r_1), estTime(t_2, r_2), \ldots, estTime(t_n, r_n)\big)$ is minimized, where r_i indicates a resource to which tasks t_i are assigned.

A function $plan_J$, is then defined, which generates the set of tasks $\{t_1, t_2, \ldots, t_n\}$ and parameterizes each task t_i according to Equation (3) so that $loadImbalance\ (J)$ is small and $resourceUtilization\ (J)$ is satisfied. Single execution of the function $plan_J$ provides a single job execution option $o_l, 1 \leq l \leq m$ for executing given job J. $O = \{o_1, o_2, \ldots, o_m\}$ defines the job option space as a set of all execution options generated for the job J.

As an example of the above formulation, consider a BLAST job where 32 input queries need to be processed. Assume two resources r_1 and r_2 are available; r_1 has 6 slots (*i.e.*, nodes, cores) available while r_2 has 2 slots available. Invocation of the function $plan_J$ (*i.e.*, the metascheduler) will result in creation of a job plan consisting of two tasks, t_I and t_{II}. The two tasks will be assigned to the two resources and parameterized to

utilize 6 and 2 slots on r_1 and r_2, respectively. Furthermore, the 32 input query file will be divided between the two tasks in such a fashion that estimated runtimes of individual tasks are approximately the same. For example, if resources were homogeneous when compared to one another and all else is constant, r_1 would be assigned 24 queries while r_2 would be assigned 8 queries.

4.1.3. *EP Application Scheduling Framework*

In order to implement function $plan_J$ described in previous section and provide accompanying functionality, as part of this dissertation, a general framework for effective metascheduling of the EP class of applications has been devised. Figure 35 provides a schematic overview of the devised metascheduling framework. This figure is also a more detailed version of the earlier discussed Figure 32. As can be seen in the figure, effective EP application scheduling is a two-step process. In step one, the application needs to be analyzed, yielding needed application-specific information. In step two, the derived information is exploited to plan execution of user jobs in an application-specific fashion. Once a job plan has been derived, the control is transferred to a job submission engine for execution on grid resources.

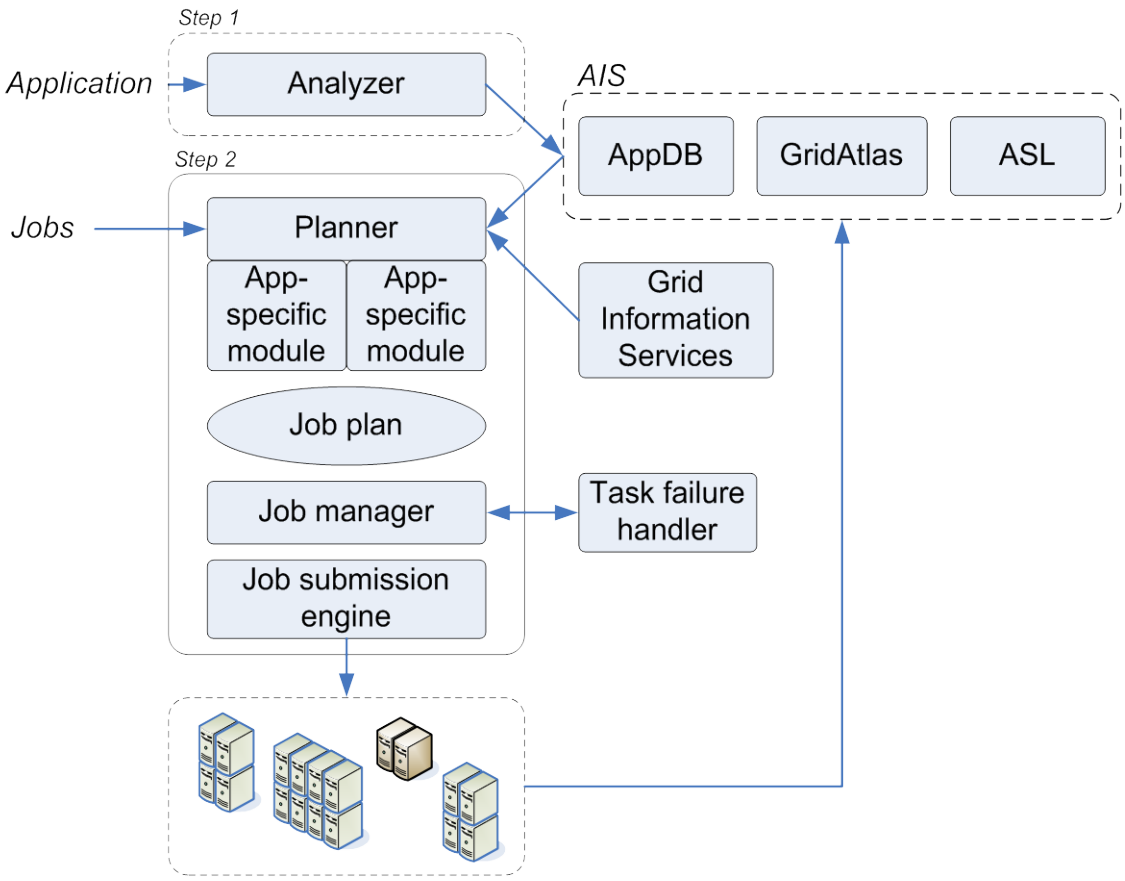


Figure 35. Devised two-step EP application metascheduling framework

Analyzer and planner are the two main components of the described framework. Analyzer operates at the application level by performing analysis of an application. Such analysis is targeted toward deriving application-specific relationships between input data, input parameters, runtime modes, and execution resources (*i.e.*, understanding factors presented in Equation (3)). Currently, the application analysis is a non-automated process that requires manual analysis of the application execution patterns. Tools and services devised as part of AIS [173] are used to store and help this analysis process. In the simplest format, application analysis involves benchmarking of various resources enabling relative comparison of those resources for later job execution (sample benchmarking tools: [16, 26]). In the most complicated format, application analysis

involves development of a mathematical model that describes application execution patterns and dependencies.

Once an application has been analyzed, the derived information can be leveraged to improve job performance. Job planner represents a general purpose metascheduler that understands notions of variable resource performance, the resource selection process, and variable data distribution. In other words, the job planner represents a general implementation of the $plan_J$ function. In order to realize application-oriented metascheduling, but also support the decoupling of the metascheduler from the application, an application-specific module (*i.e.*, wrapper or plug-in) needs to be developed that utilizes results from the analysis performed in step one and provide it to the general purpose job planner. Complementing the general purpose planner with the application-specific module enables generation of an application-specific job plan and thus application-oriented metascheduling. Once a job plan has been created, it is propagated to the job manager and a job submission engine. The job manager component is an optional component than can implement application-specific logic for partial task failure or runtime changes in resource availability.

An example of possible functionality of the job-manager component is support for job portability. Without such functionality, once created, a job plan is not modified even if better resources become available. Instead, the job-manager could modify or re-create the job plan and take advantage of the resource availability change. Such functionality, in order to be efficient, requires application check-pointing and represents possible extension of this work.

4.2. Performance Analysis

This section provides an example of application performance analysis, namely realization of step one from the framework presented in the previous section. The section is divided into three subsections corresponding to the factors defined as affecting runtime characteristics of a task (see Equation (3)). Components considered include: hardware and software configurations of individual resources focused around number and type of processors (factors r and p from Equation (3)), influence of number of threads on job runtime as well as resource utilization (factor p from Equation (3)), and influence of query input file statistics on job execution time (factor d from Equation (3)). Characteristics of relevant job runs are stored in respective AIS services and enable observation of changes in patterns in application execution. Technical details of resources used during this performance analysis are available in Appendix C with resources availability listed in Table 2.

Table 2. Availability of resources used during experimentation.

Name	Everest	Olympus	Coosa	Wave	iBook	Dual Opteron
# nodes	32	128	128	1	1	1
Total cores	64	256	256	2	2	4
BLAST ver.	v 2.2.11	v2.2.11	v2.2.16	v2.2.16	v2.2.16	v2.2.16

4.2.1. *Task Input Data Influence*

First, we study the impact of input data on runtime characteristics of a task. Typically, runtime characteristics of a task are primarily characterized by the amount of input data that needs to be processed by the task. For the case of BLAST application, the amount of input data is measured by the number of queries that need to be processed by a task. We analyze performance of BLAST with respect to the number of queries that need to be processed. Figure 36 presents the impact on task runtime as the number of queries is

varied from 32 to 256 (in increments of power of 2). The results are obtained from the Everest resource using a single process and two threads (technical resource details can be found in Appendix C). The database used was *yeast.nt* (12MB) and each of the individual queries was 393 bases long (*i.e.*, the same query was used a specified number of times). Results show that execution time of BLAST application is linearly proportional to the size of the input it needs to process.

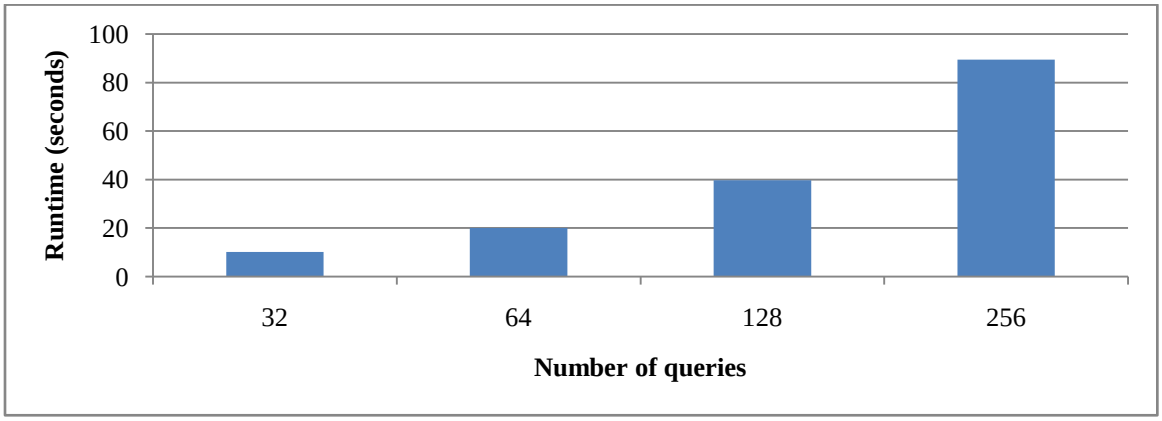
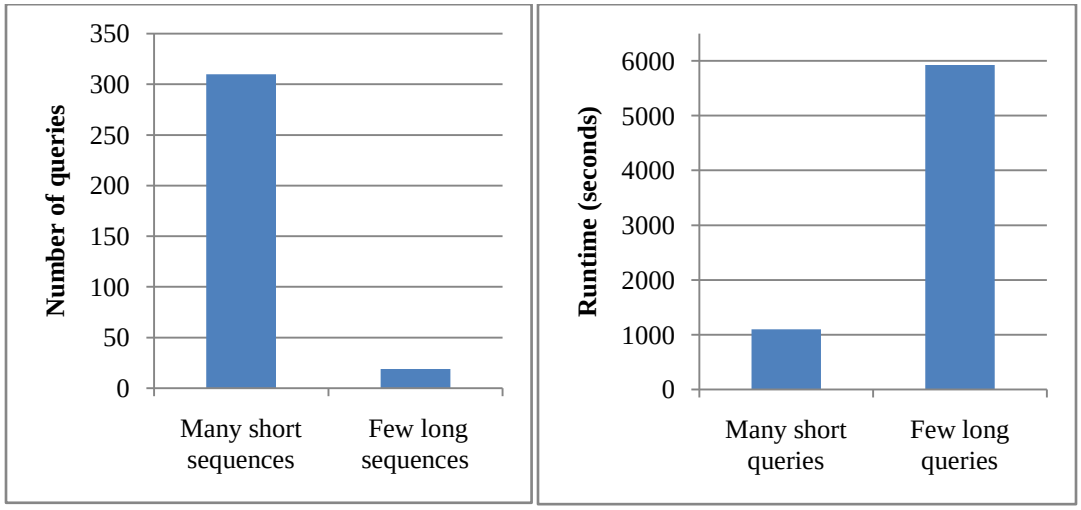


Figure 36. Impact of number of queries used as input for a BLAST task on task's runtime

Depending on the application, structure of the input file may also affect runtime characteristics of a task. In case of BLAST, individual queries may be of different lengths. In order to analyze impact of query length on task runtime, we performed additional tests where we kept the file size as even as possible but varied the number of queries in the input files. We created a file with only 19 long queries with average length of 3946 bases and a file with 310 short queries of average length 92 bases (see Figure 37a). Individual queries were selected from the VBRC database based on their length. Each file was approximately 77 Kb in size.

The runtime results of executing the two input files are shown in Figure 37b (using Everest resource and the *nr* database (1.6GB)). As can be seen, the length of the query

has significant impact on runtime characteristics of a task. Executing the set of long queries exhibited approximately five-fold slower runtime than executing the set of short queries. Although presented analysis is not conclusive in terms of direct correlation between individual query length and task runtime, it signifies the potential impact query length may have on task runtime. The conclusion is that, in order to be able to effectively compare performance of resources with respect to the input data, the length of queries must be considered alongside the number of queries.



(a) *(b)*

Figure 37. Physical characteristics of two input files used to test impact of query length on BLAST application runtime: (a) one file has a large number of short queries while the other has a small number of long queries and (b) runtime of tasks parameterized with corresponding input files.

In the above analysis, BLAST was used as an example but it does not represent the only application where input data characteristics affect runtime of a task. Such examples are plentiful among computer science applications. Quicksort sorting algorithm [185] is an example. Quicksort has average complexity of $\Theta(n \log n)$ where n is the number of elements to sort. However, if the data provided as input to the Quicksort algorithm is

already sorted and pivot is not randomized, then the algorithm takes longer time to sort the data as compared to unsorted data (specifically, complexity is $\Theta(n^2)$).

4.2.2. *Task Execution Resource Influence*

With the heterogeneity of resources present in grid environments, there is a need to understand how runtime characteristics of a task are affected by the resource selected for task execution. There are several methods for determining capabilities of a resource, namely:

- Calculate theoretical peak performance of a resource based on the clock speed of the processor
- Use a generic benchmark tool, such as the SPEC benchmark discussed in Section 2.7.8
- Use an application-specific benchmark (this may include analysis of historical runtime characteristics of a given application, executing selected application with a smaller input data set prior to the job submission, use of application skeletons to estimate application performance [137], or use of application-specific benchmark tool (*e.g.*, [130]))

Because of the application-resource dependency explained in Section 3.3, in addition to previous research [186], only the application-specific benchmarks offer a true performance of a resource as it pertains to the given application. It is thus desirable to analyze behavior of runtime characteristics of the application under question as it moves across resources available in the grid. Enabling such application introspection is one of the aims of AppDB presented in Section 3.5.2. Nevertheless, when such performance data is not available, general purpose approaches can be used (with possibly reduced

level of accuracy). For example, study [27] shows how performance of sequence alignment applications on a given resource can typically be estimated within 10% of the generic benchmark value for the given resource. Therefore, calculating theoretical peak performance of a given resource and using it to estimate available resources' relative performances can lead to significantly better metascheduling approach than blindly submitting jobs across available resources.

Figure 38 presents results of executing the same 32 query input files across a range of resources. Individual queries were randomly selected from the Viral Bioinformatics Resource Center (VBRC)[7] database. The VBRC database contains the complete genomic sequences for all viral pathogens and related strains that are available for about half a dozen of virus families. All tasks performed the search against the 1.6GB *nr* database. Technical details regarding used resources can be found in Table 2. As can be seen from the figure, individual resources exhibit significantly different runtime characteristics for the same task. Availability of such runtime data enables a metascheduler to 'understand' capabilities offered by individual resources and more adequately distribute task workloads (as defined by function $plan_J$ in Section 4.1.2).

As an example of the importance of application-specific benchmarks, one can consider the difference in performance between runtime of tasks executed on Olympus resource when using two different versions of the Linux kernel (see Figure 38). The runtime difference of the two tasks is approximately 30%! Furthermore, using the same Linux kernel but varying version of the BLAST application shows difference in tasks' performance on the order of 5%. Although these values may change from one resource to the next, they point at the impact resource configuration may have on runtime

[7] http://www.biovirus.org/

characteristics of a task. At the same time, the theoretical peak performance of the given resource is constant. This is an example of the benefit of using application-oriented benchmarks over standardized ones.

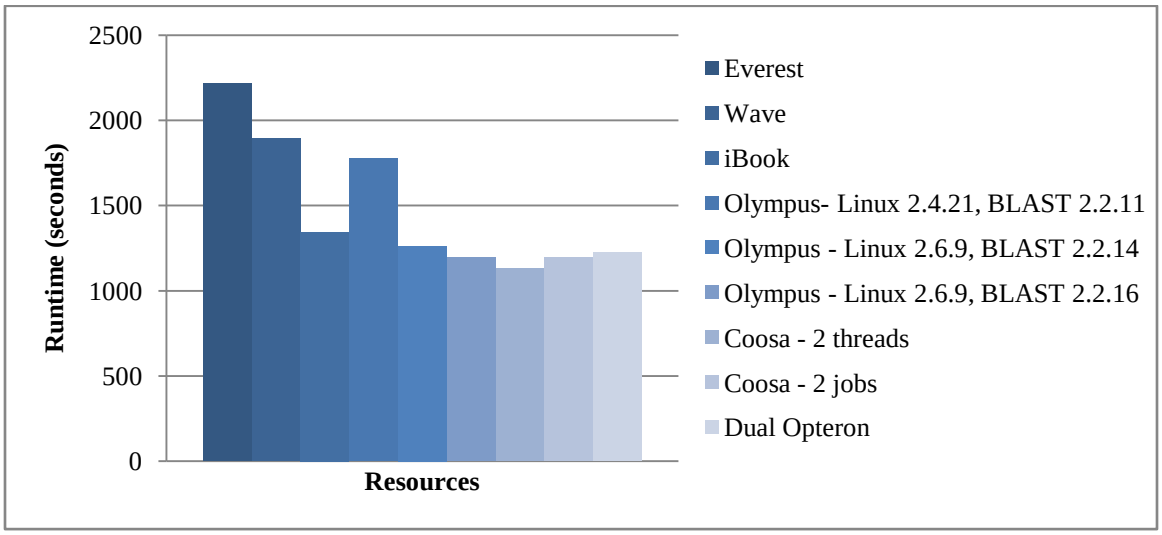


Figure 38. Comparison of BLAST execution times across resources. Architectural details of machines used are provided in Appendix C with resource availability listed in Table 2. Presented experiments searched 32 queries randomly selected from the VBRC database against the nr database (1.6GB in size).

4.2.3. *Task Parameters Influence*

Applications are oftentimes controlled through various arguments, options, and values associated with those. For example, the number of threads to instantiate when invoking an application, value for transitional probability in a Hidden Markov Model application, or step size in a Monte Carlo simulation. Depending on the application, the choice of a parameter and associated parameter's value may affect runtime characteristics of a task. The change may be observed in terms of result accuracy or task runtime.

In case of BLAST, runtime characteristics of a task may be manipulated by exploiting thread-level parallelism when comparing queries to the different parts of a sequence database. Multi-threading support within the BLAST application is supported

through the '-*a*' option. Figure 39 presents runtime results of the same task configuration as in the previous section across three different resources but with varying the number of threads employed by each task. As can be seen from the figure, parallelism scales effectively until the number of processing cores available on a given resource matches the number of threads created. Wave and iBook resources contain two processing cores and invoking two threads as opposed to one nearly reduces the runtime of the task by 50%. The Dual Opteron resource exhibits comparable performance for the values up to four threads.

Increasing the number of threads beyond the number of processing cores available on a resource has only minimal impact. This is due to the contention between competing processes. However, small performance improvement may be observed when number of threads is slightly higher than the number of processing cores because of the I/O and computation overlap that can be realized by the operating system controlling the threads.

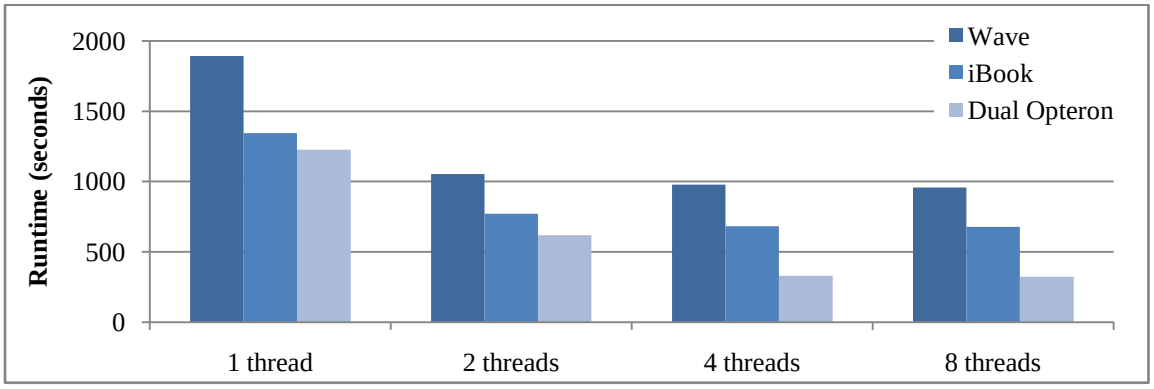


Figure 39. Effect on BLAST task runtime characteristics when varying number of threads option across resources. Application scales efficiently to the point where number of threads matches the number of processing cores on a given resource.

Depending on the scheduling policy of individual resources, information about the processing capacity of a node may be published in terms of the number of processors or number of cores. For example, a node on a resource may have two quad-core processors.

Published information about resource scheduling policy may be used to discover such a detail. There is a question whether scheduling (or instantiating) two separate processes (*i.e.*, one per processor) and parameterizing each process to start four threads may be better than instantiating a single process and parameterizing it to start eight threads. Figure 40 points at runtime differences for the two approaches. The two tasks executed a 128 query input file against the *nr* database on a resource made up of two quad-core Intel 2.33 GHz processors. As can be seen in the figure, creating a single process with the number of threads matching the number of processing cores resulted in a slightly better runtime performance (note that the y-axis in the figure does not start at zero). This difference could be attributed to the fact that for the single process case only one copy of the database needs to be read and loaded into memory whereas for the two process case two copies of the database need to be read and loaded into memory.

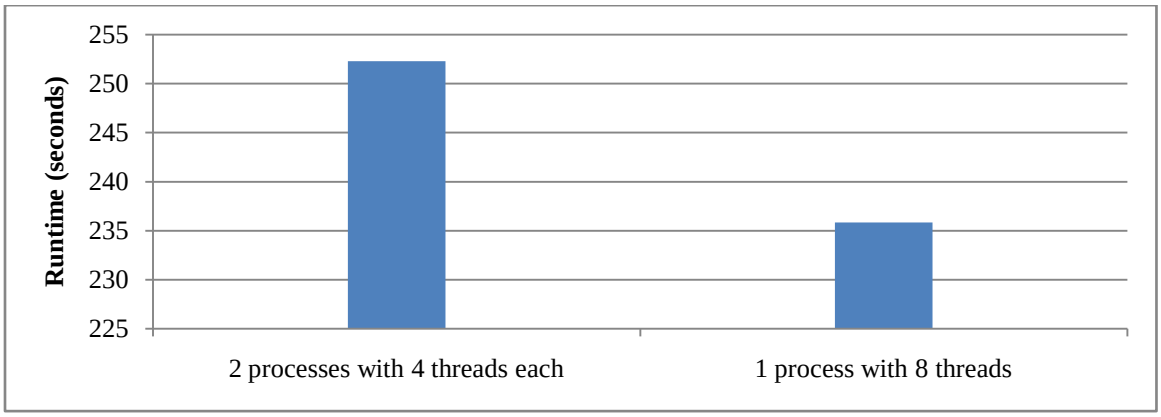


Figure 40. Runtimes of two parameterizations of the same BLAST task on one node. Tasks used 128 queries as input and searched against the 1.6 GB nr database (note that the y-axis in the figure does not start at zero).

Once again, the BLAST example shown does not represent the only application whose runtime characteristics are affected by parameters and corresponding values used when invoking a task. For example, HPL [139] is the most popular benchmark used to

rank high performance clusters[8]. At the same time, HPL is characterized by a variety of arguments and argument values that significantly affect runtime characteristics of a task [187]. Furthermore, arguments are often closely tied to the resource at hand [188] and thus require significant understanding of resource capabilities and application requirements.

4.2.4. *Job Parameterization*

The previous three sections focused on analyzing influence of individual factors comprising a task on the task's runtime characteristics. As described in Section 4.1.2, job parameterization operates on the level above task parameterization and it aims at coordinating individual task factors so that the job as a whole can realize set objective(s). At the same time, job level parameterization may need to consider and include parameterization factors beyond those controlling runtime characteristics of individual tasks. Examples include making a decision on how many tasks to invoke or how to distribute input data across instantiated tasks.

For the case of BLAST, the built-in thread-based parallelization method works well for a single SMP machine but is limited in terms of scalability. In order to gain additional performance benefits, at the level of job parameterization, multiple tasks may be created and distributed to multiple nodes of a resource. Results shown in Figure 41 present runtimes of BLAST jobs executing two threads per node (because each compute node has two processing cores available) and using 1,024-query input file (queries were randomly selected from the VBRC database) against the *nr* database (1.6GB). The same set of jobs is executed across three resources. The performance data presented shows a consistent speedup across resources as multiple tasks are instantiated. This indicates at the potential

[8] http://www.top500.org/

to parallelize execution of BLAST jobs beyond the support offered within the application through the thread-level parallelism.

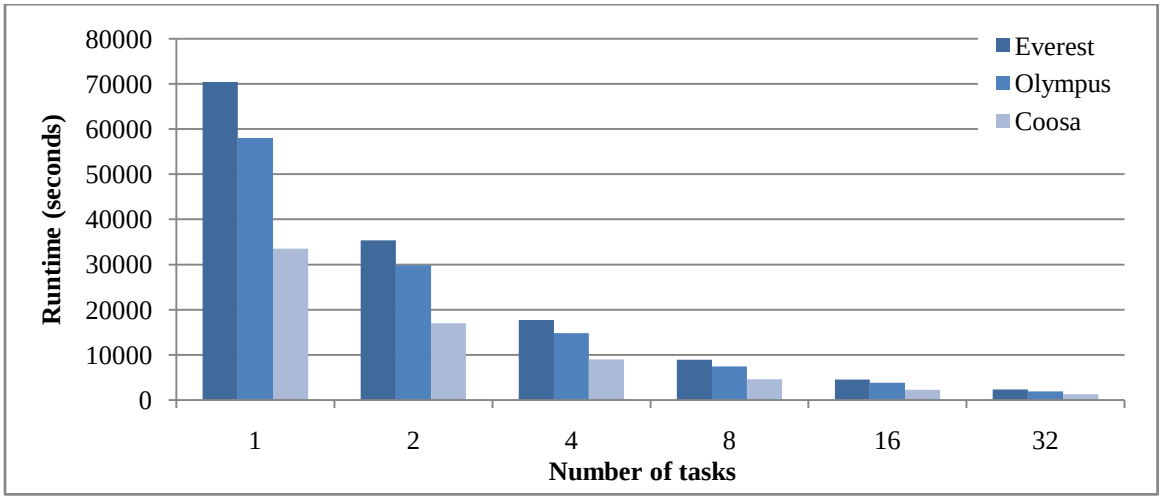


Figure 41. Runtimes of a BLAST job using 1,024-query input file against the nr database (1.6GB) when the workload is divided into specified number of tasks across multiple nodes. The same job was executed on three resources to analyze variation in performance across resource architectures. Each task initiated two execution threads.

If the metascheduling objective is cost minimization for example, the metascheduler needs to consider job execution efficiency alongside runtime minimization. As shown in Figure 41, increasing the number of tasks that are created within a job decreases the runtime of the job; however, introduced parallelism may involve overheads regarding tasks' execution. Such overheads may affect resource utilization and job efficiency. For the job cost minimization metascheduling example, such overheads can become important.

In order to understand this overhead effect, the following should be considered. Job cost C is given by $C = p \times T_p + T_o$, where p is the number of processors used, T_p is the job runtime for p processors, and T_o is the overhead included with the computation [4]. The goal is to achieve $p \times T_p \approx T_s$, where T_s is job runtime for the sequential mode of

application execution. T_s is further defined as $T_s = T_{seq} + T_{comp}$ leading to $T_p = T_{seq} + \frac{T_{comp}}{p}$, where T_{seq} is sequential time required to setup a process and T_{comp} is the process computation time. Because of the T_{seq} component, it can be concluded that the efficiency of a job must decrease with an increase in number of processors used. Implications of this analysis are that cost will be minimized only when a single processor is utilized. However, such execution will result in longer runtime. Therefore, there is a need to trade off some efficiency for reducing runtime.

Different computer architectures are implemented in different ways and they handle parallel application executions in different ways (*e.g.*, caching procedures, memory management). It can thus be important to analyze performance of an application across resources in terms of execution efficiency as well as speedup. Figure 42 provides results of architecture-dependent job efficiency experiments when executing a BLAST job for a variable number of tasks. Jobs executed used 1,024-query input file against the *nr* database (1.6GB). Based on the presented results, Everest resource (AMD based architecture) shows a faster decrease in job efficiency but remains largely constant when 4, 8, and 16 tasks are created. On the other hand, Coosa resource (Intel based architecture) experiences a continuous but more consistent decrease in job efficiency. The results show that even though creation of multiple tasks leads to significant reduction in job runtime (see Figure 41), the efficiency of job execution suffers (see Figure 42). Moreover, from the perspective of job efficiency and minimizing job cost, if using Everest (*i.e.*, AMD based) resource, it is more beneficial to utilize 8 or 16 nodes than 1, 2, or 4 because runtime of the job is considerably reduced (see Figure 41) while the job efficiency remains approximately constant. Such observations indicate at the importance

of job parameterization and a need for collecting ample information regarding application execution characteristics. AppDB service from AIS (described in Section 3.5.2) provides needed support to capture retrieve such data. Although concrete values regarding parameterization of individual jobs and resources may differ significantly, potential for described behavior is important to note and incorporate into a metascheduling model.

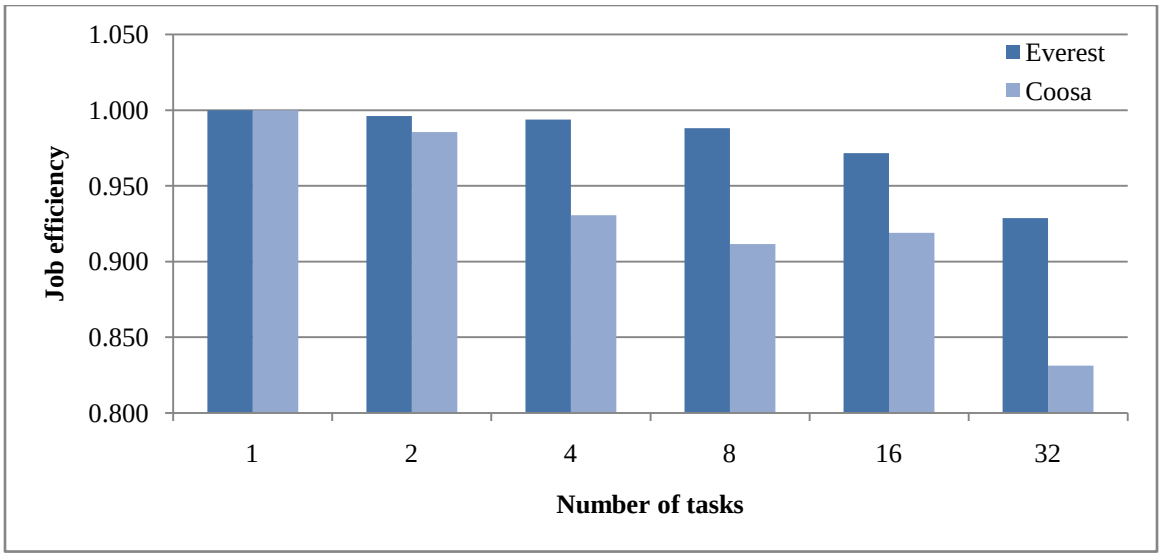


Figure 42. BLAST job execution efficiency across two resources differing in their architectures. Efficiency of Everest (AMD based) resource dips faster but remains constant longer while the Coosa (Intel based) resource shows continuous decrease in job efficiency.

4.2.5. *Performance Analysis Observations*

Based on the performance analysis presented, several observations regarding application execution characteristics are made in this section. Although some of the observations are of general nature, the focus is on how to maximize performance of BLAST jobs in heterogeneous environments.

Based on the presented data, the following components can affect task and job runtime characteristics:

- Resource architecture

- Speed and number of CPUs/cores
- Input data properties

In terms of the architecture, a trend was observed where the speed (*i.e.,* clock frequency) of a processor in SMP (*i.e.,* Intel) based machines compared to NUMA (*i.e.,* AMD) based machines needed to be much higher to provide comparable performance. Overall, based on the results shown in Figure 38, speed of CPU seemed to provide proportional increase in performance when compared across CPUs within a given architecture.

The number of CPUs or cores used has the most influence on the resulting turnaround time of a job. Increasing the number of CPUs employed provides a continuous and consistent job speedup. When multiple processing cores are available on a system, use of the available thread-based parallelism with BLAST application should be utilized. The number of threads instantiated should correspond to the total number of processing cores available on the system (or node). Instantiating multiple threads within BLAST application is the preferred way of maximizing performance on an individual system (or node) as opposed to instantiating multiple processes on the node.

Lastly, if the input data permits, splitting the BLAST input file should be performed based on number of queries and length of the queries. If the data is distributed using this approach, task (and job) parameterization should be performed to meet capabilities of individual resources aiming at minimizing job load imbalance.

4.3. Metascheduling Models

This section provides a general purpose metascheduling model, namely realization of the first part of the step two of the framework presented in Section 4.1.3. This section is

further divided into two subsections; Section 4.3.1 describes a metascheduling model for homogeneous resources and Section 4.3.2 describes a metascheduling model for heterogeneous resources. The aim of presented models is to provide a general purpose metascheduling solution that can later be instantiated by developing an application-specific module that supplies necessary information to this general purpose metascheduler. In order to support desired features and requirements discussed in Section 3.4, the general purpose metascheduler needs to understand notions of variable resource performance, the resource selection process, and weighted data distribution across task as they are assigned to resources.

4.3.1. *Homogeneous Resources Model*

Overall, parameterization of a job in a heterogeneous environment can be partitioned into two components, the first one being parameterization of the overall job in terms of resource selection and data distribution across tasks and the second one being parameterization of an individual resource upon it being selected. This section focuses on the second step and presents a metascheduler model that can be instantiated when parameterizing a job on any single resource.

In the context of a homogeneous resource, three decisions need to be made:

- How many tasks to create?
- How to distribute data among created tasks?
- How to parameterize each task?

The number of tasks that can be created on a given resource is constrained by the number of Processing Slots (PS), or nodes, available on the system. We refer to this number as n. Although this is a soft constraint, creating more tasks than available PSs

will result in resource thrashing [32] and is thus not considered. The number of created tasks can, however, be smaller than the total number of PSs. As shown in Section 4.2.4, depending on the architecture of a resource, certain job configurations result in improvements in resource utilization. Depending on the objective of a metascheduler, utilizing such information may be of interest when scheduling and parameterizing a job. Because such information is application-specific, it should be implemented at the level of the application-specific wrapper and provided to the general purpose metascheduler. The general purpose metascheduler needs to be able to handle such a request. Therefore, the number of tasks to be created by a metascheduler is provided as a variable at the time of metascheduler instantiation and it is denoted by T (thus supporting variable resource availability).

Once the number of tasks has been determined, there is a need to decide how to distribute the total amount of input data D between the number of tasks T. Because the resource is composed of homogeneous nodes, it is adequate to divide the input across the nodes (and thus the tasks) in a proportional manner. Equation (4) captures this action where d indicates the size of the input that should be assigned to each individual task (size is application-specific and can refer to the number of queries in case of BLAST or physical size of the overall input file):

$$d = \frac{D}{T} \tag{4}$$

Variable d indicates a general size of input that should be assigned to individual task t_i. However, assigning the actual d amount of data to t_i should be implemented by the application-specific wrapper because this is often an application-specific action. For example, based on the BLAST analysis from the previous section, the amount of data

assigned to each task t_i should be guided by the number of queries and the length of individual queries. Therefore, creating a script or a module that, when instructed as to how many data chunks to create and much how much data to assign to each task, it can distribute the provided input data in the appropriate fashion.

Lastly, each task should be parameterized to help realize the desired scheduling objective. Because this is an application-specific action, the metascheduler should understand the notion of task parameterization but the parameter value should be passed to the metascheduler as a variable. In case of BLAST, the parameter value could be the number of threads to instantiate within a task. The number passed would likely correspond to the total number of processing cores available on any one Processing Slot (as per conclusions of Section 4.2.3).

In conclusion, the presented metascheduler model provides a standardized interface for an application-specific wrapper to rely upon. The model supports notions of variable resource assignment, data distribution, and task parameterization. Within an implementation of the model, a general purpose solution to above actions and subsequent duties (*e.g.,* task submission, data staging) can be implemented; however, it is the application-specific wrapper that provides necessary information and performs application-specific tasks.

4.3.2. *Heterogeneous Resources Model*

At the highest metascheduling level, a model is needed that encompasses global job parameterization in terms of resource selection and data distribution. This section presents such a metascheduling model. Presented model represents a general purpose

solution that can be instantiated and it assumes existence of an application-specific plug-in that provides and implements many of the necessary details.

The following notation is used throughout this section:

R Set of normalized performance weights of resources available for executing job J

n_i Number of Processing Slots (PSs) or nodes on resource i

P_i Performance rate of individual PS on resource i

W_i Performance weight of resource i

E Set of normalized performance weights of resources selected for executing job J

T_i Number of tasks assigned to resource i

d_i The size of data chunk assigned to resource i

D The total size of input for job J

The following is the list of actions supported by the derived model:

- Understand heterogeneity of available resources
- Support notion of resource selection
- Understand the possibility for variable data assignment to selected resources

Capabilities of a resource can be measured by its size, and size can be quantified through the number of PSs available. Because of the heterogeneity of grid resources, size should not be the only measurement. Therefore, understanding resource heterogeneity involves accounting for the size of the resource and resource's performance. Furthermore, because of the application-specific relationship, performance of a resource should be measured in terms relevant to the given application. Equation (5) captures the process of resource weight calculation.

$$W_i = n_i \times P_i \tag{5}$$

The performance weight P_i of a resource can be obtained through one of the methods discussed in Section 4.2.2; how such a value is obtained is left up to the application-specific module that instantiates the metascheduling model. Note that Equation (5) exclusively considers all of the available PSs on given resource. However, this does not need to be the case and an instance of the presented metascheduling model can be invoked with a desired number of a PSs. This can be captured in above formulation by replacing the number of PSs on a resource n_i with the number of tasks T_i that wish to be created on given resource (an example is provided later, in Section 5.3).

Once the resource weights are obtained, they are normalized and used to represent the set of available resources whose individual performances can be directly compared (see Equation (6)).

$$R = \{W_1, W_2, \dots, W_m\} \tag{6}$$

Resources can now be effectively compared and the act of resource selection can be performed. Presented model realizes the notion of resource selection based on a certain threshold. The value of the threshold may be interpreted differently for different applications. Some examples are: resource performance is less than some constant, resource performance is less than 50% of the fastest resource, cost of a resource if greater than some constant. Equation (7) captures such functionality:

$$E \subseteq R \; and \begin{cases} R_i \in E_i, R_i \geq threshold \\ R_i \notin E_i, R_i < threshold \end{cases} \tag{7}$$

Lastly, the metascheduling model needs to understand the notion of variable data distribution across selected resources based on the relative resource performance. Equation (8) provides a general formulation for the resource performance relative data

distribution. Computed value d_i refers to the amount of the data that should be assigned to resource i but the data distribution is implemented by the application-specific module to adhere to the requirements imposed by the application. For the BLAST example such requirement implies that division should be performed at the granularity of individual query. Furthermore, the division module should account for number of queries as well as length of queries when performing the data distribution. As with the case of Equation (5), the total number of PSs n_i in Equation (8) can be replaced with the desired number of tasks to be created on resource i.

$$d_i = \frac{n_i}{\sum_{j=0}^{j<|E|} n_j} \times D \times W_i \tag{8}$$

Once the decisions regarding above discussion are made at the level of a job, the scheduling model presented in the previous section can be instantiated to parameterize each individual task on each individual resource. Overall, presented model captures the requirements of a metascheduler to understand and cope with heterogeneity of resources in grid environments. Implementations and instantiations of the presented model can thus rely on availability of essential functionality while interpretation of the application-specific information is implemented in a separate module (refer to Figure 32 for a graphical illustration).

4.4. Reflections on the Approach

Overall, this chapter focused on enabling and realizing goals behind application-oriented metascheduling. More specifically, an approach and a framework for metascheduling EP class of applications was described (Section 4.1), followed by an example of application performance analysis (Section 4.2), and culminating in

description of derived general-purpose metascheduling models (Section 4.3). The described framework for metascheduling EP class of applications is a two-step process consisting of application analysis (and utilization of AIS functionality described in Section 3.5) followed by development of an application-specific metascheduler, which utilizes conclusions derived from the analysis in an application-oriented fashion. Such approach enables realization of benefits resulting from application-oriented metascheduling while providing a well-defined process to follow, thus easing the required effort.

The next topic discussed (Section 4.2) focused on providing an example of application-performance analysis (*i.e.*, Step 1 from the EP metascheduling framework). Individual components of application's lifecycle and factors affecting application execution characteristics were systematically presented and analyzed offering a model for achieving the same with any EP application. Lastly (Section 4.3), general purpose models for metascheduling EP applications were formulated and presented (*i.e.*, Step 2 form EP metascheduling framework). Existence of such models enables application-oriented metascheduling to be realized by simply plugging-in application-specific analysis conclusions into the existing metascheduling framework.

The approach presented in this chapter enables application-oriented metaschedulers to be easily developed. This is achieved through use of AIS, which supports collection and retrieval of application-specific data, and the metascheduling framework, which provides an example and a model for realizing application-oriented metascheduling. With the availability of these key components (*i.e.*, information collection services and an example-supported metascheduling framework), application-oriented metascheduling is

realized. In support of realizing application-oriented metascheduling, empirical analysis and validation of presented approach and derived solutions are presented in Chapter 5, along with the utilization of application-oriented metascheduling to deliver user-oriented metascheduling.

5. REALIZING APPLICATION- AND USER-ORIENTED METASCHEDULING

Chapter 4 provided a general approach and an example for realizing application-oriented metascheduling through analysis of application performance. This chapter focuses on means for delivering application-oriented metascheduling to a user. Initially, the presentation focuses on how to realize application-oriented metascheduling through implementation of application-specific wrappers (Section 5.1 and Section 5.2). Implementations of these wrappers serve as validation scenarios for delivering higher user utility by incorporating resource-application dependencies into metascheduling and enable realization of two-way interaction between a user and the scheduler – the central hypothesis of this dissertation. Next, Section 5.3 focuses on realizing user-oriented metascheduling through utilization of information available within AIS and the results of application-oriented metascheduling. User-oriented metascheduling is realized and validated through simulation in addition to real-world grid resources, showing significant potential and benefit of the proposed methods.

5.1. Bioinformatics Application

In order to complete an implementation of the metascheduling framework presented in Section 4.1.3 (specifically, Figure 35), the general purpose metascheduling model devised in Section 4.3 needs to be complemented by an application-specific wrapper or plug-in to realize application-specific metascheduling. This section presents an implementation of such a wrapper for an application from the bioinformatics domain,

namely BLAST. The developed application was named Dynamic BLAST and it has been accepted by SURAgrid as the grid-enabled implementation for the BLAST application[9].

Dynamic BLAST is a grid enabled version of BLAST; it acts as a wrapper for BLAST that is capable of exploiting grid resources for BLAST jobs. Dynamic BLAST represents an implementation of the general metascheduler model presented in Section 4.3. Within Dynamic BLAST, all aspects of resource selection and data distribution are not only automated but also tailored for BLAST application. Additionally, Dynamic BLAST builds on top of grid tools and standards in order to deliver and support required robustness to succeed across grid environments. More specifically, Dynamic BLAST was developed using Java on top of the Globus Toolkit [35] and has adopted the DRMAA [107] standard for all job invocation operations in addition to GridWay [189] for all job submission activities. A high-level overview of Dynamic BLAST's interaction with grid components is given in Figure 43.

Authentication and authorization are moved outside the Dynamic BLAST where the user is required to have a valid grid proxy before job invocation. Resources are discovered dynamically through Grid Information Service (GIS). GridAtlas is used to monitor application-related parameters on various resources and deliver required information to Dynamic BLAST. Interaction with GridWay is performed through DRMAA API and is used for all job submission and monitoring activities. As discussed in the following paragraphs, Dynamic BLAST handles resource selection and data distribution; however, actions regarding resource allocation, data transfer, and job monitoring are all delegated to GridWay for execution through the DRMAA API. Adoption of GridWay in Dynamic BLAST development was a cornerstone with respect

[9] http://www.sura.org/programs/docs/UABBLAST.pdf

to Dynamic BLAST modularity and portability. Through its support for the DRMAA standard, GridWay provides a well-established and proven platform for high-level grid application development. Before adopting GridWay, the majority of development and maintenance effort within Dynamic BLAST was devoted to low-level resource interactions and upkeep with ever-changing technologies (*e.g.*, pre-WS to WS). Adoption of DRMAA standard within GridWay has even alleviated direct dependencies to GridWay, thus increasing code modularity.

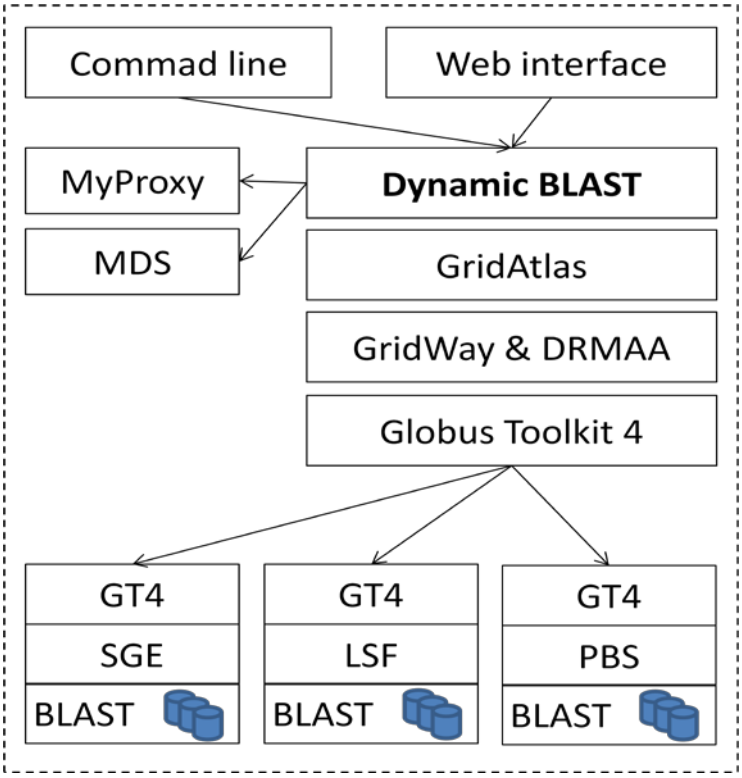


Figure 43. High-level diagram of interactions between grid components and Dynamic BLAST

Based on experience obtained during the work associated with this dissertation, presented architecture of Dynamic BLAST can be seen as a model that is suitable for future application developments. The key aspects of this architecture have shown to be development of targeted and specific components where the application is performing

only the action it was initially intended to perform (beyond standard software engineering techniques such as code modularization and low inter-module coupling). Stated approach implies making extensive use of available tools rather than trying to re-develop existing technologies.

5.1.1. *Dynamic BLAST Architecture*

Analysis of BLAST parallelization methods and grid resource characteristics has led Dynamic BLAST to be internally developed under the master-worker communication model. The master-worker model allows a single process to control the resource selection, data distribution, job submission and parameterization, and job monitoring. Thus, selected application model maximizes execution flexibility, code modularity, and fault tolerance. Going hand-in-hand with the master-worker model, and as described in Section 2.9.1, the original BLAST algorithm can be parallelized by adopting either of the two data distribution methods (*i.e.*, query splitting and/or database splitting). Suitability of one parallelization method over another is both input data and resource dependent [190, 191, 192, 193]. One method can often be found more appropriate than the other, based on current resource availability. The adopted master-worker model allows flexibility of Dynamic BLAST by supporting notions of advanced scheduling techniques to be automatically applied on the user's behalf. Given model allows for different BLAST algorithms to be invoked on available resources even within a single job. This has the potential of increasing the suitability of available resources and maximizing resource utilization while minimizing job turnaround time.

Internal dataflow for Dynamic BLAST closely follows the architectural components comprising the application and consists of several key layers/steps. A diagram of the

components and dataflow is provided in Figure 44. The main components of the given architecture are:

- Data Analysis (analyzes input files)
- Create Job Plan (decide on resource selection, data distribution, algorithm, and job parameter selection)
- File Parsing and Fragmentation Module (based on job plan, splits the input query file)
- Thread creation (each resource and dataset is assigned a thread)
- Threads (manage all job submission related tasks for assigned resource)
- Post processing (wait on threads to complete, transfer data to local machine and join it into a single results file)

Data Analysis performs statistical analysis of user input query file to extract parameters needed in later steps of data processing. This information includes the number of queries, the average query length, and the standard deviation of query lengths. Described module also extracts current resource availability and information and stores it in an internal, Dynamic BLAST specific format.

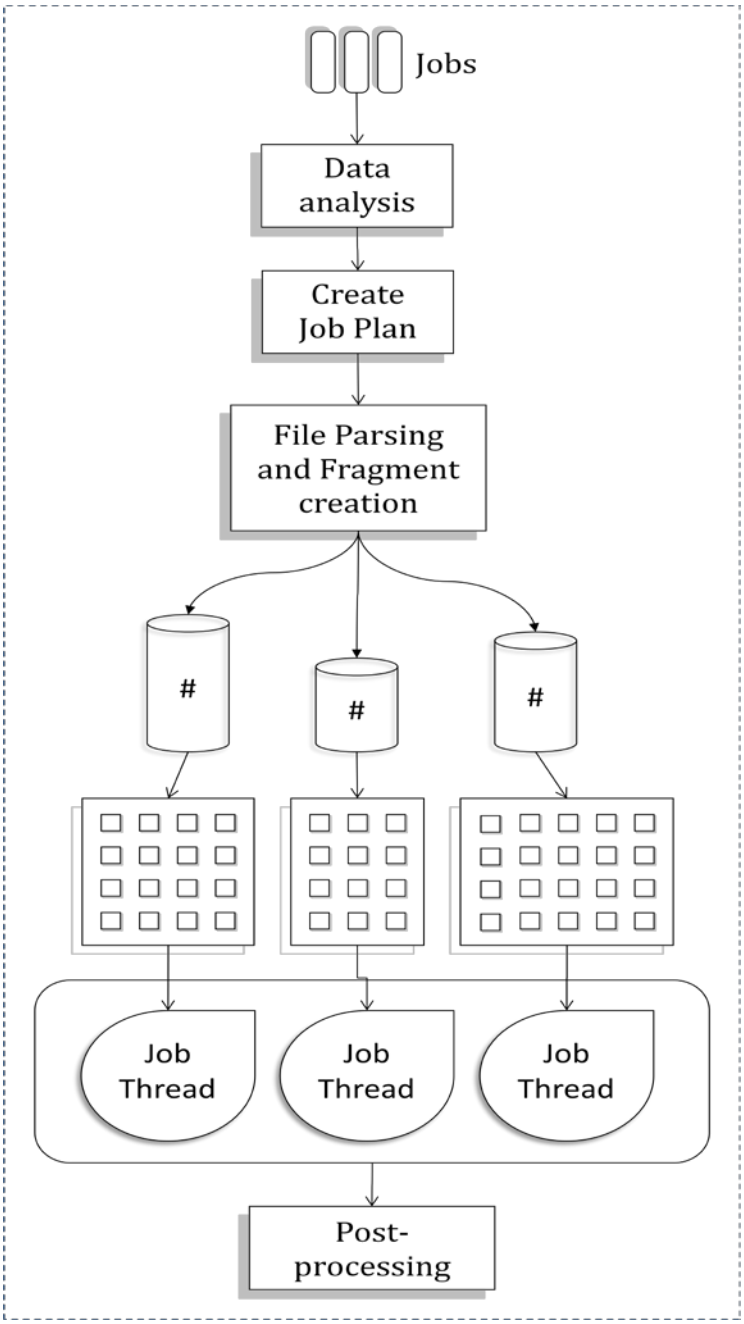


Figure 44. Internal dataflow for Dynamic BLAST

Create Job Plan module implements the general purpose metascheduler presented in Section 4.3 and it is the core of Dynamic BLAST. It uses information retrieved by the Data Analysis module to perform on-line scheduling and job parameterization. Implemented functionality generates query chunks based on relative resource

performance in order to minimize load imbalance. To do so, Equation (7) is implemented for resource selection and Equation (8) for data distribution (as derived in Section 4.3.2). Resource performance weights needed for relative resource performance comparison are obtained from historical BLAST benchmark runs, as available in AppDB. The result of executing the Job Plan module is a concrete job plan for resource assignment and data distribution. This job plan is used to guide execution of the remainder of job submission process.

File Parsing and Fragmentation module reads the job plan and proceeded in two steps. Initially, it splits the original user query file into chunks, one for each resource (chunk sizes are stored in the job plan). Once the chunks are created, each chunk is further subdivided to correspond to task assignments (according to Equation (4)). The number of tasks created is also provided in the job plan and it generally closely corresponds to the number of nodes available on selected resource.

Thread Creation takes place immediately prior to job submission. The master thread creates worker threads – one for each resource. Individual *Threads* read their respective part of the job plan to parameterize a given task (*i.e.*, implement homogeneous resources scheduling model presented in Section 4.3.1). Because of such a granular approach to job plan generation and execution, as stated earlier, different BLAST algorithms and parameters can be used for different resources. This provides needed customization and allows for maximization of resource utilization in addition to high-level of user support and QoS. Jobs are submitted directly by threads to individual resources through DRMAA and GridWay. The master thread waits on the threads to complete. Individual threads initiate an output file transfer back to the initial job submission resource before

completing their execution. If a resource fails or a task does not complete its execution as planned, the master thread can resubmit just the given task to another resource.

Post Processing module is part of the master thread and its primary task is combining all the search result files into a single result file presented to the end user. Any cleanup and additional tasks, such as bookkeeping (*e.g.*, storing new performance data into AppDB) are also performed in this step.

5.1.2. *Dynamic BLAST Performance Results*

Performance of Dynamic BLAST was tested through a set of experiments trying to determine the degree to which Dynamic BLAST was able to realize application-oriented metascheduling across a set of grid resources. Performance results of Dynamic BLAST are compared to a state-of-the-art but general purpose metascheduler.

Based on BLAST performance analysis from Section 4.2, and thus a corresponding ASL document, performance characteristics of a BLAST task are affected by the number and length of the queries to be processed. At the same time, capabilities of individual resources (*i.e.*, the weight factor used in Equation (8)), as derived from application benchmarks, are proportional to the 'standard' input data. In other words, resource performance is based on the same input data set. Because number as well as length of the input queries affect runtime of a BLAST task, in order to maintain relative resource performance derived from available benchmarks, it is important to assign comparable data chunks to individual resources. This implies that the data needs to be assigned to resources in a well-balanced fashion; a proportional number of short, medium, and long queries should be assigned to each individual resource (*i.e.*, cluster) and furthermore to each individual task (*i.e.*, node). The result is that load balance among tasks and resources

will be achieved and thus resource comparison used to derive the job plan and the data distribution will actually hold during job runtime.

This problem can be generalized into a bin-packing problem [185] where the number and size of bins is predetermined (*i.e.,* number of resources and number of queries assigned to each individual resource). A simple yet effective and efficient heuristic implementation for this problem is the first-fit decreasing algorithm (complexity is $\Theta(n \log n)$ where n is the number of queries). The algorithm assigns data elements across individual bins in a decreasing order for as long as there is input [67]. Note that under the constraints of the heterogeneous and distributed environment where this algorithm is applied, a need for an optimal solution is minimal and, instead, focus should be put on efficiency. Therefore, the file parsing module of Dynamic BLAST implements the first-fit decreasing algorithm as a two-step progress: first, user input file is divided into chunks of proportional type of data as dictated by the job plan, and second, each chunk is divided into a number of proportional tasks, as indicated in the job plan again.

Resources and the environmental setup used during the Dynamic BLAST experiments included three resources available on SURAgrid [194]. These resources are located across three independent departments, each locally administered with applicable policies and procedures in place. All of the resources had a version of the BLAST installed and required input (*i.e.,* requested database) data available for use. Technical resource details are provided in Appendix C with resource availability listed in Table 3. Experiments were performed against the 1.6 GB *nr* database and the input file consisted of 4,096 search queries randomly selected from the VBRC database.

Table 3. Availability of resources used during experiments with Dynamic BLAST and AIS.

	Cheaha 2	**Ferrum**	**Olympus**
No. of Nodes Available	12	24	64
Total No. of Cores Available	96	192	128

Initially, the overall runtime results are presented pointing at the performance variability from the user's perspective. Following, details of each run are presented and analyzed pointing at the reasons for performance variability and in turn describing the functionality automatically understood and leveraged by Dynamic BLAST. Performance of Dynamic BLAST is compared to the performance of the plain query splitting variant of BLAST job parallelization (*i.e.*, GridWay-controlled job submission). The query splitting variant operates under the model of dividing the total number of input queries across individual resources based on those resources' relative size. Equation (9), a variation of Equation (8), was used to derive proportional amount of data that should be assigned to each resource. For the given Equation, d_i represents the chunk size for a task assigned to resource i, n_i refers to the number of Processing Slots (PSs) or nodes on resource i, D represents the total size of user input.

$$d_i = \frac{n_i}{\sum_{j=0}^{j<R} n_j} \times D \tag{9}$$

Each data chunk d_i is divided into tasks according to Equation (4). Additionally, because of the described impact of data distribution and data format (*i.e.*, query length) on performance of BLAST jobs, performance of Dynamic BLAST is compared to applying only data distribution to the query splitting variant without including resource-specific weight into the Equation. The overall runtime characteristics of this set of

BLAST jobs are provided in Figure 45. The figure presents contributing runtimes of individual resources. As can be seen in the figure, obtained runtime statistics indicate that utilizing application-specific data distribution model reduces runtime of the overall job by approximately 50%. The job runtime is reduced an additional 15% by incorporating the Dynamic BLAST calculated resource weight into the data distribution model. Realized improvements are a result of the reduction of load imbalance across individual resources. This leads to greater resource utilization and thus shorter job turnaround time.

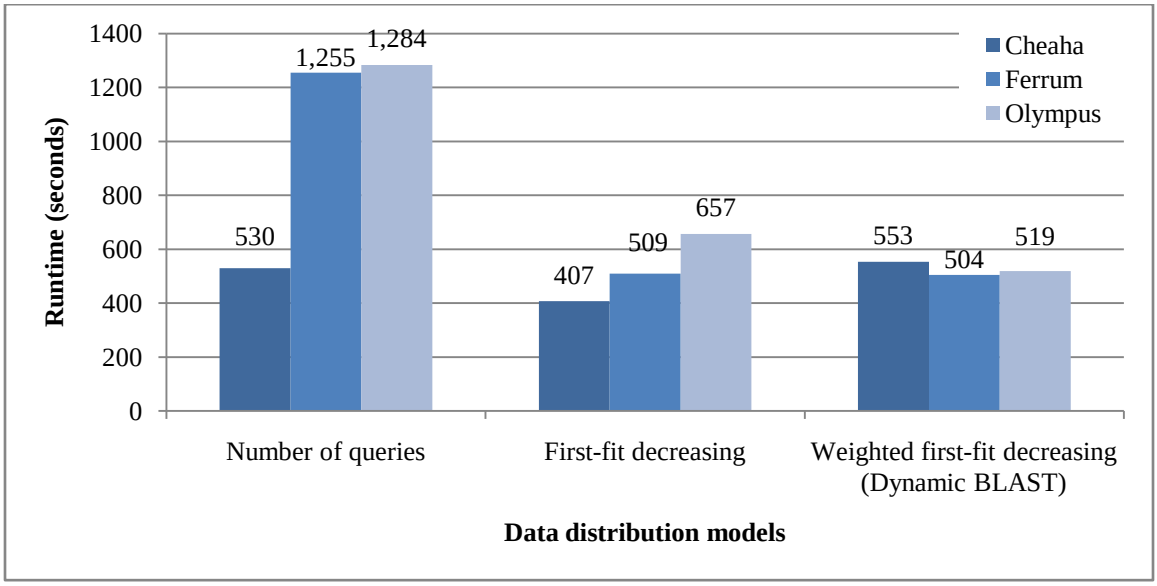


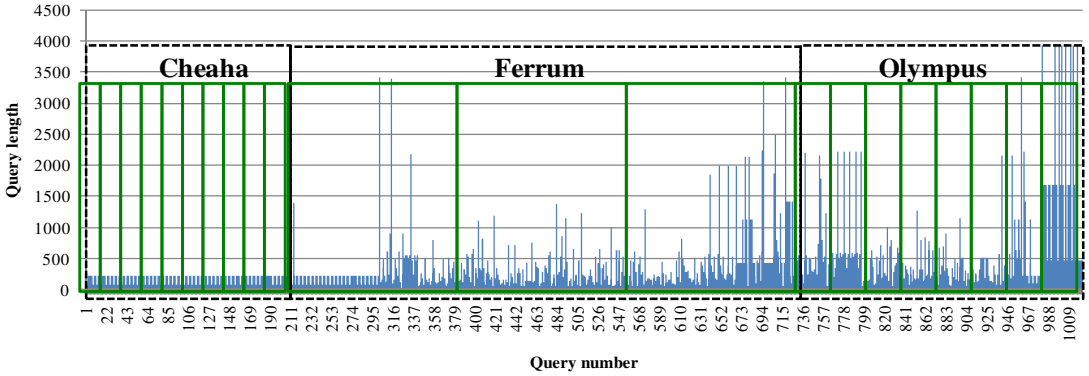
Figure 45. Runtime characteristics of a set of BLAST jobs ranging from plain query splitting BLAST parallelization to Dynamic BLAST. Different data distributions indicate the restructuring of the data assigned to individual resources and assignment of data amount to individual resources that is proportional to resources' capability. Experiments were performed against the 1.6 GB nr database using 4,096 query input file.

As discussed in Section 4.2.1 and in [17], runtime of BLAST algorithm is significantly affected by the length of the input query. By simply taking an input file provided by the user and dividing it into a number of chunks at predetermined data points (*e.g.*, using the UNIX *split* utility), the type of data that gets assigned to individual tasks (*i.e.*, nodes) within a resource can vary greatly. This can result in the load imbalance

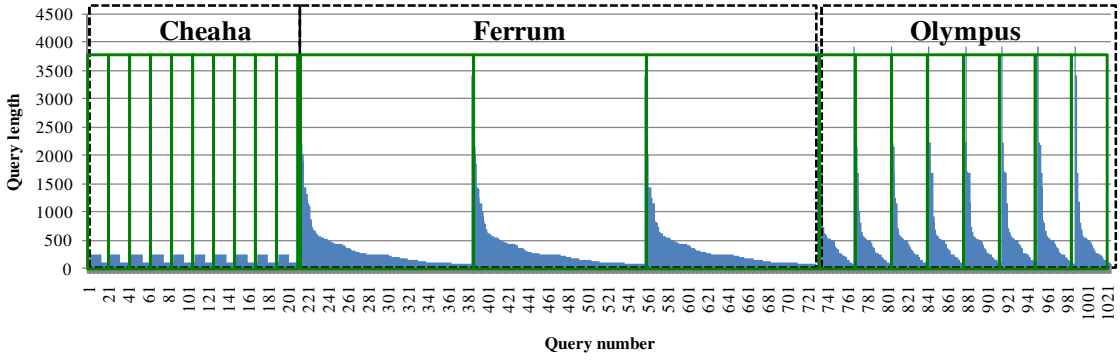
problem. Figure 46 shows a profile view for the portion of query lengths that were used during experimentation with Dynamic BLAST (only a portion of the dataset is shown to enable meaningful display of the data and this does not result in any loss of generality). As is evident from the (a) portion of the figure, lengths of individual queries vary greatly and are unevenly spread across the provided input file. When Equation (9) is applied to this input data set, corresponding to the three available resources, three well defined data chunks are created (dotted black boxes shown in the figure). Each of those data chunks are relative to the resource size. In addition, each of the data chunks is further divided among available nodes on a particular resource (solid green boxes). Figure 46 (a) then presents a great level of variability and irregularity among properties of various queries, as they are assigned to corresponding resources and then nodes. Instead of simply dividing the data at predefined points, by applying the first-fit decreasing algorithm to reorganize assignment of individual queries to corresponding nodes, data distribution shown in Figure 46 (b) can be obtained. Resulting data distribution shows a much more even distribution of comparative queries across individual compute nodes. The result is reduction of load imbalance across those nodes and reduction in runtime of the overall job, as already shown in Figure 45.

Having observed the benefits of exploiting application-specific knowledge regarding data distribution to reduce job runtime, Dynamic BLAST also implements the realization that individual resources should not be compared solely on their size but instead on their application-specific performance [14]. By applying Equation (8) when performing data distribution, the size of data chunks assigned to individual resources corresponds to the particular resource's capability in terms of BLAST application (as opposed to the generic

resource size). Figure 47 presents the two different data distributions; one is based on the resource size only while the other one is based on resource performance. Figure 45, under "Dynamic BLAST," shows the effects resource-performance based input data distribution has on overall job runtime.



(a)



(b)

Figure 46. Difference in query distribution between (a) simple query splitting model and (b) BLAST-specific data distribution model. Dotted boxes indicate the amount of data assigned to individual resources while solid boxes indicate data portions assigned to individual nodes (i.e., tasks) on any one resource. Consistent distribution of queries results in consistent node and resource performance reducing load imbalance.

In conclusion, it can be stated that Dynamic BLAST represents an implementation of the query splitting BLAST parallelization model that far supersedes performance of the plain query splitting model. This achievement is possible because of the specific ties that

have been made to understand and leverage BLAST execution characteristics across heterogeneous resources and implement the general purpose metascheduling model.

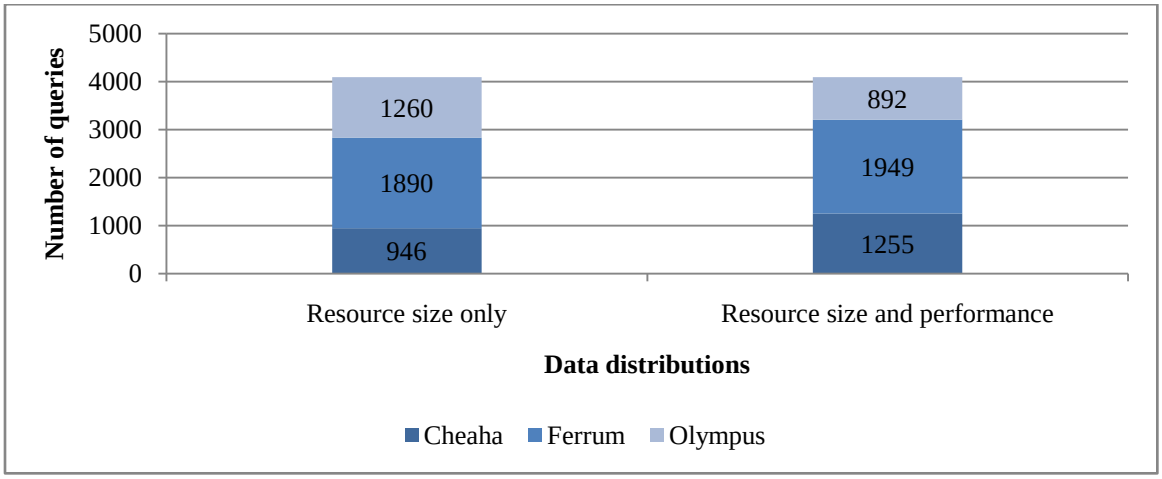


Figure 47. Different data distributions across resources based on (1) resource size only, and (2) resource performance.

5.2. Statistical Genetics Domain

Dynamic BLAST application represents an application-oriented metascheduler built on top of the derived general metascheduling model (from Section 4.3) and operates with a general objective of runtime minimization for the specific application. Mirroring the Dynamic BLAST analysis, we show how the same approach derived there, and then generalized to the EP class of applications, is just as effective when applied to statistical genetics R code.

The performance data shown (see Figure 48) represents the full range of job performance results, starting with the initial task distribution and gradually applying application-specific metascheduling information and performing corresponding adjustments to realize desired performance improvement. All the experiments focused on executing a single job parameter for 10,000 iterations; the 10,000 iterations were distributed across four resources, thus introducing parallelism into the application

execution. Details about resources used are available in Appendix C with their availability listed in Table 4.

Table 4. Technical details of resources used during experimentation with R code.

	Ferrum	**Olympus**	**Cheaha 1**	**Everest**
Number of Nodes Available	5	25	5	10
Total Number of Cores Available	40	50	10	20

Figure 48 presents the performance characteristics of progressive improvements regarding the set of executed jobs. The progression has been done through three different data distributions, resulting in significant performance improvement. In the figure, the "Basic" distribution is based on relative resource size, where Equation (9) was used to assign a number of iterations to each resource; furthermore, Equation (4) was then used to calculate a number of iterations that should be executed by each node within a resource.

In the first improved step, with the goal of obtaining an application-specific value for resource performance as opposed to resource size alone, we used Equation (8) to incorporate application-specific resource performance weight. Resource weights were obtained from the analysis of the initial run. With existence of AIS, such information is stored and retrieved on a as needed basis. In this step, we exploited information available within GridAtlas to obtain the number of Processing Elements (*i.e.,* cores) within each Processing Slot (*i.e.,* node). This permitted us to parameterize each task more appropriately to the true resource capabilities. Results of this improvement are shown in Figure 48 under "Resource-weighted distribution."

Finally, noticing that there is considerable load imbalance across employed resources, we resorted to further analysis of application execution characteristics and development of an application-specific module to accompany the general metascheduler (more details are included in following paragraphs). The developed module implemented heterogeneous data distribution to individual nodes within available resources to cope with application peculiarity regarding input data processing. The final runtime results are shown under 'Resource-optimized distribution' column in Figure 48, where it can be seen that load imbalance was further reduced, resulting in 75% reduction of runtime when compared to the initial case.

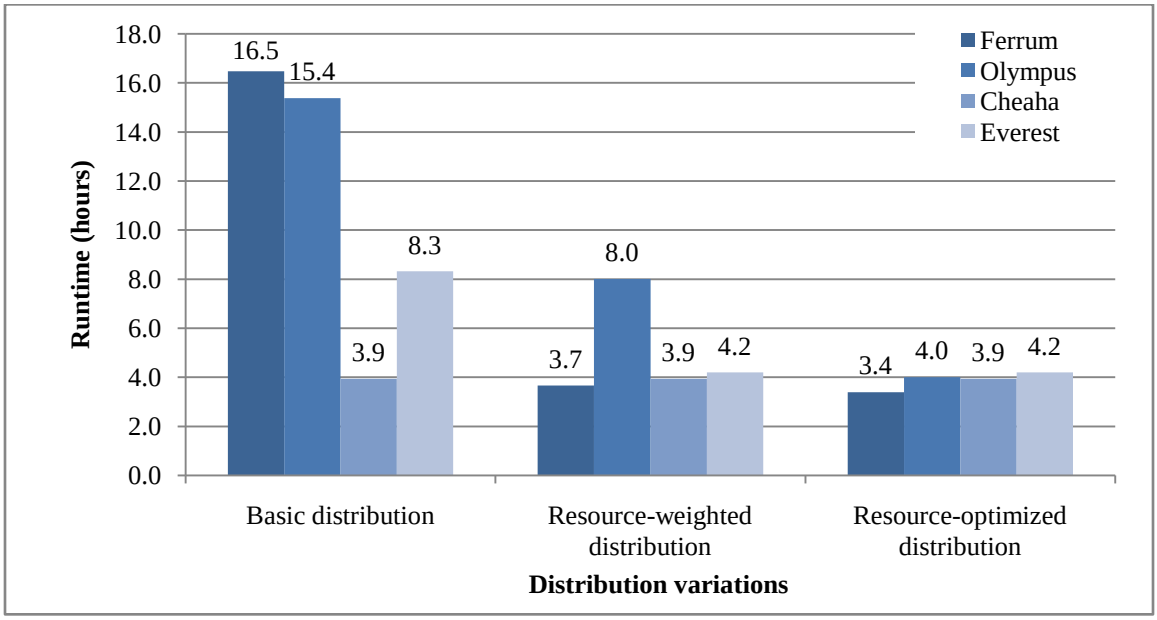


Figure 48. Runtime characteristics of a set of R jobs highlighting importance and effects of applying derived data distributions and utilizing publicized resource allocation policies. More specifically, jobs executed a single parameter set for 10,000 iterations and multiple processes were started on each node to correspond to the total number of processing cores available per resource node.

Based on the conclusion of BLAST analysis, performing data distribution by incorporating resource capability instead of resource size alone was the first level optimization when metascheduling R code. The two data distributions corresponding to

the number of iterations assigned to individual resources are shown in Figure 49. Unlike with BLAST, which required an application-specific module to perform data division, R code, being a parameter sweep type of an application, is simply parameterized with different input values when a job is invoked. This case imposed fewer requirements in terms of application specific adaptation and/or development of the metascheduling adapter component.

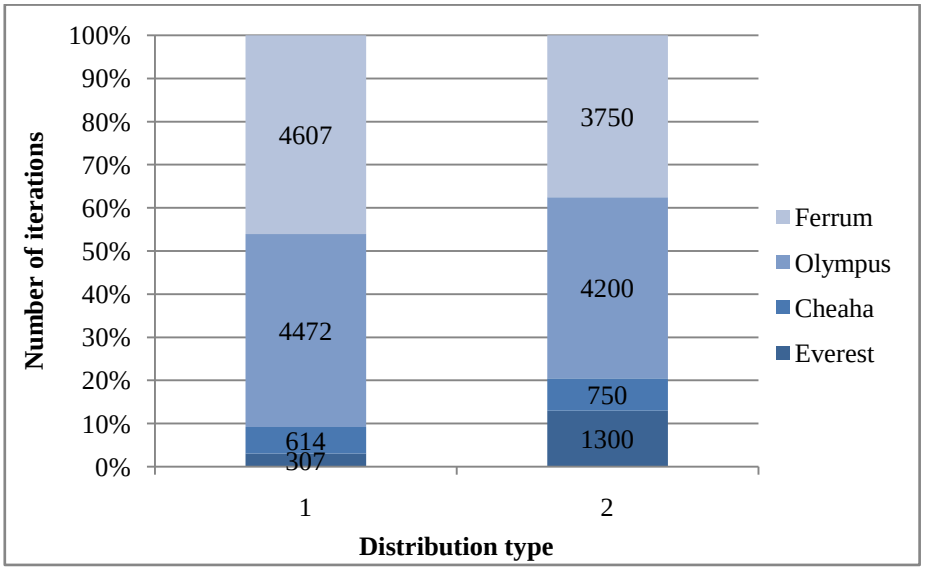
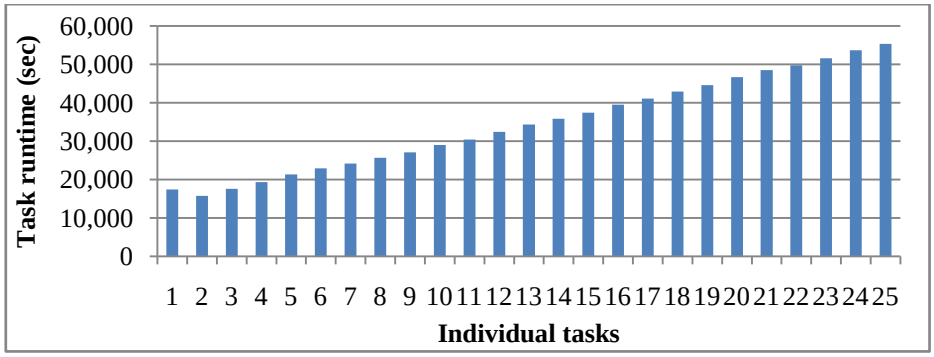


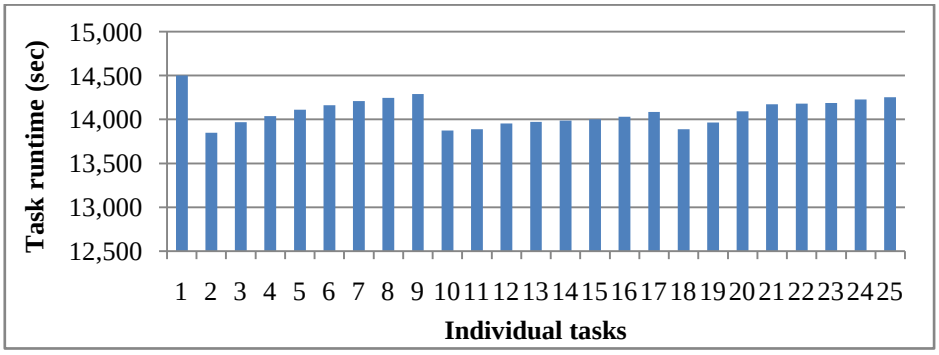
Figure 49. Two data distributions across resources based on (1) resource size only, and (2) application-specific resource performance. Values indicate number of iterations to be performed against the parameter set under analysis.

In Figure 48, under "Resource-weighted distribution," significant job load imbalance was observed by one resource (namely, Olympus). By analyzing performance of jobs that were submitted to that resource, an irregularity was observed across individual tasks (see Figure 50a). The irregularity was resulting in tasks executing progressively longer as the iteration index numbers grew. Similar observations were made across all resources, but it was most prominent on Olympus because of the largest number of nodes employed on given resource. In order to cope with this realization, an R-specific metascheduling plug-

in was developed that assigned and controlled assignment of a variable number of iterations to each node within a resource. This action can be seen as being parallel to the initial chunking of original 10,000 iterations across resources. In other words, variable iteration numbers were assigned to individual nodes within a resource. Figure 50b depicts the resulting irregularity and runtimes of tasks. Although noticeable load imbalance is still present, overall tendency is largely disrupted resulting in much less impact on the overall runtime of the job (as can be seen in final runtime results from Figure 48).



(a)



(b)

Figure 50. (a) Initial irregularity among runtimes of individual tasks within one resource (i.e., Olympus), and (b) restricted irregularity leading to more controlled load balancing. Control was achieved through heterogeneous assignment of iterations to individual nodes.

Final results of metascheduling R in an application-oriented fashion resulted in reduction of runtime on the order of 75%. Such reduction came as a result of following principles of a metascheduling methodology that were devised throughout this dissertation. Therefore, the presented methodology can be seen as a set of guidelines and directions that can help alleviate many of the issues found along the process of metascheduling grid applications. By following this methodology, it has been shown that significant improvements can be made that do not require exorbitant amount of effort or time.

5.3. Realizing User-oriented Metascheduling

Efforts discussed thus far have focused on a single objective optimization, namely minimization of job runtime. As the last major contribution of work presented in this dissertation, we have adopted the notions behind user-oriented metascheduling. Enabling user-oriented metascheduling requires extensive use of application-specific information and application-oriented metascheduling. Realization of the devised approach is observed through generation of a set of *job execution options* that are mapped onto conflicting objectives. A *single job execution option* represents a decomposition of a job into a set of tasks and a mapping of this set of tasks to a set of execution resources. Each option represents a distribution, or allocation, of job input requirements (*e.g.*, input data, number of iterations), representing the job's workload, to selected resources in a manner that best meets selected resources' capabilities and minimizes load imbalance across employed tasks. Repeated generation of such job execution options leads to a *job execution space* that covers a full spectrum of job execution alternatives and whose effective presentation

offers deep insight into available job execution tradeoffs to an individual user (as discussed in Section 3.6.1).

Realizing generation of the job execution space is achieved through a refinement of the presented metascheduling framework. Overall, the framework is complemented by a Controller module that artificially constrains resource availability passed to the application-specific metascheduler. Repeated invocations of the metascheduler result in generation of the job execution space (details are provided in the following sub-sections). These sub-sections cover the following topics: the job execution option selection process (Section 5.3.1 and Section 5.3.2), the scheduling algorithm derived for generating selected options (Section 5.3.3), and the subsequent mapping of derived options to absolute values (Section 5.3.4). Experimental validation was performed in a simulated environment and on real-world resources and is presented in Section 5.3.5 and Section 5.3.6.

5.3.1. *OptionView Architecture*

Tool presented in this section is referred to as OptionView. OptionView represents an implementation of the user-oriented metascheduling. The notion of job execution tradeoffs is implemented allowing a user to meet their utility for the current job and current situation without any regard for how to use the underlying infrastructure.

Figure 51 presents the global architecture of OptionView. Job submission is initiated by a user who simply selects which application to run and provides desired input data file. OptionView then analyzes job properties and resource availability leading to a job execution option space that is presented back to the user. The user considers presented options and selects which job option to execute. Additional motivation and explanation of

the interaction between a user and the metascheduler are provided in Section 3.6.1. The control is then transferred to a job submission engine for execution on appropriate grid resources. *Controller, scheduler, mapper, and GUI generator* are the four major components comprising OptionView (see Figure 51).

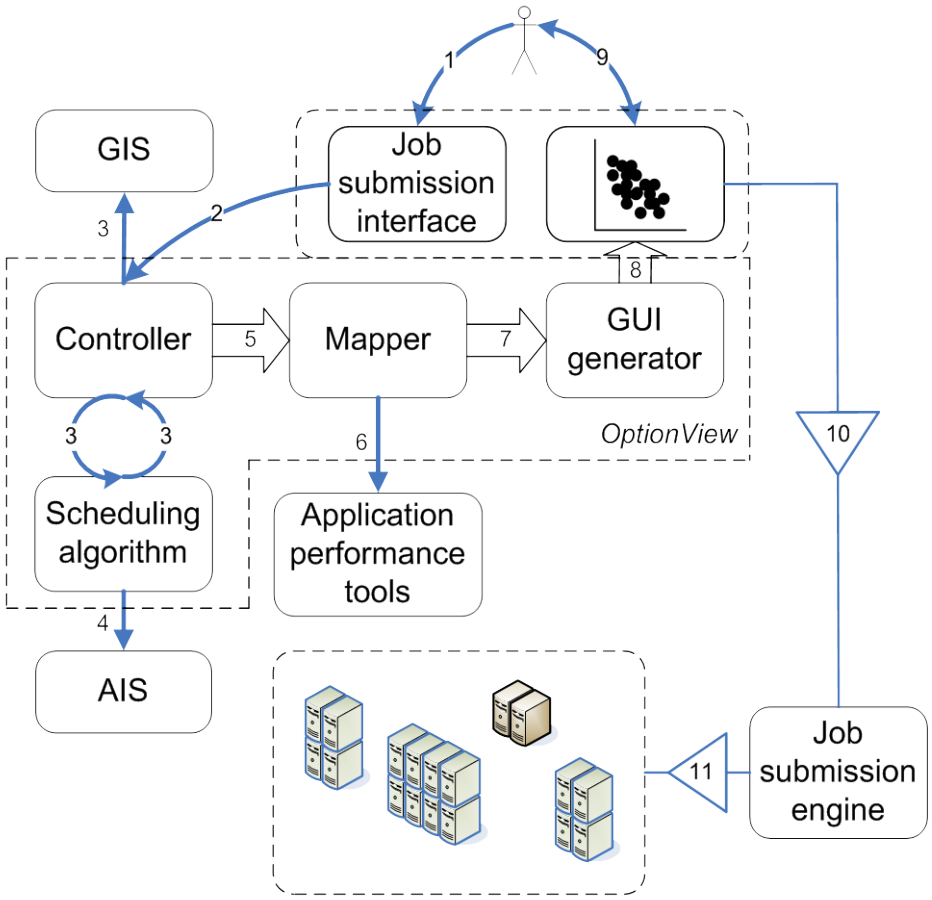


Figure 51. High-level OptionView architecture with numbers indicating general progress flow. Following user initiated job submission, the scheduling algorithm is repeatedly invoked to generate the job execution space. After presenting the job execution space to the user, the user selects desired job execution option.

The *controller* accepts a job submission request from the job submission interface and implements the logic controlling option generation. It acquires information from the GIS regarding resource availability. The *scheduler* is invoked for each option the *controller* selects to generate. The *scheduler* acquires application-specific information

required for effective scheduling from AIS [173] and generates tasks comprising the given option. After all of the selected options have been generated, the *mapper* is called. It maps options' relative values to the absolute ones based on information from available tools (*e.g.,* performance prediction tools [195], AIS, AppDB [171]). Lastly, the *GUI generator* presents the derived options to the user by mapping them onto conflicting objectives.

5.3.2. *The Controller*

The controller is responsible for coordinating generation of job execution options. As stated in the previous section, a job execution option represents a single parameterization of the job. In other words, a single job execution option corresponds to a single execution of function $plan_J$ defined in Section 4.1.2. Descriptively, a single job option is comprised of a set of resources and a CPU assignment across those resources as they are chosen among all of the available resources and resources' maximum CPU capacity. Furthermore, within each option, each resource is assigned a task that meets resource's capability. All tasks across selected resources have their workload distributed in such a fashion that the overall load imbalance within the option is minimized. Note that a task, depending on the application, may correspond to a single or multiple execution instances of the application within a resource (*i.e.,* one process consuming multiple CPUs or multiple processes consuming one CPU each).

An example is provided in Figure 52, where total resource availability is shown along with two sample job execution options. Resources 1 and 2 comprise Option 1 with 5 CPUs selected from the 10 available on Resource 1 and, similarly, 10 CPUs from 15 available on Resource 2. How these assignments are selected is discussed in the

following paragraphs. The data assignment in the example represents the amount of data units assigned to each resource such that chosen resource capability is most closely matched to other resources' capabilities within the option. Note that even though twice the number of CPUs was selected for Task 2 when compared to Task 1, Task 2 was not assigned twice the amount of data units. Proper assignment of data among selected resources within an option is based on application-oriented metascheduling approach where performance of given resources is measured in terms of the application at hand. This process is described through the scheduling algorithm described in the next section.

Resource	*Free CPUs (total)*
R1	10
R2	15
R3	8

Option 1			
Task ID	*Resource*	*CPU assign.*	*Data assign.*
1	R1	5	100
2	R2	10	150

Option 2			
Task ID	*Resource*	*CPU assign.*	*Data assign.*
1	R1	5	80
2	R2	10	110
3	R3	5	60

Figure 52. A sample two job execution options. Based on maximum resource availability, different configurations of resource availability are artificially constrained by the Controller and used to invoke the Scheduler, which automatically generates job execution option. Repeated invocations of the Scheduler lead to generation of job execution space.

In order to initiate generation of job execution options, the controller's two main functions are: (1) determining the range of possible parameters for job options, followed

by (2) selection of which options to generate. The acceptable range of parameters for options is governed by the resource availability. Because individual options are primarily distinguished by the different selection of resources and assignment of CPUs across those resources, the total number of possible options is the product of idle CPUs across available resources. For the example from Figure 52, the total number of options would be (10+1) × (15+1) × (8+1) = 1584 (note that 1 is added to the total number of CPUs on each resource to account for zero CPUs on each resource – that is, not using the resource). From this simple example, it is obvious that the total number of possible options for all but non-trivial resource availability will be very large. Because the total number of possible job options represents an exhaustive list of resource and CPU assignments, it is clear that such an approach is unnecessary and even unwanted. Rather, a subset of all possible options should be selected, processed, and presented to the user. Such a subset should be representative of the overall set of options to allow for a comprehensive overview of the possible job execution options to the user.

Selection of the subset of options can be performed using either of two main statistical sampling principles: random or targeted [196]. A form of targeted search includes selection of options based on their quality with the solution yielding a Pareto line [67] of job execution options. Because the details of the sample space (*i.e.,* job options) are unknown prior to the generation of the data points (*i.e.,* job options), the quality of individual data points cannot be calculated. Such an approach would thus require generation of all the options (by invoking the scheduling algorithm) followed by option ranking and selection. Because of the associated computational cost, this is infeasible. An alternate targeted search is systematic sampling [196], where resource- and

application-specific knowledge is exploited to drive option exploration. In such a case, certain predetermined resource configurations could be explored, such as: use all idle CPUs on a resource, use half of idle CPUs on a resource, use a quarter of the idle CPUs on a resource, etc. In addition, application-specific information about scalability of the application (*e.g.*, number of CPUs must be a power of two, or maximum number of CPUs this application scales to is n) can be used to select option configurations. The benefit of such an approach is that the option configurations can be systematically chosen, requiring generation of only a small subset of all possible options. Application-specific knowledge enables such targeted option selection to maximize job's performance. If needed application-specific information is not available though, systematic sampling resorts to resource-based sample selection and can thus omit interesting data points.

In cases where application-specific information is lacking, a simple random sampling method [196] can be used to perform option selection. Based on [123], in addition to our own observations, the exhaustive list of job execution options is characterized by the normal distribution. Well-established statistical methods (*e.g.,* [196]) can thus be used to obtain a high-level of confidence about truthful representation of a sample compared to the overall sample space. To obtain desired level of accuracy, a confidence level of 95% is used with a 5% confidence interval. These values can easily be adjusted by the user, but the presented values offer a suitable balance between required and presented number of sample data points. Based on [196], a suitable formula to use for calculating the number of data points is as follows:

$$n = \frac{(z^2 \times p \times (1-p)) + e^2}{e^2 + \frac{z^2 \times p \times (1-p)}{\mathrm{N}}} \tag{10}$$

where z is the critical standard score calculated as $1 - \frac{\alpha}{s}$ and final z value is obtained from the normal table. α refers to the likelihood that the true population parameter lays outside the confidence interval and is calculated as $1 - confidence\ interval.$ p is the population proportion; because no prior knowledge about job execution options exist, p should be set to 0.5 to maximize the size of the calculated sample. Variable e specifies the margin of error or the confidence interval while N is the overall population size.

The calculated number of job execution options is randomly selected by the *Controller* module without replacement from the exhaustive list of job execution alternatives. Using the described method, the number of options selected was less than 400 irrespective of resource availability and the total number of possible job execution options. This represents a manageable number of data points to be calculated by the metascheduler and presented to the user.

5.3.3. *Metascheduling Algorithm*

The focus of the presented scheduling algorithm is to provide a job plan for an individual job execution option under the constraints of the $plan_J$ function described in Section 4.1.2. Implementing the described function results in generation of a set of tasks that are mapped to selected resources, each directly corresponding to capabilities of an assigned resource. By definition of the $plan_J$ function, minimizing load imbalance among tasks to be created maximizes option performance, and is thus the single most important goal of the metascheduling algorithm. In the context of presented algorithm and selected application type, this is realized through relative comparison of available resources and assignment of appropriate data allocations to each resource. This results in cumulative minimization of load imbalance at the job option level.

Overall, when a job execution option data is presented to the metascheduling algorithm, the input data is decomposed by applying a static scheduling scheme and matched to selected resources. As a result, a fraction d_i of entire data is allocated to task t_i. The data decomposition is based on the performance information of current application on selected resources. The objective of the scheduling action is determining the workload distribution $\{d_1, d_2, \dots, d_n\}$ corresponding to a set of tasks $\{t_1, t_2, \dots, t_n\}$.

The pseudo code for the scheduling algorithm is provided in Algorithm I and it proceeds as follows: after receiving initialization data that consists of resource configurations, input data size and application name, the performance of resources in terms of current application is computed (lines 2 and 3). This strategy accounts for heterogeneity of individual resources, so that the amount of data d_i assigned to resource R_i is proportional to the resource capacity. Formula (11) is used to perform the calculation:

$$optionPerf_i = n_i \times W_i \tag{11}$$

where n_i is the number of processing elements on resource R_i and W_i is application performance metric, or weight, for the same resource. Calculation of such application performance across selected resources is implemented in a separate module, *adjustResourcePerfToApp* on line 3. This module interacts with AIS to obtain needed application- and resource-specific information (*e.g.,* application benchmark data for select resource). On line 4, the data allocations $dData = \{d_1, d_2, \dots, d_n\}$ for corresponding resources are calculated. d_i is computed as follows (D represents the total size of user provided input data):

$$d_i = optionPerf_i \times \frac{D}{\sum_{i=1}^{n} n_i} \quad (12)$$

Depending on the application though, workload allocation process may differ from the one just shown and can thus be implemented as a plug-in to OptionView's scheduling algorithm by implementing *divideData* module. By extracting implementations of resource weight calculation and data distribution into separate modules, the presented algorithm is made more versatile while providing support for application-specific scheduling [88]. With all of the information calculated and available, the second *for* loop iterates over the number of available resources, assembles information into a single task, and adds the task to the current option list. Generated option is then returned to the Controller.

Algorithm I – Job option generation algorithm

```
1:Receive resource info R, job input size D, application info A
2:for i = 1 to |R| do { |R| - number of resources }
3:   optionPerf[i]  = adjustResourcePerfToApp (R_i, A);
4:dData = divideData (R, D, optionPerf);
5:for i = 0 to |R| do
6:    task = new Task (i, anOption, dData, resourceID);
7:    option.add (task);
8:   return option;
```

5.3.4. *User Interaction Module*

Job execution options calculated by OptionView are presented to the user for analysis through a user interaction module. This user interaction module consists of the mapper and the GUI generator.

The *mapper* is concerned with taking the relative values of job execution options generated by the scheduling algorithm and replacing those with the absolute ones. During execution, the scheduling algorithm normalizes performance of individual resources and thus the options it returns are relative to each other on the normalized scale. In order to

map those onto absolute values, whether it is cost and time or accuracy and time, the mapper must analyze each option's parameters and adjust the values. The mapping component interacts with the necessary services to obtain actual resource cost and the application's absolute performance values (*i.e.*, base execution time). This is achieved through calls to $estTime(t,r)$ and $estCost(t,r)$ functions defined in Section 4.1.2.

Note should be taken that derived options have the property of *relative comparison*, meaning that one option is directly comparable to any other option. This property follows from the consistent method through which the options are generated (*i.e.*, a single scheduling algorithm). The benefit is that runtime and cost estimation for all the options can be derived by scaling such information from any single option. This is important because the presented method operates on an application-specific basis and, as a result, options that it generates are unique and application-specific leading to hardship in accurate runtime/cost estimation by generic methods [195]. Therefore, mapping of options to desired objectives is performed by obtaining relevant information for a single option (*e.g.*, from historical data or performance prediction tool) and then scaling the remainder of options accordingly.

The *GUI generator* represents the user-scheduler interaction module. All the complexities of resource, application, data, and scheduling calculations and interactions are abstracted and hidden from the user who is presented with a clean interface enumerating available options and visualizing them in terms of selected tradeoffs. Such an approach to user-scheduler interaction enables a two-way interaction model where the user is offered insight into their job's execution properties before committing to job

submission. Through this model, the user becomes aware of job's execution space and can choose job execution option that they might not have known even existed.

A sample user interface displaying a set of possible job execution options is provided in Figure 53. As can be seen, the interface provides a mapping of available and generated job execution options onto the two objectives, namely time and cost. The user can easily interpret available options, consider tradeoffs and select an option for execution. Such presentation of a comprehensive spectrum of available job execution options allows the user great flexibility in terms of meeting their current needs (compare requirements and experience of an OptionView user to the user accessing the grid as described in Table 1).

The interface allows the user to see details about any one option and apply a filter that will reduce the number of elements displayed on the screen. If user has narrowed down the desired area, local exploration functionality has also been implemented. Based on user input, the controller is invoked with a specific job execution option around which the user wishes to explore. The controller initiates exhaustive computation of a set of job execution options within a predefined window that are neighboring the one specified by the user (neighboring as based on resource and CPU assignment). The aim of this functionality is to provide the user with deeper insight regarding job execution options within a targeted range.

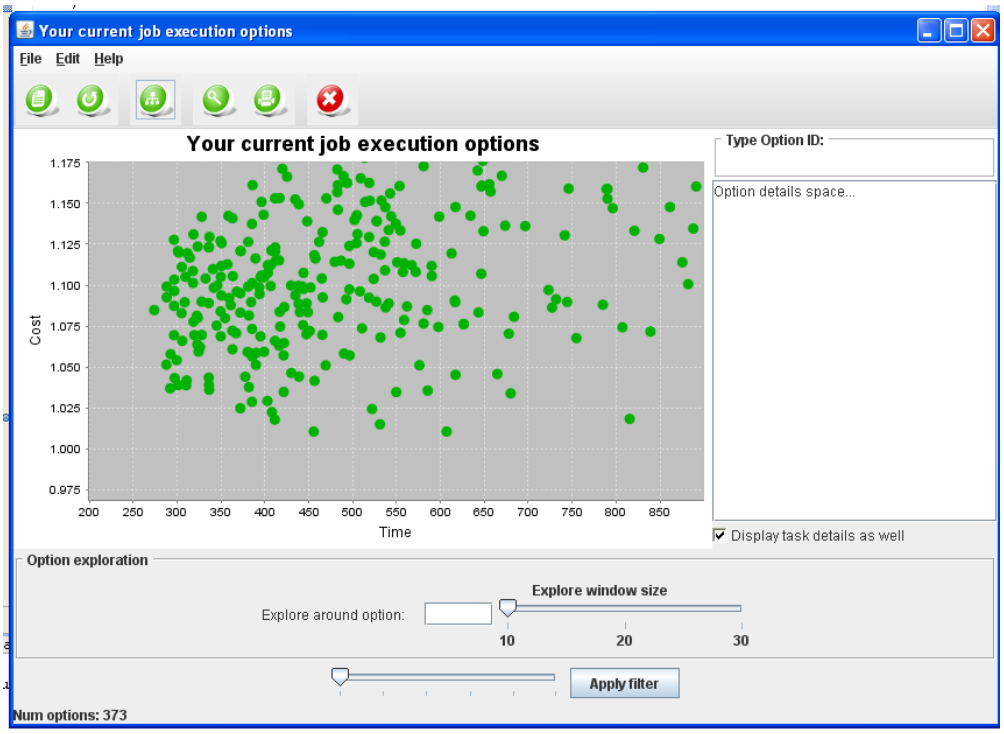


Figure 53. A snapshot of the OptionView GUI module presenting a job execution space to the user

Because OptionView's focus is on individual users and their individual jobs, it represents user's workspace for access to the grid. The workspace targets single user use and the implementation can reside either on user's local machine or in a personal account within a portal. Also, note that with the adopted approach there is no need for job queues. Within OptionView, jobs are mapped only to available resources. In environments where resources are rarely idle or where resource availability changes rapidly, presented approach can be integrated with solutions for advance reservation (*e.g.*, [197]).

5.3.5. *Experimental Validation of OptionView through Simulation*

Validation of OptionView has been done in two stages. In the first stage, the entire set of job execution options is validated for accuracy and relative comparison through

simulation. In the second stage, a representative subset of the options is executed on real-world resources further demonstrating the validity of job option generation mechanism.

With the focus on generating job execution space that is applicable to an individual application and an available set of resources, the validation methodology employed focuses on one application, namely BLAST. During performed tests, *nr* database, 1.6 GB in size, was used to execute the searches against the 1,024 input protein queries.

The simulation part of validating proposed scheduling approach is performed through the GridSim toolkit [162]. As described in Section 2.9.2, GridSim is a grid simulation package that allows for creation and customization of individual resources as well as creation of heterogeneous jobs. Jobs created were packaged as Gridlets, which are specified in terms of job length in millions of instructions per second (MI), the size of job input, and the size of job output in bytes. Processing times of jobs within GridSim are proportional to the predefined speed of resources (shown in Table 5) and the size of the job with a random variation of 0-10% to account for heterogeneity present in real-world grid environments.

For the simulation experiments, a set of resources was created within GridSim that represent available real-world resources in terms of their configuration and relative performance. Relative performance was assigned based on application-specific resource benchmarks; BLAST was executed with the same input data across all available resources and obtained runtime values were normalized to the fastest resource. Derived values were used for the performance rate (*i.e.,* Millions of Instructions per Second - MIPS) of individual Processing Elements of a resource within GridSim (see Table 5).

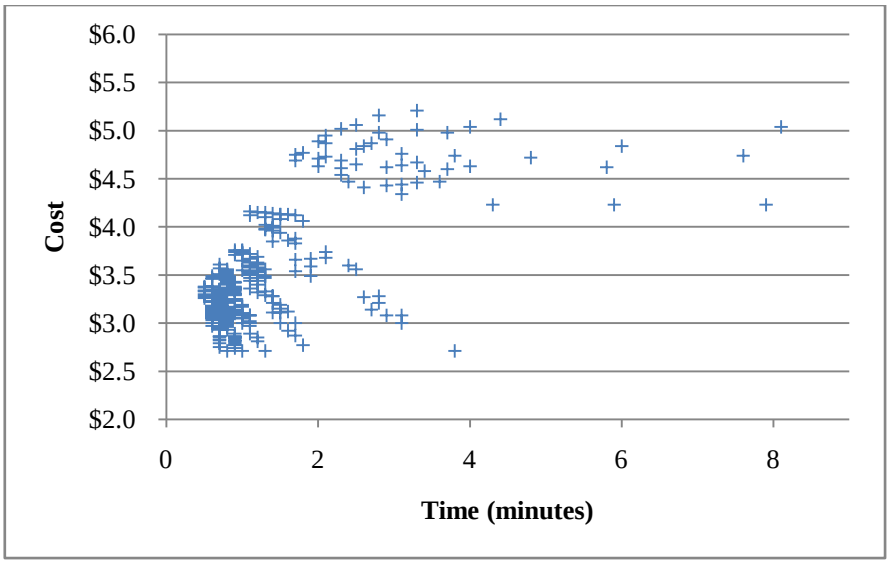
Cost associated with consuming resources was uniformly assigned to $0.10 per unit of execution (*i.e.*, the same as Amazon.com's Elastic Cloud[10]).

Table 5. Resource details used during experiments. PS refers to Processing Slot or a node. PE refers to a Processing Element or a core. MIPS stands for Millions Instructions per Second of a single PE and is a resource performance metric employed by GridSim toolkit. PE MIPS were derived based from normalized application-specific performance benchmarks for given resource.

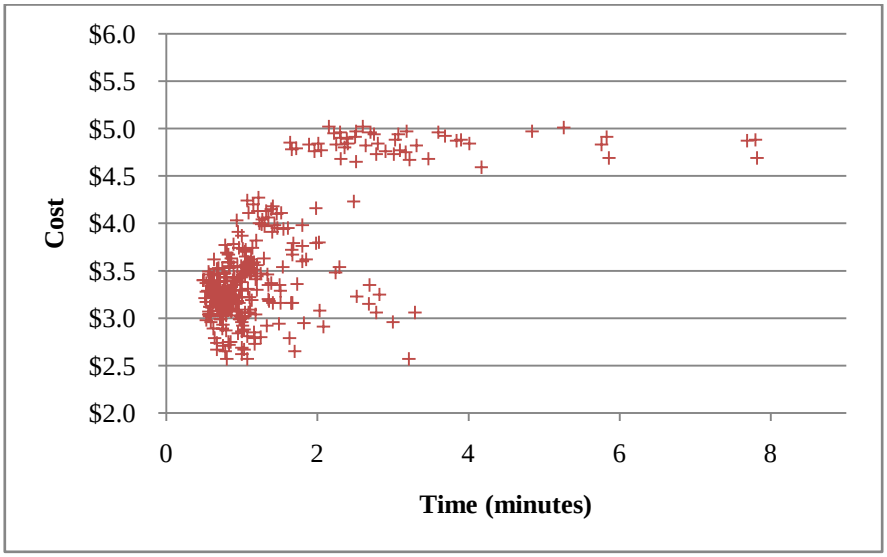
	# PSs	# PEs	PE MIPS rating	Cost/time unit
Resource 1 (F)	5	40	100	$0.10
Resource 2 (E)	10	20	52	$0.10
Resource 3 (C)	15	15	57	$0.10

OptionView was used to generate a set of job execution options containing resource assignments and appropriate data distributions. Derived options were simulated through GridSim by setting task's sizes as derived by OptionView and submitting the jobs to respective resources. Results are shown in Figure 54(a) and Figure 54(b) for generated and simulated data, respectively. From the shown data, it may be observed that the generated job execution options are somewhat more structured and regular than the simulated counterpart. Nevertheless, the overall shape of the job execution space is maintained across generated and simulated data indicating global accuracy achieved by OptionView. Groupings of shown job execution options are a result of task assignments to any one resource with different PS assignments.

[10] http://aws.amazon.com/ec2/



(a)



(b)

Figure 54. Job option execution space as (a) generated by OptionView, and (b) simulated through GridSim. Each individual point shown represents a single job execution option, namely all the details required to submit a job in an application-oriented fashion (e.g., resource(s) selected for execution, data distribution under resource capability constraints, and individual task parameterizations).

Statistical results of the analysis of simulated data at the individual option level are presented in Table 6. In the table, delta (Δ) is calculated by noting the difference between

each of the generated and simulated data points and summarizing those across the sample space. From this data, it can be observed that on average the system achieves a high level of accuracy (approx. within 4%). In addition, value for the median statistic indicates that the system is evenly generating data points on the positive and negative side. Noteworthy values for the standard deviation and maximum delta indicate that many data points are likely to be incorrectly generated. Based on values of standard deviation measurements, it is apparent that the error is largely contained within 15% of the observed results. In the area of application runtime prediction, this is considered an acceptable result (*e.g.,* [198]). The cost analysis presented shows that the cost component of the estimate is at least as accurate as the runtime component. Overall, these results present high accuracy of the system and validate the technique and the approach adopted.

Table 6. Statistics of differences between generated and simulated job execution options. Numbers indicate difference in respective units and the corresponding percentage of simulated results when compared to the estimated values.

	Runtime analysis		**Cost analysis**	
Avg Δ	0.01	4.08%	$0.04	0.93%
Mean Δ	-0.01	-0.94%	$0.03	0.98%
Std dev Δ	0.21	15.00%	$0.12	3.18%
Max Δ	0.93	84.55%	$0.46	10.87%

5.3.6. *Experimental Validation of OptionView on Real-World Resources*

Real-world validation has been performed on a set of UABgrid [8] resources. Individual tasks were parameterized as per results of OptionView and then submitted through GridWay, which was used as the job submission engine. Relevant hardware characteristics of employed resources are shown in Table 5. In the table, PSs refers to the maximum number of Processing Slots available for use. A PE refers to the smallest computational unit on given resource. With the available resources, the exhaustive

number of job execution options is 1,056. The selected sample n generated by OptionView, as per Equation (10), is 282. Therefore, 282 job execution options were simulated in GridSim and analyzed.

Because comprehensive validation of the generated options would require executing n parameterizations of the same BLAST job, for the validation purposes, a smaller, representative subset of job execution options was selected. This subset was selected in the same fashion as the selection process of the sample space generated by OptionView (*i.e.,* using Equation (10)). When using OptionView for meaningful computations, the user would obviously select only one such option to execute, namely the one that meets their utility most closely.

For our calculation of the validation sample space, the confidence level was set to 95% and confidence interval to 15%. Selected confidence interval value was selected in accordance to observed accuracy levels of application prediction tools (*e.g.,* [198]). As a result, from the 282 job execution options generated by OptionView, 37 represented the sample space and were randomly selected for execution on real-world resources. Each of the jobs was executed based on the parameterization obtained from the OptionView tool and runtime results were recorded.

Figure 55 presents the runtime results of executing selected job execution options on resources specified in Table 5. Because of the individualized parameterization of any one option, the behavior of the scheduling system can be verified by considering accuracy of runtime estimation for a given option and then generalized by the cumulative accuracy of all executed options. Because of the great level of detail regarding each job execution option, parameterization particulars on individual options that were executed are not

included here. Analysis of details at such low level is not necessary because, from the user's perspective, the interaction with OptionView takes place at the option level. When interacting with OptionView, the user only sees available job execution options without regard for the details as to how the option execution is implemented.

Figure 55. Experimental runtime data for real-world resources for selected job execution options. Circles represent job execution option runtime estimation generated by OptionView while x's represent observed runtime characteristics after job's execution on real-world resources as per instructions of the job plan generated by OptionView.

Based on the data in Figure 55, overall accuracy of individual options in real-world setting is quite satisfying. Data in Table 7 shows results of a statistical analysis of runtime accuracy across all executed options. The data shows average accuracy of the system within 2% and the maximum error within 10%. These results furthermore coincide with the results obtained through simulation, as described in the previous section.

Table 7. Statistical analysis of runtime accuracy across all executed job execution options

Average error	**Median error**	**Max error**
-1.43%	-2.13%	8.21%

Based on the experimental data, it can be concluded that job execution options calculated and presented through OptionView tool experience a high-level of accuracy.

This high-level of accuracy comes from the application-specific orientation adopted during option generation. By understanding the dependencies that exist between an application and resource, resource capabilities can be more adequately met resulting in high confidence regarding runtime estimation.

Overall, OptionView performs metascheduling actions across grid resources in a fully automated fashion. It performs such actions in application- and resource-specific manner and realizes notions behind user-oriented metascheduling. OptionView presents a revolutionary approach to grid application scheduling that has also shown a high-level of accuracy for application metascheduling in a real-world environment.

6. SUMMARY AND CONCLUSIONS

In the context of grid computing, where access to heterogeneous, dynamically available resources that can span multiple administrative domains is enabled, the process of selecting resources for running an application is an example of making a choice. It is a choice with a goal of meeting users' predetermined goals. This process can, however, present itself as a major challenge and can easily deny many of potential benefits an infrastructure offers [14]. This is because execution characteristics of user jobs depend on the individual resource in which an application runs. In order to simplify the overall resource allocation process, metaschedulers take on a task of selecting resources for a job's execution and thus abstract this step from the user.

6.1. Selected Highlights

This dissertation introduced the notion of user-driven choice into the field of metascheduling. Unlike other available approaches where much of the interaction between a user and the metascheduler is assumed, this work focuses on disclosing many of the options enabled by the grid infrastructure directly to the user. These choices are presented to the user in terms applicable to them directly (*i.e.,* by using metrics within the user's domain) instead of using terms otherwise native to the field.

6.1.1. *Contributions*

The contributions of this research build on established benefits of exploiting application-specific information to enable wide-spread adoption of *application-oriented metascheduling* (Section 3.5). This general approach was advanced to deliver application-

oriented solutions in the form of viable job execution options directly to users. Such a solution enables and delivers the concept of *user-oriented metascheduling* (Section 3.6). Overall, the described approach enables effective execution of applications across heterogeneous resources. Combined with the notion of user-oriented metascheduling, this approach enables individual users to make choices suitable to their current requirements.

The overall approach of application-oriented scheduling is realized in a set of core grid services, namely Application Information Services (AIS) that allow application-specific information to be captured and disseminated in a standardized fashion. By adopting and using AIS, especially at the level of a Virtual Organization, application-specific information can be collected, analyzed and finally consumed in a standardized fashion. The potential for AIS is to become an integral component of grid middleware as a set of core services, akin to GIS, and enable advantages discussed throughout this document on the scale of the grid. Such an approach would further enable development of tools (*e.g.*, metaschedulers, grid job managers, application performance analyzers) that can rely on existence of such information and become themselves more standardized, portable and application independent. At the same time, these tools can exploit advantages of relationships that exist between an application and resources. Benefits of such progression (*e.g.*, standardized interfaces, tool modularity) would include an ability of tools to transcend individual VO's and require fewer or no customizations each time deployed.

In addition to providing a set of services that enable application-oriented metascheduling, a framework for grid-enabling applications is presented (Section 4.1.2) and is considered a significant contribution of this work. The presented framework relies

on and exemplifies use of AIS to deliver benefits of grid environments to applications, and ultimately, the users. The framework outlines steps and provides best-practices toward simple yet effective transitioning of a sequential or cluster application into a grid application.

By exploiting application-specific information available through AIS to deliver application-oriented metascheduling, the model of interaction between a user and scheduler can be significantly altered. A user can operate entirely abstracted from low-level infrastructure details and can focus on task at hand. Through such an approach, a user becomes aware of alternatives and accepts the technology.

In research for this dissertation, application-oriented metascheduling was recognized and achieved by presenting users with a relevant subset of possible job execution alternatives for given job submission and resource availability. This is realized without requiring a user to provide much job-specific information to the metascheduler.

Concrete contributions of this work are seen as (in no particular order):

- A general purpose metascheduler model for EP applications
- Two instances of application-specific metaschedulers that instantiate a general purpose metascheduler
- A tool enabling user-oriented metascheduling,
- Taxonomy of EP applications
- A set of core grid services enabling storage and retrieval of application-specific information at grid level
- Examples of application performance analysis process

In addition, the presented work is tangentially related to the following two areas: (1) information within AIS can serve as an infrastructure for application runtime prediction in grid environments, (2) the metascheduling model realized can be complemented to support runtime job-plan modification and application migration.

Immediate benefits of the presented approach are as follows: (1) increased resource utilization by exploiting application-resource relationship, (2) direct benefit to users by delivering user-oriented metascheduling. As part of future potential, adoption of this approach and incorporation of grid-awareness into applications leads to development of self-scheduling, or smart applications that can heavily rely on availability of information about heterogeneous resources to accordingly adjust execution patterns.

6.1.2. *Validation*

Validation of the presented approach was gradually developed through experiments with individual components; when aggregated, individual components validate the overall approach and present a viable solution. These projects originated with planning and development of AIS. Information collected and made available through AIS was analyzed and used to derive a general purpose metascheduler model for EP applications. Lastly, devised methodology was applied through a novel approach to deliver the notion of job execution space and thus move away from a somewhat restrictive optimization problem into a tradeoff problem.

Accompanied with presentation of individual components and leading toward the final solution, each of the presented components was individually tested for performance. Obtained results show a high degree of success. Results also show benefit caused primarily by availability of information collected in AIS as well as the overall

methodology of iterative empirical model where data is collected, analyzed and then used to improve earlier conclusions.

6.1.3. *Future Directions*

Mirroring the general contributions by the work presented in this dissertation, devised results are directly applicable in two general directions. The initial direction is enablement of application-oriented scheduling through delivery of application-specific information that can be consumed by a metascheduler. The focus of information provided by AIS is at the level of tools and services in the grid software stack (see Figure 7). By enabling and standardizing collection and dissemination of application-specific and resource-specific information, tools can rely on such information's existence and be developed without having to construct custom and localized solutions.

Simultaneously, the part of this dissertation that deals with introducing and enabling two-way communication between a user and scheduler provides a higher level product that is purely focused on an individual user. The presented approach and solution aim to redefine the model of interaction between users and a scheduler as it is currently known. By providing users with a targeted set of job execution options, the level at which a user interacts with the infrastructure has moved from the 'infrastructural language' to 'user language', thus providing needed focus directly on users.

6.2. Vision

The work presented in this dissertation focuses on enabling use and access to heterogeneous and dynamic resources found across grid environments by users in a fashion that maximizes a user's utility for a given job. This is achieved through understanding and exploitation of relationships that exist between a resource and an

application. Overall, this field offers a large potential in terms of resource utilization and delivering higher QoS directly to users. Additional realizations of the presented approach, as well as future extensions, have a potential for significantly altering access and use patterns to grid and cloud resources. These realizations will have potential applicability in account management, runtime prediction, in addition to application and resource optimizations.

7. FUTURE WORK

This chapter presents some suggestions for extending this research. The first section focuses on extensions to existing tools and services in terms of functionality added and possible adoption scenarios. Section 7.2 focuses on more general application of presented methodologies and tools in the context of the emerging cloud computing. Finally, the last section of the chapter presents relatively more challenging ideas regarding a general metascheduling framework for grid applications that are currently under development.

7.1. Extensions

This section presents a set of possible extensions relating to automation of the overall process and applying the process to additional computational paradigms.

7.1.1. *Automating the Process*

Methods devised during the course of this research represent a detailed but nonetheless preliminary investigation into application-specific scheduling, and advances that such methods enable. Hand-in-hand with availability of application-specific information goes improvement of application execution characteristics. Consider an application-specific tool to set up execution of each application according to desired objectives. Such an approach requires complete automation of data collection, data analysis, and data interpretation processes. AIS represent a mechanism for collecting and retrieving available data, thus providing means for analyzing the same; a welcomed extension would incorporate data-mining and machine-learning techniques (*e.g.*,

knowledge extraction, supervised learning) to automate the overall process and enable development of higher level tools on a quicker basis.

As a possible direction to achieve and enable such an independent and coarse approach that still enables application-oriented metascheduling could involve application categorization (as presented in Section 2.3.1). Application categorization is an idea based on a notion of application similarity and execution implications for similar applications [26]. Categorization of applications can then be seen as a way to capture high-level detail that categorizes a class of applications and enables creation of best practices and insight into resource selection and job parameterization for that application category. This can lead to application category based performance comparison and materialization of future goals of delivering generalized yet application-oriented tools.

7.1.2. *Metascheduler Pool*

A related endeavor to notions of process automation and application categorization is whether such specific and targeted information extracted from application-resource relationships can be used to develop more customizable metascheduling models. Such metascheduling models would represent a pool of partially composed and configured tools that focus on a specific application or class of applications. A specific instance could then be developed quickly from the available pool. A concrete implication of this question is availability of a range of application-specific schedulers capable of exploiting application-resource relationships to result in more effective job execution strategies.

As a likely direction for development of such application-specific modules, one may foresee strong relationships and collaborative ties to software engineering; in particular, adaptive programming and model-driven programming. Techniques being developed in

these fields can enable high-level, effective implementation, adaptation and modification techniques for the derived models. These derived models could be utilized with little domain specific knowledge, or even by application scientists on an as-needed basis.

7.1.3. *Workflow Applications*

Building on top of solutions dealing with automated application and scheduling models, application workflows are direct extensions of current work. Many applied sciences describe solutions as a sets of tasks in which execution of some depend on executions of others. Specification of these relationships forms a computational workflow. While work has been done at optimizing ordering of tasks within workflows [199], combining presented work in application-specific scheduling with execution of individual tasks within a workflow has a large potential for two-level workflow optimizations, for which further customized execution characteristics of workflow jobs could occur. In Figure 56 a depiction of a sample workflow exhibits noticeable load imbalance among tasks across its execution stages. Application-oriented metascheduling techniques could be adopted to perform resource selection and parameter optimization that would work toward minimizing such load imbalance and lead to explained goals. One can easily imagine the level of impact a significant level of load imbalance can have on a multi-level, long-running workflow.

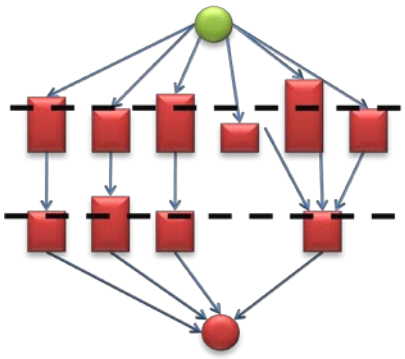


Figure 56. A sample of task-level workflow optimization

7.1.4. *Extensions Beyond EP Applications*

Because of lack of communication among individual tasks, a common assumption is that execution of an EP application in grid environments is easy or at least significantly easier than execution of lightly or tightly (*e.g.,* MPI) coupled applications (application categories 5 and 6 defined in Section 2.3.1). On the contrary, as evidenced through the case studies and examples provided earlier (specifically, in Section 4.2), execution of EP applications in grid environments is confined by resource availability, optimized through task parameterization, hindered by simultaneous use and management of heterogeneous resources belonging to different administrative domains, and dependent on user requirements. Consequently, an act of effective application execution inherently includes an act of application metascheduling. Because of multiple influencing components, metascheduling becomes a major component for effective grid execution of EP applications and should be handled comprehensively with respect to application execution environment variables and user desires.

The model of the EP class of applications possesses a versatility that can be used to model other classes of applications. This can be achieved by encapsulating those application classes within the EP class itself. A benefit of this feature of the EP class of applications is that it allows the same key metascheduling techniques to be applied to other application classes.

As an example, a sequential application can be represented by a single task that would also define the entire job within the EP application model. At that point, the metascheduling model is simplified because the requirement to minimize load imbalance (discussed on Section 4.1.2) is removed and only a direct comparison of individual

resources in terms of the given application needs to be performed. Similarly, because of general inability of MPI applications to cross individual cluster boundaries, a tightly coupled MPI application can be encapsulated within a single task and metascheduled as such. As the case is with the sequential applications, such an application instance can be metascheduled irrespective of other such instances. Therefore, metascheduling one such application job reduces to an ability to understand and leverage capabilities of individual resources from the perspective of an application, and then select the one resource that is the most likely to realize desired objective. With AIS, the ability to understand an application's requirements across resources is largely supported; the main difference in the metascheduling approach is selection of features that should be monitored, leveraged and controlled. For MPI applications with a goal of minimizing job runtime, examples of such features include a resource requirement to have high speed cluster interconnect. Alternatively, if a desired objective is cost minimization, one feature that might be considered is the number of processors that the application scales most effectively; that value would then transition into a requirement where selected resource would have that many processors available.

7.2. Moving into the Clouds

Cloud computing [31] is emerging as a major computational platform, abstracting traditional reliance on locally available hardware and its configuration into infrastructure-as-a-service (IaaS) and software-as-a-service (SaaS) paradigms. With such a paradigm shift, users are becoming disconnected from low-level hardware and software details, allowing users to focus on problems. To deliver advanced QoS to end users, together with resource owners, a need exists to customize and automate high performance application

execution on underlying resources across the entire cloud infrastructure. Such customization and automation requires understanding of an individual application's execution characteristics across individual resources, rooted in topics presented throughout this dissertation.

In a context of cloud computing, although basic resource usage rules are still mandated by a resource provider, user access is granted on one of two principles: (1) allow full access to a (virtual) resource through the IaaS paradigm, or (2) allow access to specific resource through the SaaS paradigm, hiding away any details not pertinent to the application service. Option 1 can be seen as a power-user option for users to explore low-level resource capabilities and leverage those to maximize application performance. Option 2 appeals to domain scientists primarily interested in computation results and not possessing necessary knowledge or time to invest into composing or tweaking an application for maximum performance. Additionally, it is in the interest of users as well as resource owners to maximize desired objectives (*i.e.,* turnaround time, cost, throughput, accuracy) of individual jobs. In a context of dynamic, heterogeneous resources offered within or across clouds, achieving desired objectives is non-trivial; there are existing dependencies between interacting components making manual adjustments inadequate, incomplete or not timely. High-level manifestations of poor support in today's clouds are observed through a need for a user to either manually setup an individual cloud resource, or to specify resource-specific parameters when interacting with the cloud through the SaaS interface. Such requirements signify inability of the cloud to effectively manage itself and simultaneously meet users' expectations.

Possible directions that aim at advancing the current state of computational clouds are as follows: (1) automate cloud job management to deliver application-oriented mappings of jobs to available resources, and (2) advance user interaction with a cloud by enabling automatic insight into user's job properties prior to job's submission. It can be safely stated that resources available within a cloud are heterogeneous in terms of their hardware characteristics. As shown in the Introduction Chapter as well as throughout Chapter 4, an application exhibits variable performance (typically in terms of runtime, but that can be translated into scalability, cost, accuracy or similar metrics) when executed across such heterogeneous resources. From a perspective of managing a cloud, because many factors influence job performance, it would be desirable to develop an understanding and a mechanism for job *cost and runtime control* (CRC) that would automatically understand application requirements and dependencies as well as underlying resource capabilities to a point where the two could be purposefully matched. CRC can then be realized as a set of job execution alternatives corresponding to different job parameters selected for each job alternative. Individual job parameters can be selected independently of other jobs or could be selected jointly across several jobs aiming at maximizing cumulative (resource owner) objective. Existence of such a mechanism would warrant detailed control over individual jobs, leading to effective management of a set of jobs and eventually the cloud. Concurrently, development of a CRC mechanism (and corresponding tools) would lead to support for improved user interaction methods where users are presented with results of CRC analysis and can easily choose between available and presented job execution alternatives. Such interaction with the cloud eliminates any need for users to specify or dwell on low-level job operating details,

allowing users to focus on objectives of direct impact, while the system automatically configures job execution parameters. Through application of these mechanisms, the true potential of cloud computing can be realized for resource owners as well as end users.

Although the optimal mapping of multiple applications' requirements to resource capabilities is a well-known NP-hard problem, there is a significant potential for improving general execution characteristics of applications across resources through the targeted study and subsequent matching of application requirements and resource capabilities. Combined with AIS and subsequent process automation, future developments could lead to cloud job management opportunities described in the previous paragraph. In order to accomplish set goals, the following steps are expected: aggregation of individual application information collection services into an interoperable unit that can be queried, analyzed and updated (through derivation of communication protocols); creation of a method for interpreting application and resource data leading to a composition of an Application-Resource Suitability Metric (ARSM); generalization of a computation method for job execution alternatives based on the ARSM; shift from individual application consideration toward multi-application job execution alternatives; development of a user interface to deliver described advances to users.

Overall, proposed extension and adoption would result in following, two-fold, goals: improvement in utilization of resources within a cloud, and a paradigm shift in terms of how users interact with a cloud or individual compute resources. Improvement in resource utilization results from a targeted mapping of jobs and tasks to resources yielding higher throughput of utilized resources. Derived improvement can be easily measured and compared to a resource allocation method unaware of application-resource

suitability. The user interaction component holds potential of significantly altering current users experience when interacting with the cloud.

7.3. Metascheduling-as-a-Service (MaaS)

Thus far, all discussion has been oriented around existing applications where functionality offered by metascheduling is an add-on service applied at job submission. This model applied well to existing and legacy applications whose reengineering proves to be too time consuming and costly. However, as the grid and cloud paradigms emerge and mature, more and more applications will be developed for general infrastructure; standards are emerging that foster easy grid-application development [49]. At that point, a shift in the overall metascheduling approach can be realized. In this new paradigm, instead of the metascheduler conforming to the application, the application can conform to a metascheduler.

More concretely, a metascheduling framework can be developed to provide a standardized set of services and corresponding functionality (*e.g.*, similar to the one depicted in Figure 22). At that point, an application can be developed with 'hooks' that are compatible with the general functionality offered by the metascheduling framework. Once deployed, applications can expose internal information to the metascheduling functionality; the metascheduler can consume application-specific information and return an application-oriented job plan. At that point, the application can be seen as enjoying application-oriented Metascheduling-as-a-Service (MaaS).

LIST OF REFERENCES

[1] J. Morris, "Programming doesn't begin to define computer science," in *Post-Gazette* Pittsburgh, PA, 2004.

[2] *The Grid: Blueprint for a New Computing Infrastructure*, 1st ed.: Morgan Kaufmann Publishers, 1998.

[3] A. D. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. New York: Cambridge University Press, 2008.

[4] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, CA: The Benjamin/Cummings Publishing Company, 1994.

[5] F. Berman, G. Fox, and T. Hey, "The Grid: past, present, future," in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds., Hoboken, NJ: John Wiley & Sons Inc., 2003, pp. 9-51.

[6] I. Foster, "What is the Grid? A Three Point Checklist," July 22, 2002, Available at http://www.gridtoday.com/02/0722/100136.html, Retrieved: May 31st, 2004.

[7] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid," *Lecture Notes in Computer Science*, 2150(2001, pp. 1-28.

[8] J. Gemmill and P. Bangalore, "UABGrid - A campus-wide distributed computational infrastructure," Birmingham, AL: UAB, 2006, p. 5.

[9] "TeraGrid," March 19, 2009, Available at http://www.teragrid.org, Retrieved: March 19, 2009.

[10] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web Services Architecture," *IBM Systems Journal*, 41(2), 2002, pp. 170-178.

[11] I. Foster, "The Grid: A New Infrastructure for 21st Century Science," *Physics Today*, 55(2), April 2002, pp. 42-47.

[12] I. Foster and C. Kesselman, Eds. *The Grid 2*, Second ed., New York: Morgan Kaufmann, 2004.

[13] C. Mateos, A. Zunino, and M. Campo, "A survey on approaches to gridification," *Software—Practice & Experience*, 38(5), April 2008, pp. 523-556.

[14] E. Afgan and P. Bangalore, "Embarrassingly Parallel Jobs Are Not Embarrassingly Easy to Schedule on the Grid," in *SC08 International Conference*

for High Performance, Networking, Storage and Analysis - Workshop on Many-Task Computing on Grids and Supercomputers Austin, TX, 2008, p. 10.

[15] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Mol Biol,* 215(3), Oct 5 1990, pp. 403-410.

[16] G. Tsouloupas and M. D. Dikaiakos, "Grid Resource Ranking Using Low-Level Performance Measurements," in *13th International Euro-Par Conference 2007 on Parallel Processing*, Rennes, France, 2007, pp. 467-476.

[17] E. Afgan and P. Bangalore, "Performance Characterization of BLAST for the Grid," in *IEEE 7th International Symposium on Bioinformatics & Bioengineering (IEEE BIBE 2007)* Boston, MA, 2007, pp. 1394-1398.

[18] C. B. Lee and A. Snavely, "On the User–Scheduler Dialogue: Studies of User-Provided Runtime Estimates and Utility Functions," *International Journal of High Performance Computing Applications,* 20(4), Winter 2006, pp. 495-506.

[19] A. D. Baxevanis and F. Ouellette, Eds. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins,* 3rd ed., New Jersey: Wiley-Interscience, 2004.

[20] T. Smith and M. Waterman, "Identification of Common Molecular Subsequences," *Journal of Molecular Biology,* 147), 1981, pp. 195-197.

[21] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological sequence analysis: probabilistic models of proteins and nucleic acids*: Cambridge University Press, 1998.

[22] A. E. Darling, L. Carey, and W.-c. Feng, "The Design, Implementation, and Evaluation of mpiBLAST," in *ClusterWorld Conference & Expo in conjunction with the 4th International Conference on Linux Clusters: The HPC Revolution 2003*, San Jose, CA, 2003.

[23] C.-W. T. Xue Wu, "Searching Sequence Databases Using High-Performance BLASTs," in *Parallel Computing for Bioinformatics and Computational Biology*, Y. Z. Albert, Ed.: John Wiley & Sons, 2006, pp. 211-232.

[24] E. Afgan and P. Bangalore, "Assisting Efficient Job Planning and Scheduling in the Grid," in *Handbook of Research on Grid Technologies and Utility Computing: Concepts for Managing Large-Scale Applications*, E. Udoh and F. Z. Wang, Eds.: IGI Global, 2009, pp. 22-31.

[25] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," in *10 th IEEE Symp. On High Performance Distributed Computing (HPDC)*, Los Alamitos, CA, 2001, pp. 181-195.

[26] G. Tsouloupas and M. Dikaiakos, "GridBench: A Tool for Benchmarking Grids," in *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, AZ, 2003, pp. 60-67.

[27] F. Sanchez, E. Salami, A. Ramirez, and M. Valero, "Performance Analysis of Sequence Alignment Applications," in *2006 IEEE International Symposium on Workload Characterization*, San Jose, CA, 2006, pp. 51-60.

[28] F. D. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks," in *Supercomputing '96*, Pittsburgh, PA, 1996, p. 28.

[29] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, J. M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan, "New grid scheduling and rescheduling methods in the GrADS project," *International Journal of Parallel Programming*, 33(2), June 2005, pp. 209-229.

[30] R. Buyya, M. Murshed, D. Abramson, and S. Venugopa, "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm," *Software - Practice & Experience*, 35(5), April 2005, pp. 491-512.

[31] G. Gruman and E. Knorr, "What cloud computing really means," in *InfoWorld*, 2008, p. 2.

[32] W. Stallings, *Operating systems: internals and design principles*, 6th ed.: Prentice Hall, 2009.

[33] L. Kleinrock, "Network is the computer.," ARPANET project, 1969.

[34] OGF, "Open Grid Forum," 2009, Available at http://www.ogf.org, Retrieved: March 19, 2009.

[35] I. Foster and C. Kesselman, "The Globus toolkit," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, Eds., San Francisco, California: Morgan Kaufmann, 1999, pp. 259--278.

[36] D. W. Erwin and D. F. Snelling, "UNICORE: A Grid Computing Environment," *Lecture Notes in Computer Science*, 2150(2001, pp. 825-834.

[37] F. Dvořák, D. Kouřil, A. Křenek, L. Matyska, M. Mulač, J. Pospíšil, M. Ruda, Z. Salvet, J. Sitera, and M. Voců, " gLite Job Provenance," in *Provenance and Annotation of Data*: Springer, 2006, pp. 246-253.

[38] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, "Alchemi: A .NET-Based Enterprise Grid Computing System," in *6th International Conference on Internet Computing (ICOMP'05)*, Las Vegas, NV, 2005.

[39] S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw, "The Legion Resource Management System " in *International Parallel and Distributed Processing Symposium (IPDPS '99) - 5th Workshop on Job Scheduling Strategies for Parallel Processing*, San Juan, Puerto Rico, 1999, pp. 162-178.

[40] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *5th ACM Conference on Computer and Communication Security Conference*, San Francisco, CA, 1998, pp. 83-92.

[41] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62-82.

[42] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 28(5), May 2001, pp. 749 - 771.

[43] F. Berman, "High-performance schedulers," in *The grid: blueprint for a new computing infrastructure*, I. Foster and C. Kesselman, Eds., San Francisco, CA: Morgan Kaufmann Publishers Inc., 1998, pp. 279 - 309.

[44] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," in *IEEE Symposium on High Performance Distributed Computing (HPDC10)*, San Francisco, CA, 2001, p. 9.

[45] G. Allen, T. Goodale, M. Russell, E. Seidel, and J. Shalf, "Classifying and Enabling Grid Applications," in *Grid Computing: Making the Global Infrastructure a Reality*, F. Berman, G. Fox, and T. Hey, Eds.: John Wiley & Sons, 2003, pp. 601-614.

[46] "LHC - The Large Hadron Collider," July 24, 2008, Available at http://lhc.web.cern.ch/lhc/, Retrieved: January 26, 2009.

[47] Z. Schreiber, "G.ho.st," September 8, 2008, Available at http://G.ho.st, Retrieved: January 26, 2009.

[48] T. V. Raman, "Toward 2w, beyond web 2.0," *Communications of ACM*, 52(2), February 2009, pp. 52-59.

[49] SAGA-RG, "Simple API for Grid Applications RG " December 18, 2008, Available at http://forge.ogf.org/sf/projects/saga-rg, Retrieved: January 26, 2009.

[50] D. Gannon, G. Fox, M. Pierce, B. Plale, G. v. Laszewski, C. Severance, J. Hardin, J. Alameda, M. Thomas, and J. Boisseau, "Grid Portals: A Scientist's Access Point for Grid Services," Global Grid Forum (GGF) September 19 2003.

[51] W. v. d. Aalst and K. M. v. Hee, *Workflow Management: Models, Methods, and Systems*, 1st ed. Cambridge, MA: MIT Press, 2002.

[52] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski, "Toward a Common Component Architecture for High-Performance Scientific Computing," in *The High-Performance Distributed Computing Conference (HPDC)*, Redondo Beach, CA, 1999, pp. 115-124.

[53] E. Afgan, P. Bangalore, and J. Gray, "Configuring Grid Applications from Higher Level Domain-Specific Languages," in *Designing Software-Intensive Systems: Methods and Principles*, P. F. Tiako, Ed., Langston, OK, 2007, pp. 402-439.

[54] M. P. I. Forum, "MPI message-passing interface standard Vers. 2.0," 1998, Available at http://www.mpi-forum.org/docs/docs.html, Retrieved: May 18th, 2004.

[55] N. Karonis, B. Toonen, and I. Foster, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface," *Journal of Parallel and Distributed Computing (JPDC)*, 63(5), May 2003, pp. 551-563.

[56] "The Globus Resource Specification Language RSL v1.0," 01/03/2005, Available at http://www-fp.globus.org/gram/rsl_spec1.html, Retrieved: April 6, 2006.

[57] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, "Job Submission Description Language (JSDL) Specification, Version 1.0," Global Grid Forum (GGF), Technical Report GFD-R.056, 7 November 2005.

[58] "Job Submission Description Language Working Group (JSDL-WG)," July 12 2005, 2004, Available at https://forge.gridforum.org/projects/jsdl-wg/, Retrieved: April 6, 2006.

[59] W3C, "RDF Primer," 10 February, 2004, Available at http://www.w3.org/TR/rdf-primer/, Retrieved: March 29, 2007.

[60] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, 2001.

[61] C. Systinet, "Introduction to Web Services Architecture," Cambridge, MA, 2002, p. 22.

[62] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The WS-Resource Framework," OASIS March 5 2004.

[63] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, I. Sedukhin, D. Snelling, T. Storey, W. Vambenepe, and S. Weerawarana, "Modeling Stateful Resources with Web Services," OASIS March 5 2004.

[64] S. Graham, P. Niblett, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, and B. Weihl, "Publish-Subscribe Notification for Web services," OASIS March 5 2004.

[65] "Web Services Addressing," August 10 2004, 2004, Available at http://www-128.ibm.com/developerworks/library/specification/ws-add/, Retrieved: April 6, 2006.

[66] L. Cooper and D. Steinberg, *Methods and Applications of Linear Programming*. Philadelphia: W.B. Saunders Company, 1974.

[67] J. Y.-T. Leung, Ed. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, 1st ed., vol. 1: CRC Press, 2004.

[68] J. M. Schopf, "Ten Actions when Grid Scheduling," Argonne National Laboratory 2005.

[69] S. B. Wellington, "Road to grid computing remains difficult," in *ComputerWorld*, 2009, p. 1.

[70] V. Systems, "OpenPBS v2.3: The Portable Batch System Software," 2004.

[71] "N1 Grid Engine 6 User's Guide," Sun Microsystems May 2005.

[72] S. Zhou, "LSF: Load Sharing in Large-scale Heterogeneous Distributed Systems," in *Workshop on Cluster Computing*, 1992.

[73] S. Kannan, M. Roberts, P. Mayes, D. Brelsford, and J. F. Skovira, "Workload Management with LoadLeveler," IBM SG24-6038-00, November 2001.

[74] J. Broberg, S. Venugopal, and R. Buyya, "Market-oriented Grids and Utility Computing: The State-of-the-art and Future Directions," *Journal of Grid Computing*, 5(4), December 28 2007,

[75] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide Area Cluster Monitoring with Ganglia," in *IEEE Cluster 2003*, Hong Kong, 2003.

[76] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Journal of Future Generation Computing Systems*, 15(5-6), October 1999, pp. 757-768.

[77] H. B. Newman, I. C. Legrand, P.Galvez, R. Voicu, and C. Cirstoiu, "MonALISA: A Distributed Monitoring Service Architecture," in *CHEP 2003*, La Jola, CA, 2003, p. 8.

[78] P. Z. Kunszt and L. P. Guy, "The Open Grid Services Architecture, and Data Grids," in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, A. Hey, and G. Fox, Eds., Hoboken, NJ: John Wiley & Sons Inc., 2003, p. 24.

[79] A. Kertesz and P. Kacsuk, " A Taxonomy of Grid Resource Brokers," in *Distributed and Parallel Systems from Cluster to Grid Computing*, 1 ed, P. Kacsuk, T. Fahringer, and Z. Németh, Eds.: Springer, 2007, pp. 201-210.

[80] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, 25(6), June 2009, pp. 599-616.

[81] C. Smith, "Open source metascheduling for Virtual Organizations with the Community Scheduler Framework (CSF)," Platform Computing, Whitepaper August 2003.

[82] "CSF," Available at http://www.globus.org/grid_software/computation/csf.php, Retrieved: April 6, 2006.

[83] J. Mausolf, "Use Community Scheduler Framework to implement grid meta-schedulers," IBM July 24 2004.

[84] F. Berman and R. Wolski, "Scheduling from the Perspective of the Application," in *High Performance Distributed Computing Conference (HPDC)*, Syracuse, NJ, 1996, pp. 100-111.

[85] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid," in *Supercomputing 2000*, Dallas, TX, 2000.

[86] H. Casanova and F. Berman, "Parameter sweeps on the Grid with APST," in *Grid Computing: Making the Global Infrastructure a Reality*, B. F., F. G., and H. T., Eds., Hoboken, NJ: John Wiley & Sons Inc., 2003, pp. 773-789.

[87] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," *International Journal of High Performance Computing Applications*, 15(4), Winter 2001, pp. 327-344.

[88] H. Dail, F. Berman, and H. Casanova, "A Decoupled Scheduling Approach for Grid Application Development Environments," *Journal of Parallel and Distributed Computing*, 63(5), May 2003, pp. 505-524.

[89] M. Litzkow, M. Livny, and M. Mutka, "Condor - A Hunter of Idle Workstations," in *8th International Conference of Distributed Computing Systems*, 1988.

[90] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid," in *Grid Computing: Making The Global Infrastructure a Reality*, B. F., F. G., and H. T., Eds., Hoboken, NJ: John Wiley & Sons Inc, 2003, pp. 299-337.

[91] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," in *Seventh IEEE International Symposium on High Performance Distributed Computing*, 1998, p. 7.

[92] "Classified Advertisements," Available at http://www.cs.wisc.edu/condor/classad/, Retrieved: April 6, 2006.

[93] D. Abramson, R. Sosic, J. Giddy, and B. Hall, "Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations," in *High Performance Distributed Computing (HPDC)*, 1995, pp. 112-121.

[94] R. Buyya, D. Abramson, and J. Giddy, "Nimrod-G: An Architecture for a Resource Management and Scheduling in a Global Computational Grid," in *4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region (HPC ASIA 2000)*, Beijing, China, 2000, pp. 283-289.

[95] R. Buyya, J. Giddy, and D. Abramson, "An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications," in *The Second Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000)*, Pittsburgh, PA, 2000, p. 10.

[96] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid," in *International Parallel and Distributed Processing Symposium (IPDPS)*, Cancun, Mexico, 2000, pp. 520-528.

[97] S. Venugopal, R. Buyya, and L. Winton, "A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids," *Journal of Concurrency and Computation: Practice and Experience*, 18(6), Nov 8 2005, pp. 685-699.

[98] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, "The Data Grid: Towards an architecture for the distributed management and analysis of large scientific datasets," *Journal of Network and Computer Applications*, 23(3), 2000, pp. 189-2000.

[99] K. R. A. I. Foster, "Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications," in *11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11)*, Edinburgh, Scotland, 2002.

[100] R. Buyya, D. Abramson, and J. Giddy, "An Economy Driven Resource Management Architecture for Global Computational Power Grids," in *The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, NV, 2000.

[101] A. Bose, B. Wickman, and C. Wood, "MARS: A Metascheduler for Distributed Resources in Campus Grids," in *5th ACM/IEEE International Workshop on Grid Computing*, 2004, pp. 110-118.

[102] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*, 2nd ed.: Springer, 2003.

[103] L. Hall, A. Schulz, D. Shmoys, and J. Wein, "Scheduling To Minimize Average Completion Time: Off-line and On-line Algorithms," in *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, Atlanta, GA, 1996, pp. 142 - 151.

[104] M. Maheswaran, S. Ali, H. J. Siegel, D. A. Hensgen, and R. F. Freund, "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems," in *Heterogeneous Computing Workshop*, 1999, pp. 30-45.

[105] "GridWay Metascheduler: Metascheduling Technologies for the Grid," April, 2006, Available at http://www.gridway.org/, Retrieved: April 7, 2006.

[106] "An "Ecosystem" of Grid Components," Available at http://www.globus.org/grid_software/ecology.php, Retrieved: April 7, 2006.

[107] H. Rajic, R. Brobst, W. Chan, F. Ferstl, J. Gardiner, A. Haas, B. Nitzberg, H. Rajic, and J. Tollefsrud, "Distributed Resource Management Application API (DRMAA) Specification 1.0," Global Grid Forum (GGF) GFD-R-P.022, June 2004.

[108] E. Afgan and P. Bangalore, "Experiences with developing and deploying dynamic BLAST " in *15th ACM Mardi Gras conference, Workshop on Grid-Enabling Applications*, Baton Rouge, LA, 2008, pp. 38-48.

[109] R. S. Montero, E. Huedo, and I. M. Llorente, "Grid Scheduling Infrastructures based on the GridWay Meta-scheduler," *IEEE Technical Committee on Scalable Computing (TCSC) Newsletter*, 8(2), 2006,

[110] R. S. Montero and I. M. Llorente, "Scheduling Policies in the GridWay System," GridWay Project January 2007.

[111] E. Huedo, R. S. Montero, and I. M. LLorente, "A Framework for Adoptive Execution on Grids," *Journal of Software - Practice and Experience,* 34(7), March 2004, pp. 631-651.

[112] I. Taylor, E. Deelman, D. Gannon, and M. Shields, *Workflows for e-Science*, 1st ed.: Springer Verlag, 2006.

[113] M. W. Carter and C. C. Prince, *Operations Research, A Practical Foundation*. Boca Raton, FL: CRC Press, 2001.

[114] J. J. Kanet, S. L. Ahire, and M. F. Gorman, "Constraint Programming for Scheduling," in *Handbook of Scheduling*, J. Y.-T. Leung, Ed., Boca Raton, FL: CRC Press, 2004, pp. 47-1->47-21.

[115] J. Yu, M. Kirley, and R. Buyya, "Multi-objective planning for workflow execution on Grids," in *Grid 2007*, Austin, TX, 2007, pp. 10-17.

[116] R. Duan, R. Prodan, and T. Fahringer, "Performance and Cost Optimization for Multiple Large-scale Grid Workflow Applications," in *Supercomputing 2007 (SC|07)*, Reno, NE, 2007.

[117] M. Wieczorek, S. Podlipnig, R. Prodan, and T. Fahringer, "Bi-criteria Scheduling of Scientific Workflows for the Grid," in *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (ccGrid)*, Lyon, France, 2008, pp. 9-16.

[118] E. Huedo, R. S. Montero, and I. M. Llorente, "A Framework for Adaptive Execution on Grids," *Journal of Software - Practice and Experience,* 34(7), June 2004, pp. 631-651.

[119] R. Buyya, D. Abramson, and J. Giddy, "Nimrod-G Resource Broker for Service-Oriented Grid Computing," 2004, Available at http://www.computer.org/portal/site/dsonline/menuitem.9ed3d9924aeb0dcd82ccc6716bbe36ec/index.jsp?&pName=dso_level1&path=dsonline/past_issues/0107/departments&file=res0107_print.xml&xsl=article.xsl&;jsessionid=LpLsZC1FRLjGhW22G5CCsqpDyTPvpSht3BD76bpTKLn1pv2BBYgG!1195711095, Retrieved: 8/18, 2008.

[120] F. P. Brooks, "No Silver Bullet-Essence and Accidents of Software," *IEEE Computer,* 20(4), April 1987, pp. 10-19.

[121] A. Downey, "Predicting Queue Times on Space-Sharing Parallel Computers," in *International Parallel Processing Symposium (IPPS '97)*, Geneva, Switzerland, 1997, pp. 209-218.

[122] R. Gibbons, "A Historical Application Profiler for Use by Parallel Schedulers," *Lecture Notes In Computer Science,* 1291(1997, pp. 58-77.

[123] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information," in *Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 122-142.

[124] V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, R. Stevens, M. Hereld, and I. R. Judson, "Prophesy: An Infrastructure for Analyzing and Modeling the Performance of Parallel and Distributed Applications," in *High Performance Distributed Computing (HPDC) 2000*, Pittsburgh, PA, 2000, pp. 302-303.

[125] X. Wu, V. Taylor, and J. Paris, "A Web-based Prophesy Automated Performance Modeling System," in *The IASTED International Conference on Web Technologies, Applications and Services (WTAS 2006)*, Calgary, Canada, 2006.

[126] X. Wu, V. Taylor, J. Geisler, X. Li, Z. Lan, R. Stevens, M. Hereld, and I. R. Judson, "Design and Development of Prophesy Performance Database for Distributed Scientific Applications," in *10th SIAM Conference on Parallel Processing for Scientific Computing*, Portsmouth, VA, 2001.

[127] V. Taylor, X. Wu, J. Geisler, X. Li, Z. Lan, M. Hereld, I. Judson, and R. Stevens, "Prophesy: Automating the Modeling Process," in *2001 IEEE International Symposium on Performance Analysis of Systems and Software*, Tucson, AZ, 2001.

[128] X. Wu, V. Taylor, J. Leigh, and L. Renambot, "Performance Analysis of a 3D Parallel Volume Rendering Application on Scalable Tiled Displays," in *International Conference on Computer Graphics, Imaging and Vision (CGIV05)*, Beijing, China, 2005.

[129] X. Wu, V. Taylor, S. Garrick, D. Yu, and J. Richard, "Performance Analysis, Modeling and Prediction of a Parallel Multiblock Lattice Boltzmann Application Using Prophesy System," in *IEEE International Conference on Cluster Computing*, Barcelona, Spain, 2006.

[130] F. Nadeem, R. Prodan, T. Fahringer, and A. Iosup, "Benchmarking Grid Applications for Performance and Scalability Predictions," in *CoreGRID 2007 Workshop on Middleware*, Dresden, Germany, 2007, p. 14.

[131] D. A. Bader, Y. Li, T. Li, and V. Sachdeva, "BioPerf: A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications," in *The IEEE International Symposium on Workload Characterization (IISWC 2005)*, Austin, TX, 2005, pp. 163-173.

[132] NCBI, "GenBank Statistics," February 3, 2009, Available at http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, Retrieved: March 17, 2009.

[133] A. Ping, A. Jula, S. Rus, S. Saunders, T. Smith, G. Tanase, N. Thomas, N. Amato, and L. Rauchwerger, "STAPL: An adaptive, generic parallel c++ library," in

Workshop on Languages and Compilers for Parallel Computing (LCPC), Cumberland Falls, KY, 2001, p. 17.

[134] N. Thomas, G. Tanase, O. Tkachyshyn, J. Perdue, N. Amato, and L. Rauchwerger, "A Framework for Adaptive Algorithm Selection in STAPL," in *ACM SIGPLAN 2005 Symposium on Principles and Practices of Parallel Programming (PPoPP)*, Chicago, IL, 2005.

[135] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Upper Sadle River, NJ: Prentice Hall, 2002.

[136] T. Mitchell, *Machine Learning*: McGraw Hill, 1997.

[137] S. Sodhi and J. Subhlok, "Automatic Construction and Evaluation of Performance Skeletons," in *19th International Parallel and Distributed Processing Symposium (IPDPS '05)*, Denver, CO, 2005, p. 10.

[138] E. Afgan and B. Purushotham, "Embarrassingly Parallel Jobs Are Not Embarrassingly Easy to Schedule on the Grid," in *International Conference for High Performance, Networking, Storage and Analysis (SC08) - Workshop on Many-Task Computing on Grids and Supercomputers* Austin, TX, 2008, p. 10.

[139] *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers.* 2.0 ver. 2008.

[140] "Standard Performance Evaluation Corporation," July 24, 2008, Available at http://www.spec.org/, Retrieved: July 25, 2008.

[141] A. Iosup, C. Dumitrescu, D. H. Epema, H. Li, and L. Wolters, "How are real grids used? The analysis of four grid traces and its implications," in *International Conference on Grid Computing 2006*, Barcelona, Spain, 2006, pp. 262-269.

[142] I. Banicescu and Z. Liu., "Adaptive Factoring: A Dynamic Scheduling Method Tuned to the Rate of Weight Changes," in *High Performance Computing Symposium (HPC 2000)*, Washington, D.C., 2000, pp. 122-129.

[143] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd, "Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling," in *17th International Symposium on Parallel and Distributed Processing (IPDPS) 2003*, Nice, France, 2003, p. 49.2.

[144] R. U. Payli, E. Yilmaz, A. Ecer, H. U. Akay, and S. Chien, "DLB – A Dynamic Load Balancing Tool for Grid Computing," in *Parallel CFD Conference*, Grand Canaria, Canary Islands, Spain, 2004, p. 8.

[145] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI'04: Sixth Symposium on Operating System Design and Implementation* San Francisco, CA, 2004, p. 13.

[146] B. Bergeron, *Bioinformatics Computing* 1st ed. Upper Saddle River, New Jersey: Prentice Hall PTR, 2002.

[147] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res,* 25(17), Sep 1 1997, pp. 3389-3402.

[148] W. R. Pearson and D. J. Lipman, "Improved Tools for Biological Sequence Comparison," *Proceedings of the National Academy of Sciences of the United States of America (PNAS),* 85(16), August 15 1988, pp. 2444- 2448.

[149] H. G. Program, "What is the Human Genome Project?," December 07, 2005, Available at http://www.ornl.gov/sci/techresources/Human_Genome/project/about.shtml Retrieved: April 28, 2008.

[150] R. D. Bjomson, A. H. Sherman, S. B. Weston, N. Willard, and J. Wing, "TurboBLAST: A Parallel Implementation of BLAST Built on the TurboHub," in *International Parallel and Distributed Processing Symposium: IPDPS 2002*, Ft. Lauderdale, FL, 2002.

[151] N. Camp, H. Cofer, and R. Gomperts, "High-Throughput BLAST," SGI September 1998.

[152] NCBI, "BLAST: Basic Local Alignment Search Tool," April 25, 2008, Available at http://blast.ncbi.nlm.nih.gov/Blast.cgi Retrieved: April 28, 2008.

[153] NCBI, "BLAST Frequently Asked questions," 2008, Available at http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Web&PAGE_TYPE=BlastFAQs#sigxcpu Retrieved: April 28, 2008.

[154] E. Afgan, P. Sathyanarayana, and P. Bangalore, "Dynamic Task Distribution in the Grid for BLAST," in *Granular Computing (GrC 2006)*, Atlanta, GA, 2006, pp. 554-557.

[155] D. Sulakhe, A. Rodriguez, M. D'Souza, M. Wilde, V. Nefedova, I. Foster, and N. Maltsev, "GNARE: An Environment for Grid-Based High-Throughput Genome Analysis," in *Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, Cardiff, UK, 2005 pp. 455 - 462.

[156] M. K. Gardner, W.-c. Feng, J. Archuleta, H. Lin, and X. Ma, "Parallel genomic sequence-searching on an ad-hoc grid: experiences, lessons learned, and implications," in *Supercomputing, 2006 (SC '06)*, Tampa, FL, 2006, pp. 22-36.

[157] R. D. C. Team, "R: A Language and Environment for Statistical Computing," R Foundation for Statistical Computing, Vienna, Austria 2005.

[158] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima, "Performance evaluation model for scheduling in a global computing system," *The International Journal of High Performance Computing Applications*, 14(3), Fall 2000, pp. 268-279.

[159] H. Song, X. Liu, D. Jakobsen, R. Bhagwan, X. Zhang, K. Taura, and A. Chien, "The MicroGrid: a Scientific Tool for Modeling Computational Grids," in *IEEE Supercomputing (SC 2000)*, Dallas, TX, 2000.

[160] H. Xia, H. Dail, H. Casanova, and A. A. Chien, "The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments," in *Proceedings of the Second International Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, Honolulu, HI, 2004, pp. 52- 61.

[161] H. Casanova, "SimGrid: A Toolkit for the Simulation of Application Scheduling," in *First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)*, Brisbane, Australia, 2001.

[162] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), Nov-Dec 2002, pp. 1175-1220.

[163] J. Horst, E. Messina, T. Kramer, and H.-M. Huang, "Precise definition of software component specifications," in *7th Symposium on Computer-Aided Control System Design (CACSD '97)*, Gent, Belgium, 1997, pp. 145-150.

[164] A. v. Hoff, H. Partovi, and T. Thai, "The Open Software Description Format (OSD)," W3C August 11 1997.

[165] F. A. Hernandez, "GAUGE: Grid Automation and Generative Environment Using Domain Engineering and Domain Modeling for Drafting Applications for the Grid," in *Department of Computer and Information Sciences* Birmingham, AL: University of Alabama at Birmingham (UAB), 2006, p. 188.

[166] C. Letondal, "A Web interface generator for molecular biology programs in Unix," *Bioinformatics*, 17(1), July 7 2000, pp. 73-82.

[167] E. Afgan, "Role of the Resource Broker in the Grid," in *42 nd Annual ACM Southeast Conference*, Huntsville, AL, 2004, pp. 299-300.

[168] B. N. Chun and D. E. Culler, "User-Centric Performance Analysis of Market-Based Cluster Batch Schedulers," in *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid) 2002*, Berlin, Germany, 2002, pp. 30-39.

[169] E. Afgan and B. Purushotham, "Computation Cost in Grid Computing Environments," in *First International Workshop on the Economics of Software and Computation in conjunction with International Conference on Software Engineering (ICSE) 2007*, Minneapolis, MN, 2007.

[170] E. Afgan and P. Bangalore, "Application Specification Language (ASL) – A Language for Describing Applications in Grid Computing," in *4th International Conference on Grid Service Engineering and Managemen (GSEM 2007)*, Leipzig, Germany, 2007.

[171] E. Afgan, P. Bangalore, S. Mukkai, and S. Yammanuru, "Design and Implementation of a Readily Available Historical Application Performance Database (AppDB) for Grid," University of Alabama at Birmingham (UAB), Birmingham, AL UABCIS-TR-2008-0506-1, May 6 2008.

[172] E. Afgan, P. Bangalore, and S. Mukkai, "GridAtlas," December 18, 2008, Available at http://www.cis.uab.edu/ccl/index.php/GridAtlas, Retrieved: February 1, 2009.

[173] E. Afgan and P. Bangalore, "Application Specification Language (ASL) – A Language for Describing Applications in Grid Computing," in *The 4th International Conference on Grid Services Engineering and Management - GSEM 2007* Leipzig, Germany, 2007, pp. 24-38.

[174] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, 30(7), July 2004, pp. 817-840.

[175] "Vampir - Performance Optimization," Nov 16, 2008, Available at http://www.vampir.eu/, Retrieved: Jan 20, 2009.

[176] TeraGrid, "TeraGrid Common Environment Variables," 2009, Available at http://www.teragrid.org/userinfo/jobs/variables.php, Retrieved: May 19, 2009.

[177] Sun Microsystems, Inc., *MySQL*. 5.1 ver. 2009.

[178] Hibernate, "Relational Persistence for Java and .NET," 2008, Available at http://www.hibernate.org/, Retrieved: May 9, 2008.

[179] The Apache Software Foundation, *Apache Tomcat.* 6.0.18 ver. 2009.

[180] The Apache Software Foundation, *Apache Axis.* 1.4 ver. 2005.

[181] E. Afgan, P. Bangalore, and D. Duncan, "GridAtlas - A Grid Application and Resource Configuration Repository and Discovery Service," in *Submitted for review to IEEE Cluster 2009*, New Orleans, LA, 2009, p. 10.

[182] S. B. Emma and A. R. Daniel, "Analysis of application heartbeats: learning structural and temporal features in time series data for identification of performance problems," in *2008 ACM/IEEE conference on Supercomputing (SC|08)* Austin, Texas: IEEE Press, 2008, p. 12.

[183] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison Wesley Publication Company, 1994.

[184] W. Cirne and F. Berman, "A Model for Moldable Supercomputer Jobs," in *Parallel and Distributed Processing Symposium (IPDPS) 2001*, San Francisco, CA, 2001, pp. 8-16.

[185] N. Dale and D. Teague, *C++ Plus Data Structures*, 2nd ed.: Jones & Bartlett Publishers, 2001.

[186] A. Tirado-Ramos, G. Tsouloupas, M. Dikaiakos, and P. Sloot, "Grid Resource Selection by Application Benchmarking for Computational Haemodynamics Applications," in *International Conference on Computational Science (ICCS) 2005*, Kassel, Germany, 2005, pp. 534-543.

[187] X. Pillons, "Running HPL on Windows HPC Server 2008," January 2008.

[188] S. N. Kandadai, "Tuning tips for HPL on IBM xSeries Linux Clusters," IBM 2003.

[189] E. Huedo, R. S. Montero, and I. M. Llorente, "A Framework for Adaptive Execution on Grids," *Journal of Software - Practice and Experience*, 34(2004, pp. 631-651.

[190] E. Afgan and P. Bangalore, "Performance Characterization of BLAST for the Grid," Boston, MA, 2007.

[191] A. E. Darling, L. Carey, and W.-c. Feng, "The Design, Implementation, and Evaluation of mpiBLAST," San Jose, CA, 2003.

[192] C. Wang and E. J. Lefkowitz, "SS-Wrapper: a package of wrapper applications for similarity searches on Linux clusters," *BMC Bioinformatics*, 5(171), 2004,

[193] C. Dwan, "Bioinformatics Benchmarks on the Dual Core Intel Xeon Processor," The BioTeam, Inc., Cambridge, MA 2006.

[194] SURAgrid, "SURAgrid," February, 2008, Available at http://www1.sura.org/3000/SURAgrid.html, Retrieved: April 3, 2008.

[195] E. Elmroth, J. Tordsson, T. Fahringer, F. Nadeem, R. Gruber, and V. Keller, "Three Complementary Performance Prediction Methods For Grid Applications," in *CoreGRID Integration Workshop 2008*, Heraklion, Greece, 2008.

[196] P. Olofsson, *Probability, Statistics, and Stochastic Processes*, 1st ed.: Wiley-Interscience, 2005.

[197] M. Siddiqui, A. Villazón, and T. Fahringer, "Grid allocation and reservation - Grid capacity planning with negotiation-based advance reservation for optimized QoS," in *2006 ACM/IEEE Conference on Supercomputing*, Tampa, FL, 2006, pp. 21-35.

[198] W. Pfeiffer and N. J. Wright, "Modeling and Predicting Application Performance on Parallel Computers Using HPC Challenge Benchmarks," in *IEEE Symposium on Parallel and Distributed Processing (IPDPS) 2008*, Miami, FL, 2008, pp. 1-12.

[199] R. Duan, R. Prodan, and T. Fahringer, "Run-time Optimization for Grid Workflow Applications," in *7th IEEE/ACM International Conference on Grid Computing*, Barcelona, Spain, 2006.

[200] C. Baru, R. Moore, A. Rajasekar, and M. Wan, "The SDSC Storage Resource Broker," in *CASCON '98*, Toronto, Canada, 1998.

[201] J. Novotny, M. Russell, and O. Wehrens, "GridSphere: A Portal Framework for Building Collaborations," *Concurrency and Computation: Practice & Experience*, 16(5), April 2004, pp. 503-513.

[202] S. McConnell, *Rapid Development* 1st ed. Redmond, WA: Microsoft Press, 1996.

[203] G. v. Laszewski, I. Foster, J. Gawor, and P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, 13(8-9), 2001, pp. 643-662.

[204] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. v. Nieuwpoort, A. Reinefeld, F. Schintke, T. Schutt, E. Seidel, and B. Ullmer., "The grid application toolkit: toward generic and easy application programming interfaces for the grid," *IEEE*, 93(3), March 2005, pp. 534-550.

[205] B. Sotomayor and L. Childers, *Globus Toolkit: Programming Java Services*, 1st ed.: Morgan Kaufmann, 2005.

[206] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Global Grid Forum, Open Grid Service Infrastructure Working Group June 22 2002.

[207] L. Wall, T. Christiansen, and J. Orwant, *Programming Perl*, 3rd ed., vol. 2000: O'Reilly Media, 2000.

[208] B. Momjian, *PostgreSQL: Introduction and Concepts*, 1st ed.: Pearson Education, 2000.

[209] L. Cannon, "A Cellular Computer to Implement the Kalman Filter Algorithm," Bozeman, MN: Montana State University, 1969.

[210] NCBI, "FASTA format description," 2009, Available at http://www.ncbi.nlm.nih.gov/blast/fasta.shtml, Retrieved: March 19, 2009.

APPENDIX A

GRID USER CATEGORY CLASSIFICATION

This appendix provides an overview of the roles and goals of individual user categories that comprise the grid user environment. Such classification helps in identifying the specific requirements that are imposed by each user category, as well as the tools that accommodate their needs. The categories of users introduced in this section are middleware developer, application deployer, application developer, resource owner, and end user. In some cases, a single person may play multiple roles (*e.g.*, application developer and deployer).

A.1 Middleware Developer

As the interconnecting fiber of the grid, middleware represents the component that facilitates the management of underlying resources as well as development of grid applications. Grid middleware is capable of delivering and sharing compute and data resources over secure channels while hiding intrinsic platform differences among the available resources. The middleware developers are key participants in the grid ecosystem and are closely connected to all other user categories. Middleware developers consider feature requests from user groups and participate in the design and implementation of the middleware standards that shape the way the grid operates. Examples of grid middleware include: the Globus Toolkit, Message Passing Interface (MPI) [54], Condor [89], Storage Resource Broker (SRB) [200], web portals (GridSphere [201], and the Open Grid Computing Environment (OGCE) [50].

A.2 Application Deployer

An application deployer has the responsibility of performing the necessary work to grid-enable a legacy application. Because of the well-known possibility of high costs and challenges of application modification, the community focus is not on modifying the

source code, but rather adopting the application through creation of wrappers that convert the entire application into a grid-enabled solution [154, 202]. The deployment process and wrapper creation for one of the five strategies is performed using a combination of the following techniques:

- Create script interfaces for command-line tools
- Program directly to the Globus Toolkit using an available C API
- Use a commodity toolkit such as Java CoG kit [203]
- Use the Grid Application Toolkit (GAT) [204]
- Use Grid Services [205]

A.3 Application Developer

As the grid gains acceptance, more applications will be custom tailored to take advantage of the benefits offered by the grid. Application developers focus on composing new applications that are targeted for the upcoming platform. Because grid computing does not provide a novel programming paradigm (as described in Introduction Chapter), traditional programming techniques and methodologies (*i.e.*, sequential or parallel) have to be employed. However, because of such non-native adoption, those technologies have their own set of challenges that need to be dealt with and, incorporating the grid into a legacy application, poses a new set of accidental complexities. Most of the programming techniques and application types available when developing grid applications come from the HPC area or traditional distributed computing.

Grid services, which are based on the Service-Oriented Architecture (SOA) [206] and Simple API for Grid Applications (SAGA) [49], represent the future of grid application development.

A.4 Resource Owner

Resource owners and providers assist in running the most basic layer of the grid multi-tier architecture, which consists of the resources (*e.g.,* networks, computer, storage nodes), the middle-tier (*e.g.,* middleware, certificate authority, portals), and the clients (*e.g.,* GT clients, GUIs). Resource owners are responsible for maintaining their respective resources by meeting hardware requirements for individual applications, installing those applications, obtaining relevant libraries and compilers, maintaining applications, and enforcing policies. Beyond the initial cost of installing the grid environment, there is the natural cost of running individual resources, as well as keeping licenses, subscriptions, and libraries up to date. The resource owners must also be cognizant of the QoS agreements between participating users (*e.g.,* compatibility, application versioning, standardization). In order for grid computing to move further into the mainstream, resource providers must be able to balance their operating costs with the resource utilization and find a benefit in sharing their resources with the wider community.

A.5 End user

The end users rely upon the results of a grid application. End users represent the top-most layer of the grid environments and are interested in utilizing underlying capabilities. In order for the grid to become a commodity computing technology, there is a need to attract a wide variety of users. Some of the issues preventing widespread adoption are the complexities and dynamics involved in job submission and job management, which are not yet tailored for lay-persons (*i.e.,* those who are not computer savvy). The following scenario exemplifies the manual steps needed to submit a job on the grid:

1. Select the resources needed by an application

2. Create a proxy certificate (*e.g.*,, grid-proxy-init)
3. Copy the necessary source code and input files to a remote host (*e.g.*, globus-url-copy)
4. Create a Resource Specification Language (RSL) [41] string
5. Submit the job for execution (*e.g.*, globus-job-run, globus-job-submit)
6. Copy output files generated from the remote host to local machine (*e.g.*, globus-url-copy)

Although the general concept of the above steps exists in many distributed contexts, the introduction of the grid manifests as an additional obstacle toward application usage. As such, the grid requires a level of computer expertise that is not within the skill sets of general end users. Therefore, one common goal is to simplify job submission by abstracting grid resources from the end user.

APPENDIX B

APPLICATION SPECIFICATION LANGUAGE

This appendix provides additional details and the overall schema of the Application Specification Language (ASL) [173] presented in Section ##. As a new approach toward application specification that focuses on the needs of grid users and grid applications, the ASL is able to capture essential application information. Through standardized protocols, tools can be passed the information about an application that is specified in an ASL description. Summarizing, ASL is a language for describing any application's requirements, attributes, and options. The ASL directly supports the ability to capture application-specific information that is not necessarily found in the general pool of available description tags. Using ASL, factors such as software and hardware requirements, data constraints, and algorithm complexity can be provided to a user. As can be seen in Figure 57, the ASL may be composed with other groups of established grid languages (*e.g.*, JSDL/RSL, RDL). The interactions implied in the triangle connect all perspectives and user categories of a grid environment, which enable communication to take place over well-designed paths to facilitate further communication, refinement, contract creation and the possibility of higher QoS for all participants.

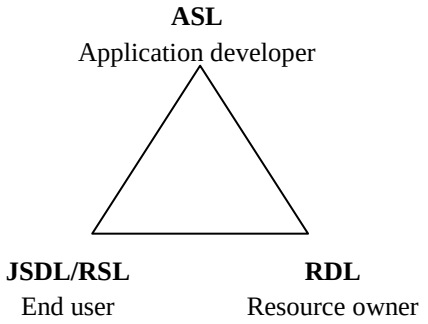


Figure 57. ASL-RDL-JSDL/RSL triangle showing direct communication paths between corresponding user categories

ASL is applicable before and during installation, during job scheduling, during job execution, and even after the job has completed. It can be complemented and modified as knowledge about an application increases. The ASL can be used with legacy applications (requiring adaptation), or with newly developed applications designed specifically for the grid (often called ‘Smart Applications’). By providing a standardized way to describe application requirements, the ASL enables an automated capability to compare applications. Without ASL, such comparisons are hard to perform manually because of their subjectivity. Such comparisons can be useful in numerous cases, such as application scheduling and software cost estimation [154].

In essence, ASL is an extended application version of RSL. It provides a set of specialized tags used to capture application-specific details and thus provide a description of an application. The goal of this language is to use *application-specific descriptions* that enable deployment, maintenance, and execution of an application in a standardized and simplified way. The structure of the language is intended to describe entire, operational applications rather than individual components of an application or other software which may subsequently be composed into an application. The intent of ASL and individual ASL documents is to enable needed communication between heterogeneous resources in the grid through standard interfaces. Just like ASL’s sister languages JSDL and RSL, ASL is not a grammar-based language, but rather a specification language establishing and defining a standard interface needed for heterogeneous grid resources to communicate with each other. A grammar-based language refers to a language that is defined and constrained by a context-free grammar. A schema-based language is rooted in an XML schema and is constrained by the tags defined in the associated schema.

By providing the appropriate set of tags, ASL enables application comparison and interface generation. Because every grid application is custom built to meet a certain need, the implementation details may be difficult to describe. Many of the options available during application specification often require significant human intervention as well as use of human language descriptions that cannot be modeled and captured by a general purpose computer language (*e.g.*, Java or C++). Providing a standardized set of tags to capture information about an application in a concise and precise manner is difficult. The requirements imposed when selecting a given set of tags must focus on capturing the core set of characteristics describing any application and then provide an extended set of tags that allow unique application components to be specified. Our first attempt at defining this set of tags considered existing languages such as JSDL and RSL, which capture job submission requirements that map onto resource and application requirements. Examples of such tags include numerical values (*e.g.*, CPU speed and amount of main memory required), as well as a predefined set of values (*e.g.*, operating system and CPU architecture type).

Additional tags were created by a systematic analysis of characteristics that describe an application but were not required for application invocation (*e.g.*, max number of CPUs an application scales to). Many of these tags are simple in nature allowing the definition of a range of valid values that can be used to validate data entered by the end user. The more difficult set of requirements deals with values that are dependent on each other, but can be viewed individually as containers of simple values. Thus, these tags were organized in groups where sub-elements define individual pieces of the larger component. An example is the operating system requirement. An application may be

suitable for many operating systems as well as different versions of an individual operating system. Thus, creating higher level elements that contain equivalent sub-elements allows different version dependencies to be specified (please see Figure 58).

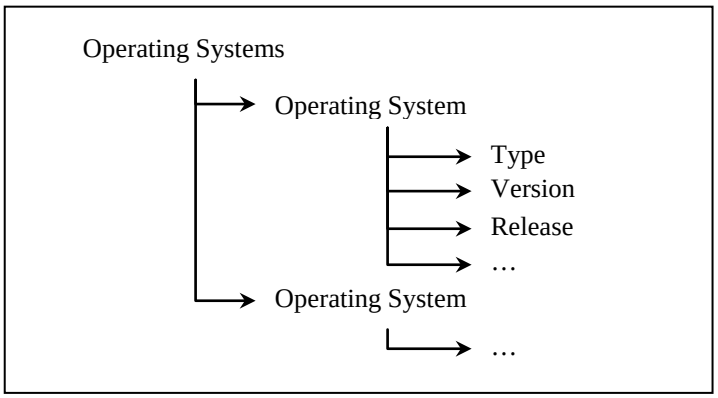


Figure 58. Grouped set of elements and sub-elements with appropriate tags

One concern with adopting ASL, especially when viewed from the perspective of a developer creating the ASL document, is the requirement of the document syntax to be specified correctly (*i.e.*, equivalent tags may have a different meaning when placed in different element groups). The most difficult part of describing an application concisely occurs with tags that cannot be constrained to a set of predefined values (*e.g.*, tags that represent a human readable text string, such as copyright policy). The obvious impediment with such tags is the lack of precision needed for formal interpretation. However, the additional information provided within these tags can benefit end user understandability of the application. The use of the tags for all types of descriptors (*e.g.,* simple, complex groups and natural language) helps to partition the entire document and provide guided help for natural language descriptors. A further benefit of such tags is the possibility of developing additional translators to generate application web documentation automatically.

The completed ASL document consists of several parts (discussed next) each focusing on a particular portion of an application deployment lifecycle. With the blend of the formal tags (*i.e.,* computer readable) and informal tags (*i.e.,* end user understandable), the ASL assists in application description from different perspectives and provides user support in multiple formats. Examples of interpretation include an application description web page with installation and invocation instructions, script generation for automated application installation, as well as optimal system requirements for job submission.

B.1 Structure of the Language

An ASL document consists of four distinct yet related sections that are described in XML. By dividing the document into these distinct sections, an ASL document is more modular and allows for easier initial generation and later modification. With the use of appropriate tools, each section of the document can have its own permissions, which allows the document to be modified independently and securely. As the application receives a wider user base, additional information may become available from its users. Because of multiple executions of an application, additional information can be gathered, such as profiles of application performance, unexpected behavior, or suggestions for future enhancements. Beyond the collection of application information, the segregation of the document into appropriate sections allows for shorter search times among users allowing them to focus on sections of the document of most interest. To provide segregation of information collection and retrieval, an ASL document consists of application name and description, installation requirements, job invocation requirements, and hints.

The sections are not directly connected to each other, but the data is stored only once per ASL document. Because of this, inadvertent references to information provided in other parts of the document may exist. These sections are correspondingly mapped to XML with appropriate tags. Each of the sections is described below and the schema is provided for the given section.

B.1.1 Application name and description

The application name and description section contains the most basic information about an application and acts as the application identification component. It specifies the name and version of the application as can be found in an application repository. This section also contains sub-elements such as the application description describing the application in a human readable format. The description identifies the problem the application solves and maps the application to an application category. The application category element is limited to a predefined set of values as described in Section ##. It is intended to offer better understanding of the application deployment process on the grid and it is essential in classifying different types of applications deployed in a grid environment.

When selecting the application category for an ASL specification, the following considerations should be examined by a user composing the document. Applications that belong to categories (1) and (2) can be distributed and scheduled across any available computational resource on the grid because there is no synchronization or coordination required between individual tasks. However, applications in categories (3) and (4) must be scheduled on a single computational resource and cannot be distributed across multiple resources. This does not imply that the application may not use additional,

distributed resources. If an application has been developed specifically for the grid, it can utilize middleware components to enable cross-resource task execution. In that case, task scheduling is the responsibility of the application itself because it would be deployed on a dedicated resource. Applications in categories (5) and (6) expect that the individual applications are distributed and assume that the individual tasks are scheduled to execute at the same time (through advanced reservation or mutual agreement with the resource providers). Workflow applications (category 7) assume that the scheduler can trigger the execution of one or more applications as described by the workflow. Because most of the existing schedulers [85, 87, 97, 111] do not handle advance reservation, the use of workflow applications is limited and intended mostly for future generations of applications and grid schedulers.

The remaining elements in this section are illustrated in Figure 59. *Category* element set has a predefined set of values from which a user must choose. The remaining elements found in this schema section do not have their values predefined, but can be defined by the person creating the document enabling desired application description.

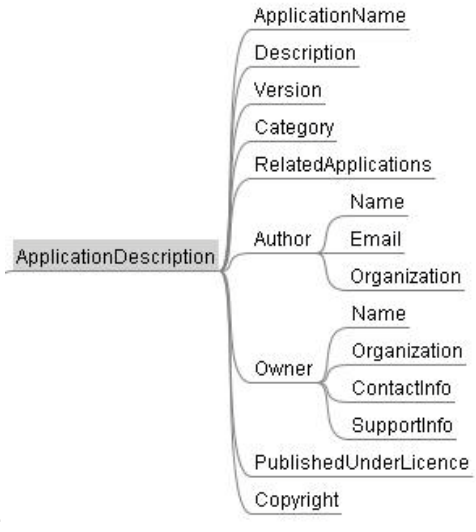


Figure 59. Application Description section schéma

B.1.2 Installation requirements

The installation requirements section of an ASL document contains a set of required elements that describe the installation requirements and the installation procedure. Some of the examples of this type of element set include minimum processor speed, processor architecture, minimum amount of memory needed for installation of the application, libraries, applications required for the installation procedure (*e.g.*, compilers, (un)packaging tools), licenses needed for application installation, network requirements, and required amount of disk space for the installation. The tags used are simple declarations that specify the value of a predefined type (*e.g.*, string, integer). Even though this model may result in unnecessary inconsistencies between application descriptions, we believe at this stage of ASL development and definition this variability is necessary to allow the correct words to be selected from a constrained set of choices. The full schema of the installation requirements section is given in Figure 60.

Among the elements defined in the installation category are *SoftwareDependencies* and *Applications Required* tags. The information these tags contain is intended strictly to be used during the installation procedure. The *SoftwareDependencies* tag refers to any other software that will be needed for the application execution. This can be viewed as a prerequisite for the installation; *i.e.*, in case software packages declared within this tag are not installed, the application cannot be expected to execute. Examples of such software dependencies would include Perl [207] with certain libraries and Postgres database [208]. With respect to installation, the *ApplicationsRequired* tag refers to other complete applications required to perform the installation. These applications may be invoked

during the installation procedure, such as unpackaging and installation tools (*e.g.*, Ant, make).

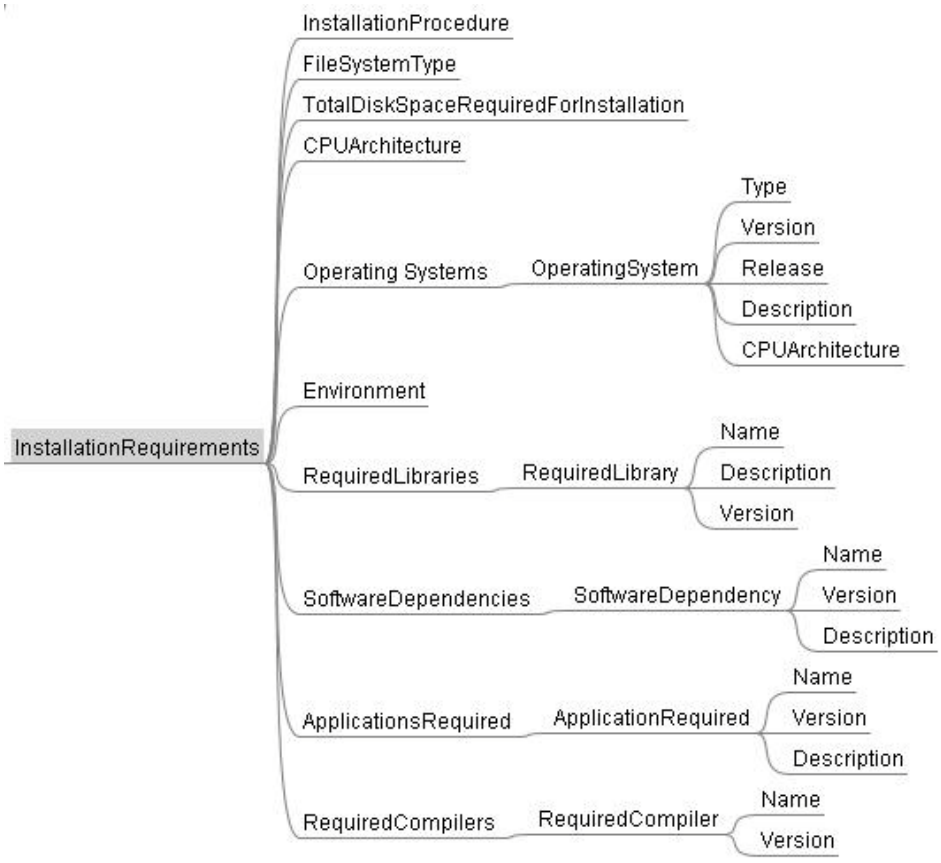


Figure 60. Application Installation section schéma

B.1.3 Job invocation requirements

The job invocation requirements section focuses on providing a user with the information needed to execute the application. Starting with the executable name, it also provides the available switches and minimum hardware requirements, as well as allows the developer to specify the number of input and output files with examples of their respective formats. This section does not represent a duplication of effort found in JSDL/RSL, but it is alternatively used to specify requirements for the entire application. Such specification is needed not only when executing a single job, but to describe the

available options and how to use them. Rather than specifying exact input files and other job-specific parameters, the category defines application requirements, such as: the required input files, required format of those files, any output files and corresponding format of the output files, libraries required to invoke the application, and licenses needed to run the application. This capability can be viewed as a more detailed version of *man* pages in UNIX. This category allows the developer to be shielded through a contract-like document; *i.e.*, if any of the requirements are not met, the application cannot be expected to execute correctly.

The majority of the application description is provided in this section of the ASL document, so it is natural for a set of tools to be based on this category. An example tool is a translator that formats the appropriate information into a web page allowing the information to be read through a browser, or a correct and application-specific job submission interface. Another example tool serves as a data verification tool that ensures input files are in the correct format. The complete schema is shown in Figure 61. Similar to the installation section, the application invocation section has elements *SoftwareDependencies* and *RequiresApplications*. In this context, software dependencies refer to any software packages that are necessary and will be used as part of the application during its execution. An example would include a call to a Perl module. The description of the application requirements tag is similar to the description from the application installation section of ASL, where it specifies any other applications that may be invoked during this application's execution. This tag can be used to specify the requirements for a workflow, even though any further enforcement and coordination during execution must be done within the given application.

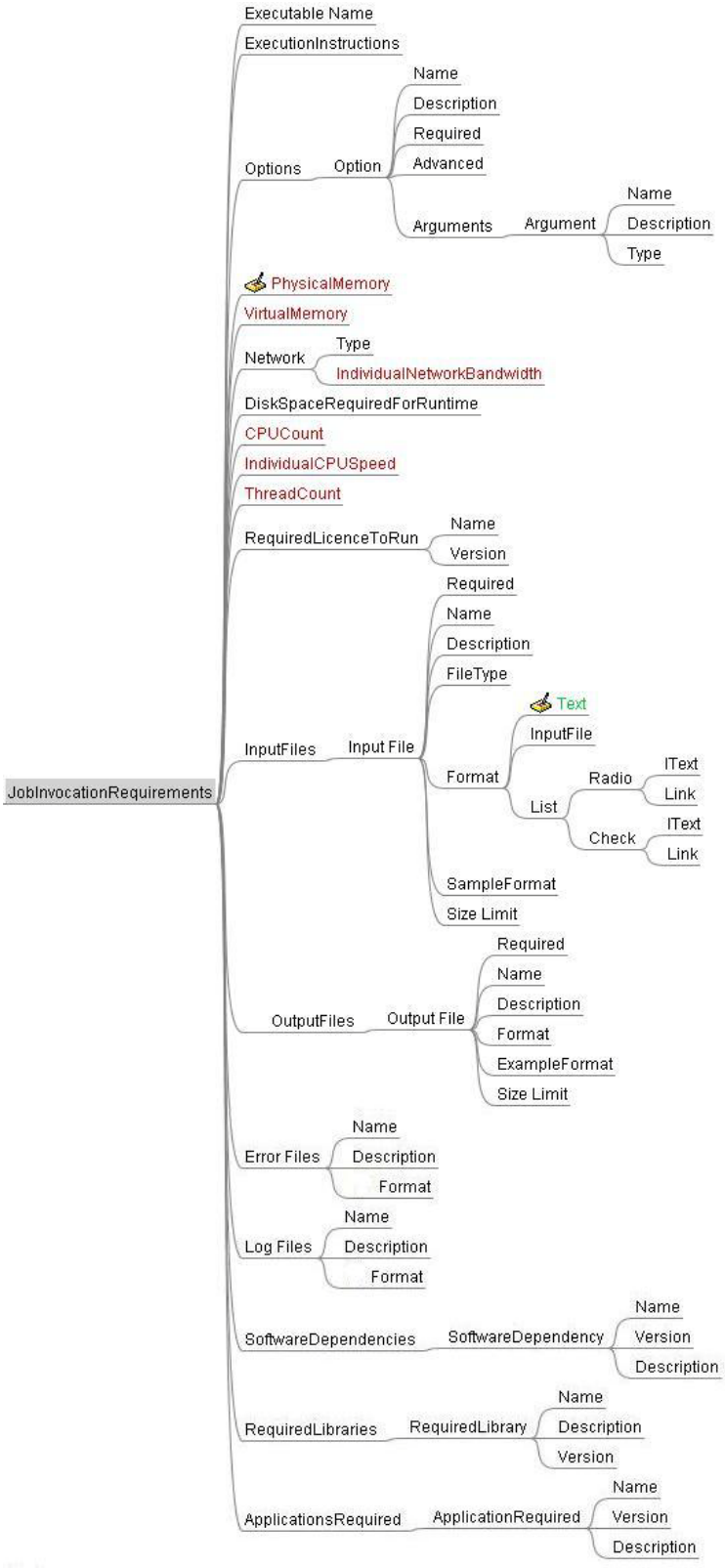


Figure 61. Application Invocation section schema

B.1.4 Hints

The inherent variability of applications results in information describing an application not to be adequately captured in the preceding sections of ASL because of non-compliance and uniqueness of the application. Also, the succinctness of available options in ASL tags or already existing data may prevent additional and possibly more complete application information to be stored. In order to accommodate for these possible shortcomings, there is an additional section found in every ASL document, which is entitled *Hints*. This section contains instructions and comments, mostly in natural language, which provide additional information about the application. The purpose of this information is to allow detailed descriptions for areas of high application complexity, either for the users of the applications or other developers who may use this application as a base layer for development. Another important goal behind this section and its element set is that it can be accessed and edited by a wide user group. Performance information may be stored in this section to specify the optimal parameters on a particular hardware architecture.

Depending on application type (*e.g.*, sequential, embarrassingly parallel, MPI-based application), certain input parameters (*e.g.*, size of input file, input file format, number of processors) may alter application performance and thus information found here could be useful for the resource owner, end user and even the scheduler developer. By giving permissions to a wide range of users, known bugs as well as suggestions for future advancements can be documented. A large portion of its use can be found in

troubleshooting an application where expected errors can be explained. Figure 62 depicts the Hints section schema.

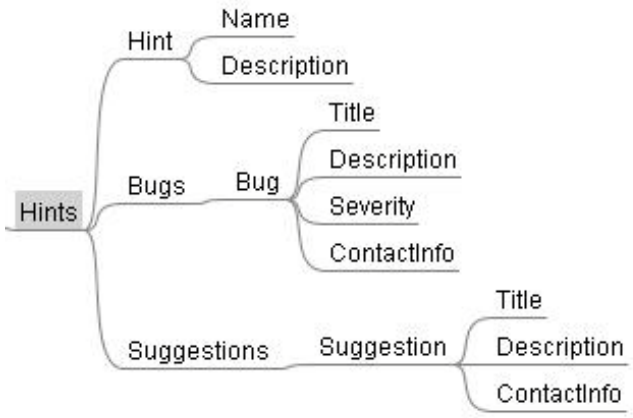


Figure 62. Hints section schéma

B.2 Example ASL Documents

This section demonstrates the use of ASL to describe an initial set of applications that show the ability of the language to specify and distinguish applications. In these examples, ASL was manually generated by providing the values associated with appropriate tags as defined in the schema. This generation is quite straightforward, which provides the user variability in selecting the tags to be defined depending on the application. The rest of this section provides snippets of ASL documents with their respective applications. Differences between applications are outlined and the ability of ASL to capture these differences is discussed.

Application Descriptions

To show the ability of ASL to capture descriptions of applications belonging to different categories, three applications are specified, each from a different application category. The first two applications correspond to the sample scenario described in the *Grid Application Deployment Scenario* section. The first application is a sequential

application implemented in Perl called *QuerySplit* that performs segmentation of the BLAST input query file (please see Figure 64). The *QuerySplit* application takes a text file as a parameter, which contains query sequences of variable length. It then proceeds to analyze the file and create several smaller files, each containing a number of queries so that the overall size of all the files is as close to each other as possible. This application is used by the second application called Dynamic BLAST – a master-worker type application [154] (please see Figure 65 and Figure 66). Dynamic BLAST is a custom-built application intended to maximize use of small, distributed, readily available resources found in the grid to minimize time needed to perform BLAST searches [15]. It uses established grid protocols for communication and task coordination while the custom scheduler handles task allocation and data transfers. The final application for this appendix is a parallel implementation of matrix-matrix multiplication using Cannon's algorithm [209] (please see Figure 67 and Figure 68). Figure 63 illustrates relationships between application categories, applications, and corresponding ASL documents.

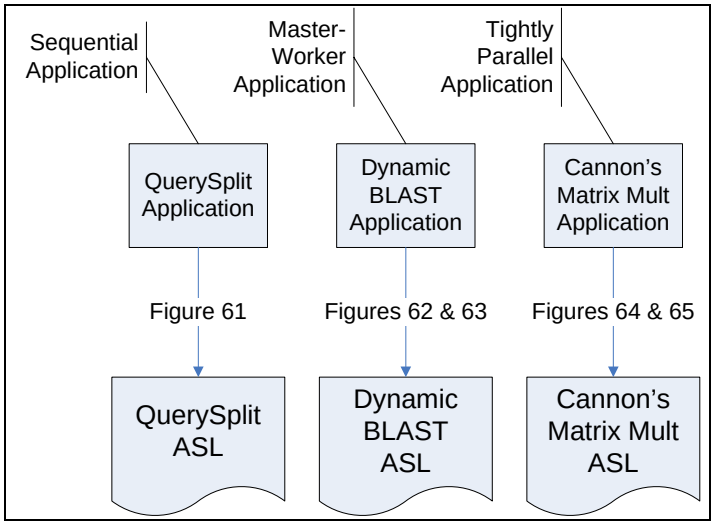


Figure 63. Relationship between application categories, applications and shown ASL documents

ASL is a schema and tag driven language enabling a new type of communication between heterogeneous grid resources. As indicated earlier by Figure 57, ASL enables capturing of application developer specific information and subsequent communication between other existing languages. In this context, the information that needs to be communicated is the information that can be compared to information already existing in other, currently available languages. In the grid environment, the goal of such information is to mitigate inherent heterogeneity of underlying resources, and thus the information needs to be capable of representing individual components of communicating systems that are the cause of this heterogeneity.

B.2.1 ASL Document Snippets

This section provides selected snippets with essential parts of the application descriptions outlining capabilities of ASL to capture application information and how dependencies can be drawn between applications, software packages and required libraries.

Figure 64 is an example of a complete ASL *JobInvocationSection* section for the *QuerySplit* application described in the previous section. This description starts off by capturing all the basic application run-time information (lines 2 through 9), such as the application invocation method, the minimum amount of memory required for the application to run, as well as the minimum speed of a host CPU. The most significant part of the provided example is to show how to describe an input file for an application. Lines 10 through 38 point out that the given application requires one input file to be provided in plain text (*i.e.*, ASCII) and should be formatted according to the comments available in lines 19 through 37. In particular, lines 20 and 21 indicate that the properties input file

```
1. <asl:JobInvocationRequirements>
2. <asl:ExecutableName>QuerySplit</asl:ExecutableName>
3. <asl:ExecutionInstructions>Usage: perl QuerySplit propertieFile
   </asl:ExecutionInstructions>
4. <asl:PhysicalMemory>
5. <asl:LowerBoundedRange>5.0</asl:LowerBoundedRange>
6. </asl:PhysicalMemory>
7. <asl:IndividualCPUSpeed>
8. <asl:LowerBoundedRange>200.0</asl:LowerBoundedRange>
9. </asl:IndividualCPUSpeed>
10.   <asl:InputFiles>
11.     <asl:NumberOfInputFiles>1</asl:NumberOfInputFiles>
12.     <asl:InputFile>
13.       <asl:Name>PropertiesFile</asl:Name>
14.       <asl:Required>true</asl:Required>
15.       <asl:Description>Job properties file whose format is
16.        described below.
17.    </asl:Description>
18.       <asl:FileType>ASCII</asl:FileType>
19.       <asl:Format>
20.         <asl:Text required="1">userName</asl:Text>
21.         <asl:Text required="1">jobName</asl:Text>
22.         <asl:InputFile>
23.              <asl:Name>queryInputFileName</asl:Name>
24.              <asl:Required>true</asl:Required>
25.              <asl:Description>This input file is submitted as
26.               part of the properties
27.               input file. A query input file in FASTA format
28.               containing at least one
29.               query.</asl:Description>
30.              <asl:FileType>FASTA</asl:FileType>
31.              <asl:Format>
32.               >gi|5524211|gb|AAD44166.1| cytochrome b (Elephas
33.               maximus maximus)
34.               LCLYTHIGRNIYYGSYLYSETWNTGIMLLLITMATAFMGYVLPWGQ
35.               MSFWGATVITNLFSAIPYIGTNLVEWIWGGFSVDKATLNRFFAFHFIL
36.               PFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFHPYYTIKDFLGLL
37.             </asl:Format>
38.        </asl:InputFile>
39.        <asl:Text required="0">totalNumFragments</asl:Text>
40.       </asl:Format>
41.   <asl:SizeLimit>
42.       <asl:UpperBound>0.1</asl:UpperBound>
43.   </asl:SizeLimit>
44.   </asl:InputFile>
45.   </asl:InputFiles>
46.   </asl:JobInvocationRequirements>
```

Figure 64. ASL document snippet showing Job Invocation Section for QuerySplit application.

must contain two plain text fields with user name and job name, respectively, followed by a pointer to another input file. The format of the contained input file is described as being

in FASTA format [210] whose sample is also provided in the *format* tag. The tags used for capturing necessary information (*e.g.*, text) map favorably to individual components of a job submission interface (*e.g.*, HTML). By analyzing the available tags and provided information, enough details are available to fully automate creation of a job submission interface by a higher level tool with items such as text boxes, radio buttons and drop down menus with entries predefined in the ASL document. Through this approach, an application-specific job submission interface can be generated directly from the application description, thus properly directed by an application developer rather than an application deployer.

```
1. …
2. <asl:InputFiles>
3. <asl:NumberOfInputFiles>1</asl:NumberOfInputFiles>
4.     <asl:InputFile>
5.         <asl:Name>PropertiesFile</asl:Name>
6.         …
7.         <asl:List>
8.             <asl:Radio>
9.                 <asl:lText>blastp</asl:lText>
10.                <asl:lText>blastn</asl:lText>
11.                <asl:lText>blastx</asl:lText>
12.                <asl:lText>tblastn</asl:lText>
13.                <asl:lText>tblastx</asl:lText>
14.                <asl:lText>psiblastpn</asl:lText>
15.             </asl:Radio>
16.          </asl:List>
17.          …
18.      </asl:InputFile>
19.      ...
20.  </asl:InputFiles>
```

Figure 65. Job Invocation Section of an ASL document for Dynamic BLAST application showing input file options.

Figure 65 provides an example of how an application developer can provide all the available values for a selected application invocation option, even within an input file. The XML snippet describes an application's single input file argument options. These are listed in lines 8 through 15, which limit the user's choice to any single value when

invoking the application. This is an additional example of how ASL information can be applied in dual context for automated interface generation denoting that this information belongs to a single list and can be organized into a radio button group.

```
1. <asl:ApplicationsRequired>
2.     <asl:ApplicationRequired>
3.          <asl:Name>GridWay</asl:Name>
4.          <asl:Version>5.0+</asl:Version>
5.     </asl:ApplicationRequired>
6.     <asl:ApplicationRequired>
7.         <asl:Name>QuerySplit</asl:Name>
8.         <asl:Version>1.0</asl:Version>
9.     </asl:ApplicationRequired>
10. </asl:ApplicationsRequired>
11. <asl:RequiredLibraries>
12.     <asl:RequiredLibrary>
13.         <asl:Name>org.ggf.drmaa.*</asl:Name>
14.     </asl:RequiredLibrary>
15. </asl:RequiredLibraries>
16. <asl:SoftwareDependencies>
17.     <asl:SoftwareDependency>
18.         <asl:Name>GlobusToolkit</asl:Name>
19.         <asl:Version>4.0.2+</asl:Version>
20.     </asl:SoftwareDependency>
21.     <asl:SoftwareDependency>
22.         <asl:Name>java</asl:Name>
23.         <asl:Version>1.5+</asl:Version>
24.     </asl:SoftwareDependency>
25. </asl:SoftwareDependencies>
```

Figure 66. Job Invocation Section of an ASL document for Dynamic BLAST application showing application, library, and software dependencies

Figure 66 is a continuation of the ASL document for a Dynamic BLAST application. This snippet highlights the application, library, and software dependencies that Dynamic BLAST depends on or requires. As can be seen from the XML, a Dynamic BLAST application depends on being able to correctly invoke two other applications, namely GridWay [118] and QuerySplit, whose description was provided in Figure 64. This idea, although simple to comprehend, has a significant potential in terms of application dependency visualization and installation tools. Through the use of this formalized method of declaring direct dependencies, much automation can be achieved. The

remainder of the application description, shown in lines 11 through 15, indicates that Dynamic BLAST requires a specified library. Finally, lines 16 through 25 denote other software packages required to run Dynamic BLAST.

```
1. <asl:ExecutableName>pmatmul</asl:ExecutableName>
2. <asl:ExecutionInstructions>Usage: mpirun -np [numCPUs]
3. pmatmul [N] [P] [Q] [flag]
4. </asl:ExecutionInstructions>
5. <asl:Options>
6.     <asl:Option>
7.         <asl:Arguments>
8.             <asl:Argument>N</asl:Argument>
9.                 <asl:Description>Matrix size. Works for
10.                square matrices only.</asl:Description>
11.         </asl:Arguments>
12.         <asl:Required>true</asl:Required>
13.     </asl:Option>
14.     <asl:Option>
15.     …
16.     </asl:Option>
17. </asl:Options>
```

Figure 67. Job Invocation Section of an ASL document for parallel matrix-matrix multiplication application showing and describing application invocation options and arguments

Figure 67 also depicts the job invocation section of a parallel matrix-matrix multiplication. Lines 2 and 3 specify the format of the invocation command with several options. Lines 7 through 11 point out the necessary details about the first argument (*e.g.*, argument description, whether it is required or optional). The remainder of the argument descriptions are omitted for brevity, but the information provided shows the ability of ASL to structurally and formally capture such information while allowing the user enough freedom to describe each of the arguments at the desired level of detail.

```
1. <asl:PhysicalMemory>
2.     <asl:Formula>5*N^2/sqrt(P)</asl:Formula>
3. </asl:PhysicalMemory>
4. <asl:Network>
5.     <asl:Type>single</asl:Type>
6. </asl:Network>
7. <asl:CPUCount>
8.     <asl:LowerBoundedRange>2.0</asl:LowerBoundedRange>
9. </asl:CPUCount>
```

Figure 68. Job Invocation Section of an ASL document for parallel matrix-matrix multiplication application showing ability of ASL to capture memory and network requirements

Finally, Figure 68 points out two more interesting points supported by ASL. Lines 4 through 6 deal with network requirements. Because this application is an example of a tightly coupled parallel application, the communication patterns are frequent throughout the algorithm iterations and thus the message passing requires the existence of a fast-speed, local network. Although this requirement can be built into an ASL definition and made a default requirement for all applications of this type, advances in message-passing technologies are enabling the communication to take place across administrative domains (*e.g.*, MPICH-G2 [55]), which would make this a possible hindrance to future applications. To avoid this limitation, the network type tag accepts ‘single’ as its value denoting this application can be executed only on a local network, limited to a single resource. There are other possibilities here, but these options would all require relationships to be made between parts of an ASL document. The final interesting feature found in this part of the sample ASL document is the use of formulas as part of the application description (line 2). This information can be used by a scheduler when the input data is already known to perform not only application-specific but also data-specific scheduling.

APPENDIX C

TECHNICAL RESOURCE DETAILS

The following table contains technical details about compute resources utilized during the work presented throughout this document.

Table 8. Technical details of resources used during performed experiments.

Resource Name	CPU Type	CPU Clock Frequency (GHz)	CPU Instructions per Cycle	Number of Cores per Node	Memory per Node (GB)	Scheduler	Operating System
Ferrum	Intel Xeon E5345	2.3	4	8	12	SGE 6.0	Linux 2.6.9
Olympus	Intel Xeon	3.2	2	2	4	SGE 5.3	Linux 2.4.21
Everest	AMD Opteron 265	1.6	2	2	2	SGE 6.0	Linux 2.6.9
Cheaha 1	AMD Opteron 265	1.6	2	2	2	SGE 6.0	Linux 2.6.9
Cheaha 2	Intel Xeon E5450	3.0	4	8	16	SGE 6.0	Linux 2.6.18
Coosa	Intel Xeon	3.6	2	2	2	SGE 6.0	Linux 2.6.9
Wave	Intel Pentium D	3	2	2	1	Fork	Linux 2.6.18
iBook	Intel Core 2 Duo	2.33	2	2	2	Fork	Mac OS X
Dual Opteron	AMD Opteron 265	1.8	2	4	8	Fork	Linux 2.6.9