The Mini-Grid Framework: Application Programming Support for Ad hoc Volunteer Grids

Neelanarayanan Venkataraman

Software Development Group
IT University of Copenhagen
Rued Langgaards Vej 7, DK-2300, Copenhagen S

November 5, 2012

Abstract

To harvest idle, unused computational resources in networked environments, researchers have proposed different architectures for desktop grid infrastructure. However, most of the existing research work focus on centralized approach. In this thesis, we present the development and deployment of one such infrastructure, called the *Mini-Grid Framework* for resource management in ad hoc grids using market-based scheduling and context-based resource and application modeling. The framework proposes peer-to-peer architecture that supports several futures: ease of deployment, decentralized task distribution, small scale ad hoc grid formation, and symmetric resource. Furthermore, users can model and specify non-performance based parameters that influence resource allocation.

We evaluated the framework through simulation experiments at the IT University of Copenhagen (ITU), as well as a pilot deployment at the Interdisciplinary Nanoscience Center (iNano), Aarhus University. For the simulation experiments we used an application that calculates prime numbers between a given range, and another application that searches for a key in a large data set. In the simulation experiments, we studied the technical performance and overhead of the Mini-Grid Framework and compared its performance with other relevant systems. For the pilot deployment, we have integrated a parallelized version of the Basic Local Alignment Search Tool (BLAST) algorithm and the RNA secondary structure prediction algorithm developed by iNano research center with the Mini-Grid Framework. These algorithms have been developed on top of our framework through an integration of the framework with the CLC Bio Workbench, a software suite for bioinformatics algorithms. The pilot deployment studied the resource participation, the deployment efforts needed, and the performance of the framework in a real grid environment.

The main contribution of this thesis are: i) modeling entities such as resources and applications using their context, ii) the context-based auction strategy for dynamic task distribution, iii) scheduling through application specific quality parameters, iv) the definition of an extensible API for ad hoc grid formation and v) enabling symmetric resource participation. The Mini-Grid framework has been designed and developed as proof-of-concept. The Mini-Grid framework has been evaluated using LAB deployment at ITU, and has been deployed at iNano research center using real-life application.

Contents

Declaration

Section 6.6 in Chapter 6 include data from collaborative work, in which I performed major part of the experimental work. In the controlled experiments, the theoritical execution time calculations were performed by Sükösd Z and I have used here for illustration.

Section 8.5 in Chapter 8 include graphical user interface developed by Morten Vaerum. I have used here for illustration.

Acknowledgment

Thanks to my supervisor, Prof. Jakob E. Bardram, for his expert guidance. I would like to thank all my friends and colleagues. Following an alphabetical oder, I would like to thank Andrzej Wasowski, Aurelien Tabard, Joe Kiniry, Kasper sterbye, M. Ali Babar, Peter Sestoft, Rune Mller Jensen, Sofiane Gueddana, Sren Lauesen, Vibeke Erv and Yvonne Dittrich. I would like to thank my friends Afsaneh Doryab, Alberto Delgado-Ortegon, Dario Pacino, Dermot Cochran, Hannes Mehnert, Helge Pfeiffer, Josu Martinez, Kosta Pandazo, Johan Bolmsten, Juan Ramos Hincapie, Kevin Tierney, Mads Frost, Mohammad Amin Kuhail, Paolo Tell, Rosalba Giuffrida, Shangjin Xu and Sren Lippert. Kind regards to all our colleagues from the FIRST research school, including the administrative staff who have always been very helpful. Thanks to our industrial partner, CLC Bio, for their collaboration, especially: Morten Vaerum, Thomas Knudsen and Alex Andersen. Thanks to our research partner, Nanoscience Center at Aarhus University, Ebbe Sloth Andersen, Zsuzsanna Sukosd and Jrgen Kjems. My special thank to Tinus Abell. I would like to thank Adrian Friday, Lancaster University for his support during my stay abroad. I would like to thank the PhD administration, the finance department and the facility management for their support. To the forgotten ones: if your name should have been in the list, please blame my memory, not intentional.

Part I

Introduction

Chapter 1

Introduction

Bioinformatics applications are computationally very intensive and require vast amounts of processing power and memory requirements. For example, finding genes in DNA sequences, predicting the structure and functions of new proteins, clustering proteins into families, aligning similar proteins, and building phylogenetic tree showing the evolutionary relationships are all computationally intensive. Such kind of computational biology problems require design and development of solutions that maximize performance and programmability. Using computational Grid infrastructure is an effective way to tackle the problem. The computational Grid infrastructures are built using a set of processors that are able to cooperate in solving computational problems. This cooperation is achieved by splitting the computational load of the problem into parts, and then combining the partial computations to obtain the result. Thus modeling a bioinformatics application as a Bag-of-Task (BoT) application, where each computational process (task) works at its own rhythm in an asynchronous fashion with complete independence from other computational process, makes it possible to use computational Grid infrastructure to meet their computational demands.

A computational Grid has been defined as a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [Foster and Kesselman, 1999]. The computational Grids explore mechanisms for access to computational resources and quality of service beyond the best effort provided by the Internet protocol (IP). Grid infrastructure supports computation and other collaborative activities that spawn across multiple organizations. This new paradigm called grid computing has resulted in the emergence of distributed computing measures wherein there is development of a mechanism for allowing current collection of connected computing systems to form large scale data as well as computing networks which promote and ensure that distributed resources are shared and distributed in a manner that involved heterogeneous applications ensuring that different application domains which include science, industry, engineering and governmental resources in order to solve large scale computing problems by making use of large

scale high capacity resources or enabling volunteer computing resources Jacob [2003]; Jacob et al. [2003, 2005].

Generally speaking, grid computing platforms can be classified into two main categories: classic high-end grids and volunteer computing grids [Kurdi et al., 2008]. Classical grids provide access to large-scale, intra-and-inter institutional high capacity resources such as clusters or multiprocessors [Foster and Kesselman, 1999; Foster, 2006]. On the other hand, volunteer computing is based on the idea of utilizing the CPU cycles of idle machines [Litzkow et al., 1988; Anderson, 2004]. Classical grid computing platforms deploy the Globus middleware [Foster, 2006] on distributed resources for delivering high-performance computational power to restricted community of users (e.g., TeraGrid - www.teragrid.org) with a specific goal. However, such grids require dedicated hardware and software resources interconnected by high speed reliable network connections and follow a centralized approach – meta-schedulers – for task distribution.

The BOINC infrastructure [Anderson, 2004], the most popular volunteer computing platform, is composed of a central scheduling server and a number of clients installed on the volunteer's machines. The client component periodically contacts the server to report its availability and to request work. However, BOINC requires a fixed and static set of data servers deployed centrally that need to be maintained by each project team. Further, creating a volunteer computing application involves the process of obtaining and retaining volunteers.

The volunteer computing infrastructure is different from the classical Grid computing infrastructure in terms of the type and characteristics of resources, and the type of sharing. These volunteer computing systems when compared to traditional grid computing resources are found to show presence of highly heterogeneous resources which are found to have architectural platforms and varying CPU speeds, disk space, as well as type of software used Iamnitchi and Foster [2000]. Another important aspect is the presence of dynamic resources The resources of volunteer computing consists of mainly personal computers (i.e., desktop), whereas the resources of classical Grid include high capacity supercomputers, clusters, servers, storage device, etc. The resources of volunteer computing are non-dedicated, highly volatile, unreliable, faulty and highly heterogeneous compared to the resources of the classical Grid. The volunteer computing resources are administrated by individual users, whereas the Grid resources are managed by professional system and network administrators. Normally, the volunteer computing applications consists of a set of independent tasks and require no communication between the tasks. However, classical Grid computing applications can include tasks that need to communicate between them.

A major advantage of desktop grids over traditional grid systems is that they are able to utilize non-dedicated machines. Besides, the requirements for providing resources to a desktop grids are very low compared to traditional grid systems using a complex middleware. Even though the barrier may be low for resource providers, deploying a desktop grid server is a more challenging task because a central server creates a central point of failure and a potential

bottleneck, while replicating servers require more effects Marosi et al. [2007]. In volunteer computing grids a number of users are promoted to share their resources to solve a common goal however they are not given the access to make use of these resources to any problems they might have.

Most of the existing approaches to grid computing is, however, very centralized both technically and in use. For example, classical grids using Globus follow a centralized approach for task distribution and the alternative volunteer computing uses variants of a master/slave architecture for task distribution. Furthermore, volunteer computing is highly asymmetric: volunteers supply computing resource but cannot demand computing resource. Practically, individual users cannot use volunteer computing to meet growing needs for computational resource in his routine computer use (i.e., volunteer computing is intended for global computing). Finally, installing, configuring and customizing such solutions require high technical knowledge Thain and Livny [2001]; Foster et al. [2001].

1.1 Problem Description

Let us consider the typical environment of the computer set-up in a biological research-based institution. Usually, Biologists working with laboratory-based and theoretical molecular biological research have usually access to one or two clusters and have to share their computation time with others. At the same time, these biologists work in labs or institutions, which have a large number of desktops and laptops. Those machines are usually under-utilized and are only available to a single user. Organizing such desktop machines as a distributed system for computations or other kinds of resource sharing is highly desirable. For example, a group scientists or researchers can contribute their desktop machines when idle for a common objective. In this example, every participant want to participate for a limited amount time, normally until the participant has some interest in the collaboration. For such a group it may be infeasible to build or participate in a classical grid infrastructure or volunteer computing infrastructure because of administrative overheads, cost, etc. Thus support for formation of "ad hoc" grid infrastructure among participants of one-time or short lived collaboration is required. Furthermore, such an infrastructure is suitable to applications with low communication/computation ratio, such as parallel search algorithms. Ad hoc grid is a distributed system with the aim of harnessing unused computational resources inside an organization. It can be considered as a temporary coalition of resources contributed by individuals to achieve common objective of access to high computational environment. In an ad hoc grid, every resource can spontaneously arise as a resource consumer or a resource producer at any time when it needs a resource or it possesses an idle resource.

However, existing models and infrastructures for desktop grid computing do not support for such ad hoc collaboration among participants. Existing desktop grid computing system follow either the client-server or unstructured

peer-to-peer architecture. Unstructured peer-to-peer approach use centralized structure for connecting edge peers to super-peers. Thus most of the current desktop system have some form of centralization in their architecture. Most of the current desktop grid system focus on asymmetric resource utilization, i.e., volunteers supply capabilities of computational resources but do not consume any. However, in a research collaboration users (i.e., volunteers) not only wish to supply computational resources but also, if required, may wish to consume other's computational resources.

Furthermore, most of the current approaches are limited as they do not support application specific quality of service parameters during resource allocation. Most of the current approaches to resource allocation in classical grid focus on the quality attributes that maximize the performance of the application or resource, for example timeliness of task completion. However, applications also have quality attributes based on security, reliability, location etc. For example, a user may wish to execute his tasks on resources belonging to his group. He can form an ad hoc grid using the available resources in his group. This ad hoc grid need to support the requested quality of service parameter. Scheduling policies such as First-Come-First-Served cannot serve this kind of requirement. Quality of service mechanisms used in high performance grids involve off-line human negotiation between resource providers and resource consumers, which is not suitable for ad hoc peer-to-peer grids. In peer-to-peer grids - without a centralized administrative infrastructure - the implementation of negotiation is difficult. We need to device a mechanism that can model application specific quality of service requirements.

Most existing policies that schedule jobs in desktop grid projects are based on First-Come-First-Served (FCFS) or random assignments. FCFS are very easy to implement in desktop grids. It assigns the first available job instance to the first requesting worker. In random assignment job instances are selected randomly. Such centralized scheduling policies have the disadvantage of poor scalability. Further, such centralized approach lack support for application specific quality of service. Alternatively, distributed computational economy framework for quality-of-service (QoS) driven resource allocation can be used. Such framework provides mechanisms and tools that realize the goal of both resource provider and resource consumer. Resource consumers need a utility model, representing their resource demand and preferences. And resource providers need an expressive resource description mechanism. Ontology based resource description framework can be considered for defining the application requirements and resource capabilities. Specifically, context information surrounding application and resource can be used to model the application requirements and resource capabilities.

Current volunteer grid architectures though support a wide range of applications with diversity related to scope and requirements they fail to support sporadic collaborations in the absence of a central regulating authority which presents the need for an ad hoc architecture which will promote structural independence, technological independence and control independence for more than a single administrator enabling promotion of better user collaboration Amin

et al. [2004]. When limited shared resources serve large number of request using best-effort resource provisioning, the Grid performance degrades. Resource provisioning in any grid infrastructures require mechanisms which take into account the particular user constraints needed to satisfy resource demands. Hence, it is necessary to explicitly, precisely, and unambiguously describe Grid resources and specify various constraints over resource descriptions during resource request.

1.2 Research Questions

As explained in earlier section, current desktop systems, due to centralized architecture do not promote ad hoc grid formation. Such ad hoc grid formation should support ease of deployment, quality of service, and symmetric resource participation. The research questions are:

- How can be ad hoc desktop grid formation supported?
- How we enable symmetric resource usage in a desktop grid environment?
- How can be resource provisioning happen in the absence of a centralized scheduler in the peer-to-peer desktop grids?
- How can a resource provider (or resource consumer) can describe Grid resources explicitly, precisely, and unambiguously during resource provision (or resource request)?
- How can we model entities such as task and resources in a computational Grid environment?
- How can we incorporate user constraints and preferences (i.e., support for application specific quality of service parameters) into resource provisioning?

1.3 Research Methodology

Desktop grids can be used efficiently and conveniently on a smaller scale as well. The easy deployment of desktop grids in small organizations can lead to a grid system that spreads much faster than heavy-weight grid implementations. Most of the current desktop grids are based on master/slave paradigm. In such centralized approach, the master is the one who has the opportunity to present independent tasks to a large number of slaves. On the other hand, if such desktop grids can share the resources and their owners can use other's desktop resources, the many user concept of the traditional grid trend is also realized. We build a light-weight extensible framework, called the *Mini-Grid Framework* supporting the formation of peer-to-peer desktop grids. Compared to the traditional approaches, the Mini-Grid Framework supports:

- *Ease of deployment:* The Mini-Grid environment can be setup with minimal configuration and installation effort,
- *Symmetric resource usage:* The users can contribute their resources to the Mini-Grid environment as well as use the available resource in the environment,
- *Smaller scale grid formation:* The Mini-Grid environment can be created by combining the power of the computers at an institutional level or at an organizational level,
- *Decentralized task distribution:* The Mini-Grid Framework adopts auction strategies for dynamic resource discovery and selection.
- *Resource and task modeling:* We have used ontology based context model to describe resource capabilities and to model resource requirements of an application
- *Application specific quality parameters:* The Mini-Grid Framework supports specification of application specific quality parameters for scheduling. We achieve this by modeling the application specific quality parameters as bids used in auction process.

The research methodology consists of two distinctive phases that aim to design, develop and deploy and evaluate the Mini-Grid Framework. The first phase consists of the following steps:

1. Survey of existing desktop grid system and identification of research gap
2. Architectural design of the Mini-Grid Framework supporting decentralized task distribution by adopting auction-based scheduling
3. Defining an extensible API for the formation of ad hoc Mini-Grid environment
4. Development of the Mini-Grid Framework
5. Perform experimental evaluation at ITU for evaluation of the framework against efficiency and measuring the overhead
6. Integrate the developed framework with CLC Bio Workbench and real-world application
7. Deployment at iNano research center for evaluation of the framework
8. final version 1.0 of the Mini-Grid Framework

The second phase consists of the following steps:

1. Architectural design of the Mini-Grid Framework supporting modeling of resources and applications using their context information and modeling of application specific quality parameters as bids

2. Development of the architectural design
3. Integrate the developed framework with CLC Bio Workbench and real-world application
4. Perform control experiments at ITU for evaluation of the framework against efficiency and measuring the overhead
5. Deployment at iNano research center for evaluation of the framework
6. Final version 2.0 of the Mini-Grid Framework

The pilot deployment at iNano research center evaluates the framework against:

- Ease of deployment
- Support for dynamic resource participation
- Symmetrical resource sharing
- Support for heterogeneous resource participation

1.4 Contributions

This research investigates on the development and the deployment of the Mini-Grid Framework for resource management in ad hoc grids using market based scheduling and context based resources and application modeling. The author of this thesis has contributed to the field of study as given below.

1. Data collection - Survey of existing desktop grid computing systems
2. Software produced
   (a) Design of Conceptual Architecture of the framework
   (b) Design of Module Architecture of the framework
   (c) Design of Execution Architecture of the framework
   (d) The Mini-Grid Framework Version 1.0
   (e) The Mini-Grid Framework Version 2.0
   (f) Building of simulation environment for the evaluation of the framework under resource failures
3. Technical Reports
   (a) Technical report on " Design and Development of the Mini-Grid Framework "
   (b) Tutorial for the Mini-Grid Framework

(c) Technical report on " Mini-Grid message Field Definitions, General Message Format and Message Sections "

(d) Technical report on " RNA secondary structure prediction on an ad-hoc peer-to-peer network infrastructure " (along with others)

4. Publications

(a) Jakob E. Bardram and Neelanarayanan Venkataraman: The Mini-Grid Framework: Application Programming Support for Ad-Hoc, Peer-to-Peer Volunteer Grids. GPC 2010: 69-80

(b) S. Kailash, P. Prasanna, V. Prabha, and Neelanarayanan V: Semantic Resource Description for Grid. Asia International Conference on Modelling and Simulation 2007: 112-115

1.5 Thesis Outline

The organization of the remainder of this thesis divides as follows:

Chapter 2 presents a survey of state-of-the-art desktop grid systems by using a taxonomy based on resource provision, scalability, supported quality of service, resource utilization and deployment effort.

Chapter 3 proposes the Mini-Grid Framework based on a peer-to-peer architecture that manages a pool of resources; and owing the resources to the infrastructure with applications having easy access to them.

Chapter 4 presents our approach to model computational capability of resource and computational capability requirement of tasks based on their context information. We also present the resource selection process that involves matching capability requirements to available resource capabilities using semantic technology.

Chapter 5 presents the some insights in the use of the Mini-Grid Framework in developing a sample mini-grid application, grid enabling an application toolkit, and the possible extensions to the framework.

Chapter 6 presents the performance evaluation of the Mini-Grid Framework by using lab experiments to determine the efficiency of the framework and the overhead caused by the framework. The performance of the framework has been compared with other systems.

Chapter 7 presents the performance evaluation of the Mini-Grid Framework by using simulation experiments to determine the efficiency of the framework in a volatile environment.

Chapter 8 presents the deployment study of real-world application on top of the Mini-Gird Framework in a biology lab. The learning during the deployment and the evaluation of the framework has been presented.

Finally in Chapter 9, we summarize the research contribution of this thesis and pointers to further research. Currently, the framework supports minimal support for failure handling. We have identified the need for support of proactive

failure handing mechanism, trust based security and efforts to scale Internet wide computing platform.

Chapter 2

Related Work

A distributed system built on top of a network appears as a single entity by hiding the existence of multiple autonomous computers and provides the user with required service. There are several definitions for distributed systems from different view points. Coulouris defines a distributed system as "a system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing" Coulouris et al. [2005]. Tanenbaum defines it as "a collection of independent computers that appears to its users as a single coherent system" Tanenbaum and van Steen [2007]. Thus distributed systems consists of computer systems that contain multiple processors connected by a communication network.

There are various types of distributed systems, such as Clusters, Grids, Peer-to-Peer networks, and so on. Cluster refers to a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single, integrated computing resource Baker and Buyya [1999].Cluster computing provides a cost-effective parallel computing platform as a network of nodes (for example, PCs or workstations or SMPs) with memory, I/O facilities and operating system, providing a comparable performance to those expensive, specifically designed parallel super computers at a fraction of the cost. Often, clusters have special network/switch with its own communication protocol for fast communication among its nodes. The cluster middleware offers an illusion of single system image. Programming environment includes message passing libraries, debuggers, and profilers. Message passing libraries libraries allow efficient parallel programs to be written for distributed memory systems. The most popular message-passing systems are PVM (Parallel Virtual Machine)[1] from Oak Ridge National Laboratory, and MPI (Message Passing Interface) defined by MPI Forum[2].

The growth of the Internet, along with the availability of powerful computers and high-speed networks at low-cost commodity components have enabled the clustering of a wide variety of geographically distributed resources, such as su-

[1] www.csm.ornl.gov/pvm
[2] www.mpi-forum.org

percomputers, storage systems, data resources, instruments and special devices, and services, which can be used as a unified resource. Furthermore, they have enabled seamless access to and interaction among these distributed resources, services, application and data. The new paradigm that has evolved is popularly termed as "Grid" computing. Ideally, a grid should provide full-scale integration of heterogeneous computing resources of any type. However, real world grid implementations are more specialized and generally focus on the integration of certain types of resources. As a result, we have different types of grids: computational grid, data grid and service grid Krauter et al. [2002]. Computational grid focus on integration of computational resources (for example, High Performance Clusters). Data grid focus on integration of data storage resources (for example, Network Attached Storage device). The European DataGrid project focus on the development of middleware service to enable distributed analysis of physics data from CERN[3] Gagliardi et al. [2002]. Service grid provides a service that cannot be achieved through one single computer. In such grids, several different resources exist each providing a specific function that needs to be aggregated in order to collectively perform the desired service. For example, we can have a service obtained its functionality by integrating and connecting databases from two separate VOs in order to output their correlation Taylor and Harrison [2009].

Similar to Grid computing, peer-to-peer (P2P) computing is a relatively new computing discipline in the realm of distributed computing. Grid computing has many similarities to P2P computing. The difference between grid computing and P2P computing is in the design point. Grid computing systems tend to be designed to operate well for the case in which there are small number of very powerful locations, whereas P2P computing system tend to be designed for a very large number of weaker machine Verma [2004].

At present, computational grids can be broadly classified into two types. The first type, the "classical" computational Grid system provide rich feature-sets (e.g., resource discovery service, multi-user authentication, etc.) and tend to concern themselves primarily with providing access to large-scale powerful computational resources. The second general class of computational grids is the Desktop grid. Desktop grid systems aggregate the idle CPU cycles of desktop resources from individuals or from within organizations or from multiple organizations. The desktop grid system provide high computational capability at low cost and this motivates its use. However, there are several distinct differences between them in terms resource participation, resource sharing nature, application support, service quality, deployment effort required etc. Building a desktop system has to consider resource's heterogeneity, non-dedication, volatility, failures, security, etc. Therefore, it is important to comprehend current research approaches in desktop grid system development to identify research gaps.

This research aim to develop a software platform that allows aggregation of resources and provision of resources to grid applications. Hence, a taxonomy has been provided focusing from resource and application perspectives. Then,

[3]www.cern.ch

the proposed taxonomy has been mapped to few existing desktop grid systems.

2.1 Introduction

While the concept of a "computing utility" providing "continuous operation analogous to power and telephone" can be traced back to the Multics Project in the 1960s Corbató and Vyssotsky [1965]. The term the "Grid" has emerged in the mid 1990s to denote a proposed computing infrastructure which focuses on large-scale resource sharing, innovative applications, and high-performance orientation Foster et al. [2001]. The grid concept provides virtual organization environment for resource sharing and problem solving across multiple institution. The sharing involves a direct access to computing resources, storage devices, network resources, software, scientific instruments and other resources subjected to highly controlled sharing rules. The sharing rules define clearly what is shared, who is allowed to share, and the sharing conditions. A set of individuals and or/institutions defined by such sharing rules forms a virtual organization Foster et al. [2001]. Researchers and corporations have developed different types of grid computing systems to support resource pooling or sharing. Typically, such grid computing systems can be classified into computational and data grids. In Krauter et al. [2002], a taxonomy for grid systems is presented, which proposes a three types of grid systems. As stated earlier, they are computational grids, data grids and service grids.

Now, let us focus on computational grids that optimizes execution time of applications that require greater number of computing processing cycles. The "Computational Grid" refers to the vision of a hardware and software infrastructure providing dependable, consistent, fast, and inexpensive access to high-end computational capabilities Foster and Kesselman [1999]. Generally speaking, such grid computing platforms can be classified into two main categories; classic high-end grids and volunteer or desktop grids Kurdi et al. [2008]. Classical grids provide access to large-scale, intra-and-inter institutional high capacity resources such as clusters or multiprocessors Foster and Kesselman [1999]; Foster [2006]. For example, Globus Foster [2006] and Legion Chapin et al. [1999] provide a software infrastructure that enables applications to handle distributed heterogeneous computing resources, normally dedicated, in multiple administrative domains. TeraGrid[4], build on the Globus Toolkit is one such example. However, installing, configuring, and customizing Globus middleware requires a highly skilled support team, such as the London e-Science Centre[5] or the Enabling Grids for E-science project[6]. Participating in grid projects involves time consuming networking and training processes. The application developer must possess a knowledge of both the middleware being used and the underlying computational hardware. Using this information, task dependent libraries and binaries can be produced. These are typically managed by the user, who also

[4] www.teragrid.org
[5] www.lesc.imperial.ac.uk
[6] www.eu-egee.org

has to possess some knowledge of the target architecture. This makes the process of application deployment both time consuming and error prone Cafferkey et al. [2007]. Globus based grid computing infrastructure requires third party resource brokers or meta-schedulers for task distribution. Meta-scheduler, a software scheduling system, allows a designated node to act as an active gateway for other passive nodes, for example GridWay[7]. Thus effort towards development, integration, testing and packaging of many components are substantial. For these reasons, the deployment and operational cost of such systems are substantial, which prevents its adoption and direct use by non-technical users. For example, NSF Middleware Initiative (NMI) program[8] has invested roughly $50M for development of various components. TeraGrid cyber infrastructure facility has allocated approximately 25% of the staff to common integration functions and 75% of the staff to resource provider facility functions Catlett et al. [2006].

Consider a situation where the participants are offering different resources to collaborate on a common objective. In this scenario, every participant wants to participate for a certain limited amount of time, normally till the participant has some utility interest in the participation. Administrative overheads erupting from classical grid participation make it impractical for such transient communities to undergo a formal grid establishment process.

Volunteer or Desktop Grids, in contrast, is designed to distribute computational tasks between desktop computers owned by individuals at the edge of Internet or idle desktop computers in an institution. Volunteer Grid computing systems allow formation of parallel computing networks by enabling ordinary Internet users to share their computer's idle processing power Litzkow et al. [1988]; Anderson [2004]. Projects based on volunteer computing system provide computational infrastructures for complex computational intensive research problems that range from searching for extraterrestrial intelligence (SETI@home) to exploring new HIV/AIDS treatments (fightAIDS@home). Such systems require setting up a centralized control system responsible for managing the contributed clients, who in turn periodically request work from a central server. Volunteer computing is highly asymmetric; it is a 'master-slave' architecture in which volunteers supply computing resource but do not submit any work to be done. Public outreach and incentive structures (like high-score competitions) play a significant role in attracting volunteers. In volunteer computing, users require system administration and database expertise to create and operate projects Maurer [2005].

Historically, the Condor project Litzkow et al. [1988] pioneered using the idle time of organizational workstations to do parallel computing. Increasing computational power and communication bandwidth of desktop computers are helping to make distributed computing a more practical idea. By using existing desktop computers from a local network, the cost of such an approach is low compared with parallel supercomputers and dedicated clusters. The main dif-

[7] www.gridway.org
[8] www.nsf-middleware.org

ference in the usage of institutional desktop grids relatively to volunteer ones lies in the dimension of the application that can be tackled. In fact, while volunteer grid computing projects usually embrace large applications made up of a huge number of tasks, institutional desktop grids, which are much more limited in resources, are more suited for modestly-sized applications.

Peer-to-peer platforms provide an operating system independent middleware layer, which allows sharing of resources in a peer-to-peer fashion. Various protocols for supporting P2P service discovery (e.g. Gridnut Talia and Trunfio [2004] and GridSearch Koh et al. [2006]) and P2P resource discovery Pham et al. [2006] has been proposed. Grid computing focus on infrastructure. On the other hand, peer-to-peer computing focus mainly on scalability and not infrastructure. However, a convergence between peer-to-peer and Grid computing has been foreseen in literature Foster and Iamnitchi [2003]; Trunfio et al. [2007].

2.2 Overview of Desktop Grid Systems

In this section we present some of the well known grid systems but with special focus to desktop grid systems. While this list is not exhaustive, but merely representative from this research point of view. The survey includes desktop grid systems that have been demonstrated with proof of concept application or working prototype or have been widely used or deployed in real environment. The survey excludes research work that reports theoretical model with no implementation and evaluation are excluded from the study.

2.2.1 BOINC

The Berkeley Open Infrastructure for Network Computing (BOINC) Anderson [2004]; Anderson et al. [2006] is a software platform for distributed computation using idle cycles from volunteered computing resources. BOINC is composed of a central scheduling server and a number of clients installed on the volunteers' machines. The client periodically contacts the server to report its availability and gets workload. BOINC is mainly based on voluntary participants connected through Internet.

Although projects using BOINC are diverse in their scientific nature, in general they are data analysis applications composed of independent tasks that can be executed in parallel. Each project must prepare its data and executable code to work with the BOINC libraries and client/server infrastructure. Also they need to set up and maintain their own individual servers and databases to manage the project's data distribution and result aggregation. Though each project requires individual server setup and maintenance, BOINC users can participate in multiple projects through single client interface.

The BOINC server consists of seven different daemon programs, some of which are provided by BOINC and others need to be implemented individually for each project Buck [2005]. The *feeder*, and *transitioner*, are components provided by BOINC. The BOINC server maintains a queue of work units that need

to be sent to the clients. The feeder retrieves newly generated work units from the database to fill the queue. The transitioner controls the state transitions of work units and results throughout their lifecycles. The lifecycle of a *work unit* begins when it is generated by the *work generator* and is added to the database. The work generator daemon need to be developed by the application. The work units can then go through several state transitions as they are distributed to one or more clients for execution. If a client has received a work unit and has not returned the results within a predetermined amount of time, then the work unit is said to have *timedout* or *expired*. The transitioner detects work units that have timed out and redistributes them to different clients. The lifecycle of a work unit ends when enough valid results for that work unit have been collected and a single result called the *canonical result* is chosen for that work unit. Similar to work unit, the result can also undergo several state transitions. All results that are invalid or not selected to be the canonical result are deleted.

Database purge and *file deleter* daemons provided by BOINC. The file deleter and the database purge daemons remove files and work units that are no longer needed to keep the database size at constant size. The *validator* daemon attempts to determine which results are valid by comparing results from several different clients. The assimilator daemon processes valid results, which usually mean storing them to a separate database for later analysis.

The BOINC architecture is highly modular and scalable. If the project server becomes inundated with client requests, additional servers can be added to the project with daemons running on all the servers each handling only a fraction of the total incoming requests. With a sufficient number of project servers, the only bottleneck in the system is the MySQL[9] server Anderson et al. [2005].

BOINC, a powerful and robust system for public resource computing has significant limitations. The BOINC client has been ported to several platforms, but the BOIN server can only be executed on Linux-based operating systems. Hence, researchers or application developer need to have expertise in Linus system administration and database expertise Maurer [2005]. Furthermore, project creators must have a large knowledge of C++ or Fortran programming Silva et al. [2008]. Compared to the high complexity of BOINC system, there is very little documentation available about how to create a BOINC project. This lack of documentation is the largest barrier that researchers face when creating BOINC project Baldassari [2006].

Projects must have a large visibility in order to attract enough cycle donors (i.e., volunteers) Silva et al. [2008]. Resource providers are concerned with the potential harm inflicted by Internet sharing systems, especially when the installation or use of the system requires administrator (or root) privileges Pan et al. [2006].

[9] www.mysql.com

2.2.2 DG-ADAJ

DG-ADAJ (Desktop Grid - Adaptive Distributed Application in Java) Olejnik et al. [2006] is a middleware platform, facilitating Single System Image (SSI), to enable efficient execution of heterogeneous applications with irregular and unpredictable execution control in desktop grids.

DG-ADAJ is designed and implemented above the JavaParty and Java/RMI platforms. DG-ADAJ automatically derives graphs from the compiled bytecode of a multi-threaded Java application that account for data and control dependencies within the application. Then, a scheduling heuristic is applied to place mutually exclusive execution paths extracted from the graphs among the nodes of the desktop grid system.

DG-ADAJ does not include methods for resource brokering. Before scheduling any computational jobs resources need to be found/selected manually by the user. ADAJ applications had to be written utilizing the JavaParty, so the application had to be tightly coupled with the platform.

2.2.3 SZTAKI Local Desktop Grid

SZTAKI Local Desktop Grid Balaton et al. [2007] is built on BIONC technology but significantly extends the client concept of BOINC in order to enable the creation of hierarchical desktop grids within a large organization or community.

The hierarchical Desktop Grid allows a set of projects to be connected to form a directed acyclic graph. Task is distributed among the edges of the directed graph. By doing this, SZTAKI can reduce load on the primary BOINC server by using the second and third-level BOINC servers. But, each level of BOINC servers still has the same characteristics of the original BOINC, for which performance is not guaranteed. Communication and data transfer between the client and the desktop grid system is performed via HTTPS.

SZTAKI LDG focuses on making the installation and central administration of the local desktop infrastructure easier by providing tools to help the creation and administration of projects and the management of applications.

SZTAKI LDG supports DC-API for easy implementation and deployment of distributed applications on local desktop grid environment.

2.2.4 distributed.net

A very similar effort to BIONC is distributed.net Distributed.net. However, the focus of the distributed.net project is on very few specialized computing challenges. Furthermore, the project releases only binary code of the clients and hence impossible to adapt in other projects. The project that attracted the most participant was an attempt to decipher encrypted messages. However, the volunteers were provided with cash prizes.

2.2.5 XtremWeb

XtremWeb Fedak et al. [2001], a research project from University of Paris-Sud, France aims to serve as a substrate for large scale distributed computing. Similar to BOINC, XtremWeb is based on the principle of cycle stealing.

XtremWeb supports the centralized setup of servers and PCs as workers. However, it can be used to build a peer-to-peer system with centralized control, where any worker node can become a client that submits jobs. It implements three distinct entities: the coordinator, the workers and the clients to create a XtremWeb network. Clients can be used by any user to submit tasks to the XtremWeb network. They submit the task to the coordinator and permits the end user to retrieve results. The workers installed on volunteer hosts execute the task.

Tasks are scheduled to workers according to their demand (i.e., pull model) in FIFO (First In First Out) order Petiton et al. [2006]. All actions and connections are initiated by workers. The coordinator registers every worker connection. The communication between coordinator and workers are encrypted for network security. The workers downloads the executable software and all other related components (e.g., the input files, the command line arguments for the executable binary file, etc). Workers sends the result (the output files) to the coordinator.

During registration workers provide configuration information such as XtremWeb worker version, operating system, CPU type, memory size, etc. XtremWeb coordinator performs matching about CPU type, OS version and Java version Lodygensky et al. [2003]. Coarse grained, massively parallel and applications that are not communication intensive are suited for deployment on XtremWeb Petiton et al. [2006]. XtremWeb uses replication and checkpointing for fault tolerance Djilali et al. [2004].

XtremWeb-CH Abdennadher and Boesch [2007] is an upgraded version of XtremWeb with major improvements in communication routines and improved support for parallel distributed application.

2.2.6 Alchemi

Alchemi Luther et al. [2005] is an open source project from Melbourne University developed in C# for Microsoft .NET framework. It allows flexible application composition by supporting an object-oriented application programming model with a multithreading paradigm. Grid application consists of several grid threads that can be executed in parallel. Although tightly coupled with .NET platform, Alchemi can run on other platforms using Web services. Alchemi is based on the master-worker parallel programming paradigm.

Alchemi grid consists of three components: manager, executor, and owner. The manager manages grid application execution and thread execution. Executors sign up with manager. Owner submits grid threads to manager who adds them into to a thread pool. Then the manager schedules threads for execution on available executors. Executors after completion of execution of the threads submit the results to the manager. Later, the owner can retrieve the results

from the manager.

Aneka Chu et al. [2007], an improved version of Alchemi allows the creation of enterprise Grid environments. It provides facilities for advance reservation of computing nodes and supports flexible scheduling of applications.

2.2.7 Bayanihan

Bayanihan Sarmenta and Hirano [1999] is a web-based volunteer computing framework that allows users to volunteer their resources.

Bayanihan system consists of client and server. A client executes Java applet on a web browser. It has a worker engine that executes computation or a watcher engine that shows results and statistics. A server consists of HTTP server, work manager, watch manager and data pool. The HTTP server serves out Java class file. The work manager distributes tasks and collects result. Bayanihan basically uses eager scheduling, in which a volunteer asks its server for a new task as soon as it finishes execution of current task. Eager scheduling works by assigning distributed tasks to hosts until every task has been scheduled at least once. At this point, redundancy is introduced by rescheduling any task that has not yet completed. This process continues until every task completes. The client uses remote method invocation technique called HORB Satoshi [1997] on the server to get a task.

The applications are mainly compute-intensive and independent. In addition, Bayanihan supports applications running in Bulk Synchronous Parallel (BSP) mode, which provides message-passing and remote memory primitives Sarmenta and Hirano [1999].

2.2.8 Condor

Condor Tannenbaum et al. [2001] developed at the University of Wisconsin-Madison scavenge and manage wasted CPU power from otherwise idle desktop workstation across an entire organization.

Workstations are dynamically placed in a resource pool whenever they become idle and get removed from the resource pool when they become busy. Condor can have multiple resource pool and each resource pool follows a flat resource organization. Condor collector listens for resource availability advertisements and acts as a resource information store. A Condor resource agent runs on each machine periodically advertising its availability and capability to the collector. Condor provides job management mechanism, scheduling policy, priority scheme, resource monitoring and resource management. Condor is comprised of a server and large number of volunteers. A central manager in the server is responsible for matchmaking, scheduling and information management about task and resources.

Condor provides ClassAd Raman et al. [1998] in order to describe characteristics and requirements of both tasks and resources. It also provides a matchmaker for matching resource requests (tasks) with resource offers (i.e., available resources). Condor provides Directed Acyclic Graph Manager (DAGMan) DAG

for executing dependable tasks. Condor enables preemptive-resume scheduling on dedicated compute cluster resources. It can preempt a low-priority task in order to immediately start a high-priority task.

Condor-G Frey et al. [2002] is the technological combination of the Globus and the Condor projects, which aims to enable the utilization of large collection of resources spanning across multiple administrative domains. Globus contribution composes of the use of protocols for secure inter-domain communication and standardized access to a variety of remote batch systems. Condor contributes with the user concerns of job submission, job allocation, error recovery and creation of user friendly environment.

2.2.9 Entropia

Entropia Chien et al. [2004] facilitates a Windows desktop grid system by aggregating the desktop resources on the enterprise network.

The Entropia system architecture consists of three layers: physical node management, resource scheduling and job management layer. The physical node management layer provides basic communication, security, local resource management and application control. The resource scheduling layer provides resource matching, scheduling of work to client machines and fault tolerance. The job management layer provides management facilities for handling large number of computations and data files. A user can interact directly with resource scheduling layer by means of API or can use the management functionalities provided by job management layer. The applications are mainly compute intensive and independent.

Entropia virtual machine (EVM) runs on each desktop client and is responsible for starting the execution of the subjob, monitoring and enforcing the unobtrusive execution of the subjob, and mediating the subjobs interaction with the operating system to provide security for the desktop client and the subjob being run. The EVM communicates with the resource scheduler to get new task (subjobs), and communicates subjob files and their results via the physical mode management layer.

The resource scheduler assigns subjobs to available clients according to the client's attributes such as memory capacity, OS type, etc. The Entropia system uses a multi-level priority queue for task assignment.

The applications are mainly compute intensive,independent and involve less data transfer between server and clients Chien [2003].

2.2.10 QADPZ

QADPZ (Quite Advanced Distributed Parallel Zystem) Vladoiu and Constantinescu [2008] is an open source desktop grid computing system. The users of the system can transmit computing tasks to these computers for execution in the form of dynamic library, an executable program or any program that can be interpreted, for example Java, Perl, etc. Messages between the components

of the system are in XML format and can optionally be encrypted for security reasons.

A QADPZ consists of three types of one master, many slaves and multiple clients delegating job to the master.

- Master - A process running on the master computer responsible for jobs, tasks, and slaves accounting. The slaves talk to the master when they join or leave the system, or receive or finish tasks; the clients talk to the master when they start or control user jobs or tasks.
- Slave - A process running on the volunteer computer as a daemon or Win NT service. The slave communicates with the master and starts the slave user process. Without the slave running the slave computer cannot take part in collaborative computing.
- Client - A process running on a client computer. It communicates with the master to start and control jobs and tasks of a specific user, may also communicate directly with slaves and is responsible for scheduling the tasks of user jobs as required by particular user application.
- Slave computer - One of many computers where the distributed collaborative computation takes place, for example: a UNIX server, a workstation, a computer in a student PC Lab, etc.
- Client computer - Any computer that the user uses to start his application. A client computer may be a notebook connected to the network using a dial-up connection, a computer in the office, lab, etc.

All slaves participating in the system run a slave program that accepts the tasks to be computed. The master program keeps track of the status of the slave computers. The master registers the status of the slaves, i.e., idle or busy computing a QADPAZ task or disabled because a user has logged out. When a user wishes to use the system, he prepares a user application consisting of two parts: A *slave user program*, i.e., the code that will effect the desired computing after being distributed to the slaves, and the *client* that generates jobs to be completed.

The user can choose the QADPZ standard client, which allows him to set up and submit a job. The job description is saved into an XML-formatted project file and can be manually edited by more advanced users. Alternately, a user may want to write his or her own client application to have full control over the submission of tasks. It is possible to directly write a slave library to speed up the execution. In that case, the slave service or daemon will not start a new process with the downloaded executable but a dynamic shared library will be loaded by the slave process.

QADPZ is implemented in C++ and uses MPI as its communication protocol. The client and the slaves talk to each other by the use of a shared disk space, which is certainly a performance bottleneck and requires costly synchronization Fuad [2007].

All the components need to be installed manually and have their own configuration files requiring manual configuration. The system has been used in the area of large-scale scientific visualization, evolutionary computation, and simulation of complex neural network models.

2.2.11 Javelin

Javelin, a Java-based infrastructure for global computing with a goal to harness the Internet's vast, growing, computational capacity for ultra-large, coarse-grained parallel applications. The work that started with SuperWeb has been continued with new versions named Javelin++, Javelin 2.0, Javelin 3.0, CX, JANET and currently JICOS, each with improvements in performance, scalability, computation and programming model. Javelin Christiansen et al. [1997]; Neary et al. [1999b, 2000] is a Java based infrastructure. Applications run as Java applet in Javelin version or screen saver in Javelin++ version Neary et al. [1999a].

Javelin consists of three entities: broker, client and host. A broker is a system-wide Java application that functions as a repository for the Java applet programs, and matches the client tasks with hosts. A client submits a task to the broker through a Web browser by generating an HTTP POST-request, and periodically polls the broker for the results. Hosts are Web browsers, generally running on idle machines, that repeatedly contact the broker for tasks to perform, download the applet code, execute it to completion, and return the results to the broker. The communication between any two applets is routed through the broker Christiansen et al. [1997]. The Javelin is hindered by Java applet security model. Hence, Javelin 2.0 abandoned the applet-based programming framework and added support for problems that could be formulated as branch and bound computations. However, the essential architecture remained almost same.

In Javelin, the work stealing is performed by a host, and eager scheduling is performed by a client. Applications are mainly compute-intensive and independent. Currently, Javelin supports branch and bound computations. Javelin achieves scalability and fault-tolerance by its network of brokers architecture and by integrating distributed deterministic work stealing with a distributed deterministic eager scheduling algorithms. The main advantage of this architecture are:

- URL based computation model permits the worker hosts to perform the necessary computation off line and later reconnect to the Internet to return the results from the computation.
- Administrative overheads involved in preparing worker nodes (during deployment) is minimized by ubiquity of Web browsers.

However, centralized match making process and task repository by broker limits Javelin. Further, it assumes the user's a priori knowledge of the broker's URL address. Eager scheduling leads to excessive data traffic and presence of multiple copies of the same task.

2.2.12 Charlotte

Charlotte Baratloo et al. [1999] developed at New York University is a web-based volunteer computing framework and implemented using Java. Charlotte supports Java-based distributed shared memory over the Java Virtual Machine and parallelism. The key feature of the shared memory architecture is that it requires no support from the compiler or the operating system, as it is the case with most shared memory architectures. The implementation of distributed shared memory is at data level. Every basic data type in Java has a corresponding Charlotte data type. Charlotte maintains data consistency by using the atomic update protocol allowing concurrent reads but exclusive write/updates.

Charlotte, based on Calypso's programming model to provide parallel routines and shared memory abstraction on distributed machines. The main entities in Charlotte program are a manager (i.e., a master task) executing serial steps and one or more workers executing parallel steps called *routines*. A routine is analogous to a standard Java thread, except for its ability to execute remotely. A distributed shared name-space provides shared variables among routines. The manager process creates an entry in well-known Web page. Volunteer users load and execute the worker processes as Java applets embedded into web pages pointed by pointing their browsers to this page. A direct socket connection provides the manager-worker communication.

Initially, it uses eager scheduling and then redundant assignment of tasks to multiple clients to handle client failures and slow clients. Multiple executions of a task can result in incorrect program state. Charlotte uses *two-phase idempotent execution* (TIES) to ensure idempotent memory semantics in the presence of multiple executions. TIES ensures guarantees correct execution of shared memory by reading data from the master and writing it locally in the worker's memory space. Upon completion of a worker the dirty data is written back to the master who invalidates all successive writes, thus maintaining only one copy of the resulting data Batheja and Parashar [2001]. Further, in order to mask latencies associated with the process of assigning tasks to machine by employing dynamic *granularity management*. Granularity management technique (bunching) assigns a set of task to a single machine at once. The size of bunch is computed dynamically based on the number of remaining task and number of available machines.

Charlotte makes use of existing Internet infrastructure such as HTTP servers and web browsers for running applets. As with Javelin project, Charlotte is also hindered by Java applet security model. Charlotte programs must conform to the parallel routine program structure and must be implemented as a Java applet. This approach is not very transparent or flexible because programmers must adhere to both the applet API and Charlotte's parallel routine structure Baratloo et al. [1997]. Charlotte does not address the question of how a Web browser, i.e., the worker, finds the work. Earlier version requires manual input of the URL location. However, current version uses a registry and lookup service provided by KnittingFactory Baratloo et al. [1997]. The primary function of the manager is not only scheduling and distributing works but also

responsible for communication of workers as well as applet-to-applet communication. Hence, the manager could be a bottleneck. Further, Charlotte requires that a manager run on a host with HTTP server. If multiple applications are run by single machine, then multiple managers are needed to run on this machine. Hence, it would introduce additional overhead and leads to performance problem.

2.2.13 CPM

Compute Power Market (CPM) Buyya and Vazhkudai [2001] is a market-based middleware system for global computing on personal computing devices connected to the Internet. It applies economic concepts to resource management and scheduling across Internet-wide volunteer computing.

It consists of a market, resource consumers and resource providers. The market consists of a market resource broker and market resource agent. The market resource broker is component of resource consumer and the market resource agent is a component of resource provider. It maintains information about resource providers and resource consumer – a kind of registry. The resource consumer buys computing power using the market resource broker. The market resource broker finds suitable resource based on the information provided by the market. It is responsible for negotiating cost with resource providers and task distribution to resource providers. The resource provider sells computational power using market resource agent. The market resource agents updates resource information and deploys and execute tasks. The resource broker selects the resource based on deadline or budget.

CPM takes the advantage of real markets in the human society. However, the centralized market servers introduce limitations, such as single point of failure and limited scalability. Furthermore, the centralized market server requires additional organizations for regular maintenance Xiao et al. [2005].

2.2.14 POPCORN

POPCORN Nisan et al. [1998] is a Web-based global computing system for scheduling Java-based parallel applications. POPCORN uses market model for matching sellers and buyers.

It consists of a market, buyers and sellers. The sellers provide their resources to a buyer by using Java enabled browser. The market has the popcoin, a currency used by buyer to buy resources. The seller can earn the popcoin for selling its resources. The market is responsible for performing matching between buyers and sellers, transferring task and result between them and for account management.

POPCORN uses several market models for scheduling. The Popcorn system implements three different kinds of auction mechanisms: Vickrey auctions, first-price double auctions, and k-price double auctions. Popcorn has a central repository by means of a Web platform for information aggregation. However, this platform may become a communication bottleneck.

The applications are mainly compute-intensive and independent.

2.2.15 The Spawn System

Waldspurger Waldspurger et al. [1992] et al. proposes a distributed system that uses market mechanism for allocating computational resources called Spawn.

The Spawn system organizes participating resources as trees. A leaf resource can join randomly selected auction among the available auctions. The managers of each leaf serve as funding sponsors for their children. In case an agreement is reached after the auction process, a resource manager controls the communication with and monitoring of the supplied resources.

The Spawn system supports tree-based applications in which partial results are computed on different levels of the tree and are subsequently sent to a leaf on a higher level of the tree. On each leaf, a manager combines and aggregates the results received from its children. Subsequently, this manager reports the aggregated results to its higher leaf on the tree.

The Spawn system uses market models for scheduling. The managers of each leaf serve as funding sponsors for their children. This funding is used to purchase CPU resources for subtasks associated with each leaf. These resources are purchased by participating in Vickrey auctions which are instantiated by each node that offers idle CPU slice. In Vickrey auction uses sealed-bid that means bidding agents cannot access information about other agent's bid. Further, the winner pays the amount offered by the next highest competitive bidder. The auction instance and the pricing information are advertised to the neighbors of each node. Out of all available auctions, a requesting leaf randomly joins one particular auction. In case an agreement is reached after the auction process, a resource manager controls the communication with and monitoring of the supplied resources.

The authors through simulation of prototypical implementation show that Vickery auctions leads to economically efficient outcomes. In addition it has lesser communication costs compared to an iterative auction. However, as the information is only propagated to neighbors, new information is disseminated with delays. Besides, the seller-initiated auction is suitable only for heavily loaded systems. In lightly loaded system, the buyer-initiated auction is proved to outperform the seller-initiated auction Eager et al. [1986].

2.2.16 OurGrid

OurGrid system Andrade et al. [2005]; Cirne et al. [2006] is an open, free-to-join, cooperative grid in which labs donate their idle computational resources in exchange for accessing other labs' idle resources when needed.

OurGrid is based on a peer-to-peer network, where each labs in the Grid correspond to a peer in the system. OurGrid has three main components: the OurGrid peer, the MyGrid broker and the SWAN security service. The MyGrid broker is responsible for providing the user with high-level abstractions for resource and computational task. The SWAN security service guarantees secure

resource access. Each peer in represents a lab. Each peer has direct access to a set of resource. Each resource has an interface called GridMachine that provides access to the resource.

The OurGrid system mainly focuses on compute intensive and independent tasks. The OurGrid system provides resource matching and heuristic scheduling. The OurGrid architecture implements the idea of symmetric resource participation.

2.3 Classification of Desktop Grids

In this section, we present the various classification of desktop grids. There are several taxonomy of Desktop Grids. Baker et al. present the state of the art of Grid computing and emerging Grid computing projects. They hierarchically categorize Grid systems as integrated Grid systems, core middleware, user-level middleware, and applications/application driven efforts Baker et al. [2002]. Krauter et al. proposes a taxonomy of Grid focusing on resource management. Their taxonomy classifies the architectural approaches from the design space Krauter et al. [2002]. Venugopal et al. proposes a taxonomy of Data Grids based on organization, data transport, data replication and scheduling. Sarmenta classifies volunteer computing into application-based and web-based Venugopal et al. [2006]. Chien et al. classifies Desktop Grid into Internet scalable Grid and Enterprise Grid Chien et al. [2004]. Gilles Fedak et al. provide a taxonomy of Desktop and Service Grids based on distribution of computation units, scalability, type of resource contribution, and organization Fedak et al. [2008]. SungJin Choi et al. presents the taxonomy of Desktop Grid systems and mapping of taxonomy to the existing Desktop Grid systems Choi et al. [2007]. However, their taxonomy is focused on scheduling in Desktop Grid systems. We provide a taxonomy and mapping that is more generic and inclusive in nature. The objective of this taxonomy are to:

- Categorize based on attributes related to system architecture, resource provision, and grid deployment effort.
- Provide mapping of main desktop grid system according to the taxonomy.

2.3.1 Centralized Desktop Grids

Desktop grid system can be categorized into *centralized* and *decentralized* desktop grids. Centralized desktop grid system consists of three logical components: client, workers and server. Client allows platform users to interact with the platform by submitting jobs and retrieving results. Worker is a component running on the volunteer's desktop computer which is responsible for executing jobs. Server, a coordination service connects clients and workers.The server receives jobs submissions from clients and distributes them to workers according to the scheduling policy. Servers also manage fault tolerance by detecting worker crash or disconnection. If needed tasks are restarted on other available workers.

Workers on completion of task execution return the results to the server. Finally, the server verifies the correctness of the results and stores them for the client to download them. Typical examples are BOINC, XtremWeb, Entropia etc. Properties of centralized desktop grid has been presented in Table 2.1 and 2.2.

The desktop grid system can be classified based on deployment and resource location into *local* and *global* desktop grids. Local desktop grid, also known as Enterprise desktop grid, consists of desktop PC hosted within a corporation or University interconnected by Local Area Networks (LAN). Several companies such as Entropia and University projects such as Condor, SZTAKI local desktop grid, etc., have targeted these LANs as a platform for supporting desktop grid applications. Such grid environments have better connectivity and have relatively less volatility and heterogeneity than global desktop grids. Local desktop grids connect resources from the same administrative domains, normally by using local-area networking technologies and poses low security risks. Internet desktop grids connects resources from different administrative domains, normally by using wide-area networking technologies. They aggregate resources provided by end-user Internet volunteer. Global grids are characterized by anonymous resource providers, poor quality of network connections, being behind firewalls, having dynamic addressing techniques (DHCP) and poses high security risks. Desktop grids based on BOINC such as SETI@HOME, Einstein@HOME, etc. falls under global desktop grids.

In order to maximize the potential work pool and minimize setup time, a volunteer computing system must be accessible to as many people as possible. Desktop grid system can be categorized into Web-based and middleware-based desktop grid computing according to the accessibility platform running on volunteers. In the web-based desktop grid systems, clients write their parallel application in Java and post them as Applet on the Web. Then, volunteers can join by pointing to the web page using their browsers. The Applet gets downloaded and runs on the volunteer's desktops. Typical examples are Charlotte, Bayanihan, Javelin and so on. In the middleware-based desktop grid computing systems, volunteers need to install and run a specific middleware that provides the services and functionalities to execute parallel applications on their machine. The middleware fetches tasks from a server and executed them on volunteer desktops, when the CPU is idle. Typical examples are BOINC, XtremWeb, Entropia, and so on.

In desktop grid environment, large number of people will want to share their resources. In addition, a higher number of application will be deployed. Therefore, special attention should be given to ease of deployment of new application and preparation of the execution environment. In Laszewski et al. [2002], von Laszewski et. al. identify three deployment strategies: thick, thin and slender deployment based on the hosting environment. In thick deployment, an administrator uses a software enabling service, for example Grid Packaging Tool (GPT) GPT to install the Grid software on a resource participating in the Grid. In thin deployment, end user uses a Web browser to access Grid service in a transparent fashion, for example Charlotte, Bayanihan, Javelin, etc. In slender deployment, slender clients are developed in an advanced programming

DG System	Platform	Scheduling Policy	Deployment Effort
BOINC	Middleware-based	Simple	Thick deployment
Entropia	Middleware-based	Simple	Thick deployment
distributed.net	Middleware-based	Not available	Not available
XtremWeb	Middleware-based	Simple	Thick deployment
DG-ADJ	Middleware-based	Deterministic	Thick deployment
POPCORN	Middleware-based	Economic	Thick deployment
Spawn	Middleware-based	Economic	Thick deployment
Bayanihan	Web-based	Simple	Thin deployment
QADPZ	Web-based	Simple	Thick deployment
Javelin	Web-based	Deterministic	Thin deployment
Charlotte	Web-based	Simple	Thin deployment

Table 2.1: Centralized Global Desktop Grid System

language such as Java and made available from a Web-based portal. Initially, end user can install these slender clients on their resources to participate in the Grid. Additionally, slender deployment provides an automatic framework for updating the component if a new one is placed on the Web server.

Scheduling policy matches tasks with resources by determining the appropriate tasks or resources. It can be classified into three categories: simple, model-based and heuristics-based Choi et al. [2006]. In simple approach, tasks or resources are selected by using First Come First Served (FCFS) or randomly. This scheduling policy is appropriate for high throughput computing requirement and commonly implemented by major desktop grid middleware like BOINC, Condor, XtremWeb, etc. The model-based approach is categorized into deterministic, economy, and probabilistic models. The deterministic model is based on structure or topology such as queue, stack, tree or ring. Tasks or resources are deterministically selected according to the properties of structure or topology. For example, in a tree topology, tasks are allocated from parent nodes to child nodes. In deterministic scheduling models, a set of jobs has to be processed by a set of machines and certain performance measures have to be optimized. In the economy model, scheduling decision is based on market economy. In the probabilistic model, resources are selected in probabilistic manners such as Markov, machine learning or genetic algorithms. In the heuristic-based approach, tasks or resources are selected by ranking, matching, and exclusion methods on the basis of performance, capability, weight, etc.

DG System	Platform	Scheduling Policy	Deployment Effort
Condor	Middleware-based	Simple	Thick deployment
SZTAKI LDG	Middleware-based	Simple	Thick deployment

Table 2.2: Centralized Local Desktop Grid System

DG System	Platform	Scheduling Policy	Deployment Effort
CPM	Middleware-based	Economic	Thick deployment
OurGrid	Middleware-based	Deterministic	Thick deployment

Table 2.3: Distributed Global Desktop Grid System

2.3.2 Distributed Desktop Grid System

A distributed desktop or peer-to-peer Grids constructs a computational overlay networks using tree, graph or distributed hash table. In the absence of a server, volunteers need to maintain partial information about other volunteers in the grid environment. Scheduling is performed at each volunteer depending on the computational overlay network. Volunteers exchange their information between other volunteers. The volunteers self-organize themselves into a computational overlay network based on a criteria such as resource capability or timezone. A client can submit a set of independent computational tasks to known volunteer. The known volunteer distributes tasks based on scheduling mechanism. Each volunteer executed the computational task and returns the result to parent volunteer (in the tree). Finally, the parent volunteer returns the result to the client. Distributed desktop grids have been presented in Table 2.3.

Typical examples are Compute Power Market (CPM), OurGrid, etc. Other P2P Desktop Grid platforms such as the Organic Grid Chakravarti et al. [2005], Messor Montresor et al. [2003], Paradropper Zhong et al. [2003], and the one developed by Kim et.al. Kim et al. [2007]. These platforms are still under development and do not have any applications implemented on top of it. Further, they have not been deployed and evaluated. Hence they do not provide a solid substrate enabling the deployment, use, and management of a production level desktop grid environment Anglano et al. [2010]. As stated earlier, we have excluded such projects from the survey.

2.3.3 Computational Grids based on JXTA Technology

The JXTA platform defines a set of protocols designed to address the common functionality required to allow any networked device to communicate and col-

laborate mutually as peers, independent of the operating system, development language, and network transport employed by each peer.

Several projects have focused on the use of JXTA as a substrate for grid services. The P2P-MPI project, the Jalapeno project, the JNGI project, the JXTA-Grid project, the OurGrid project, the P3-Personal Power Plant project, the Triana project, the Xeerkat project, the NaradaBrokering project, and Codefarm Galapagos are listed in the JXTA web-site (as on 2008). However, none of these projects are being actively developed Antoniu et al. [2005].

Out of this 10 projects, 8 are FOSS projects. Currently, the P2P MPI project, the OurGrid project, the Triana project and the XeerKat project have switched to other platforms. The Jalapeno project, the JNGI project and the P3 project stopped their development around 2005-2006. The JXTA-Grid project is not yet in production Ferrante [2008].

JNGI uses software developed by Project JXTA to communicate in peer-to-peer fashion. Peer groups are used as fundamental building block of the system. The system consists of monitor groups, worker groups, and task dispatcher groups. The monitor group handles peers joining the system and redirects them to worker groups if they are to become workers. The system can have multiple monitor groups in a hierarchical manner. During task submission the submitter queries the root monitor group for an available worker group which will accept its tasks. The root monitor group decides the sub-group to which the request should be forwarded to. The request would finally reach a worker group. The worker group consists of workers performing the computations. Once the request arrives at a worker group, a task dispatcher in the task dispatcher group of that worker group will send a reply to the submitter. The task dispatcher group distributes individual tasks to workers. The task dispatcher group consists of a number of task dispatchers, each serving a number of workers. If a task dispatcher disappears, the other task dispatchers will invite a worker to become a task dispatcher and join the task dispatcher group. Task dispatchers periodically exchange their latest results and thus results are not lost even if a task dispatcher becomes unavailable. The submitter polls the task dispatcher about the status and results of previously submitted tasks.

The Jalapeno system consists of manager peers, worker peers and task submitter peers. Each host can play one or more of these roles. Initially every host starts a worker peer that starts to search for available manager peers. When a manager is found, the worker tries to connect to it. Manager can have only a limited number of connected workers and rejects any worker when this limit has been reached. Accepted workers will join the worker group created by the manager and start executing tasks. If a worker is unable to connect to a manager within certain time, it will start a new manager peer on the local host and connect to it. The first host becomes a manager after some time and start to accept worker peers. When a manager becomes unavailable its workers will either find another manager or become managers themselves. Workers with a worker group can communicate with other workers or the manager. The task submitter submits a collection of tasks to a randomly chosen manager. The manager splits the bundle into a set of smaller bundles. The manager keeps a

limited set of bundles from which tasks are extracted and handed to the connected workers. The rest of the bundles are forwarded to a number of other, randomly chosen, managers which repeat the process. Bundles which are not forwarded are returned to the task submitter. When a worker finishes a task it will return the result to its manager which in turn will forward the result to the task submitter.

In P3 system, peers form their own base peer group that is a subgroup of always existing JXTAs base group. A peer can run *host* daemon to share resources and/or *controller* tool to access resources shared by the others. To distribute a job to resource providers, a user first creates a distinct peer group for the job, job group, with the controller tool. After joining the group itself, controller publishes an advertisement of his job within the group. It includes a description of the job. Peers running a host daemon discover the advertisements of job groups made by controllers. According to their policy, they decide whether they want to join a job group and contribute to processing the job. If a host decide to join, it discover and obtain a Java Archive (JAR) from the controller. The archive contains compiled Java application code.P3 has an object passing library, by which an application can unicast or broadcast Java objects, and receive them synchronously or asynchronously. P3 provides two parallel programming libraries a master-worker library and a message passing library for application developers. The master-worker library supports master-worker style parallel processing and the message passing library supports MPI programming. If a worker leaves a job group, a workunit delivered to the worker but not completed is delivered to another worker.

2.3.4 Jini for Grid Computing

Jini Waldo [2000], a Java-based infrastructure developed by Sun Microsystems provides a distributed programming model that includes dynamic lookup and discovery, support for distributed events, transactions, resource leasing and a security model. A number of research groups have considered Jini in the realization of their Grid middleware Juhasz et al. [2002b,a]; Huang [2003]; Hampshire and Blair [2003]; Teo and Wang [2004]; Furmento et al. [2004]; Kent et al. [2008].

Jini is built on top of RMI, which introduces performance constraints Hawick and James [2000]; de Roure et al. [2003]. Further Jini is primarily concerned with communication between devices and hence file system access and processor scheduling need to be implemented de Roure et al. [2003]. ALiCE (Adaptive and scaLable internet-based Computing Engine) Teo and Wang [2004], a technology for developing and deploying general-purpose grid applications implemented in Java, Java Jini Waldo [2000] and JavaSpaces[10]. The ALiCE system consists of a core middleware layer that manages the Grid fabric. It hosts compute, data and security, monitoring and account services. The core layer includes an Object Network Transport Architecture (ONTA) component that deals with the transportation of objects and associated code across the Grid fabric. The core

[10]http://river.apache.org/doc/specs/html/js-spec.html

layer uses Jini technology to support the dynamic discovery of resources within the Grid fabric. For object movement and communications within the Grid, a Jini based tuple space implementation called JavaSpaces is used.

JGrid Hampshire and Blair [2003], based on Jini aim at exploiting Jini's support for dynamic self healing systems. To virtualize a local computing resource, *Compute Services* are introduced that manage job execution on the local JVM. These Compute Services can be hierarchically grouped using *Compute Service Managers*. Resource brokers schedule jobs onto these Compute Service Managers and Compute Services on a client's behalf. JGrid supports a number of application models including batch style execution using a Sun Grid Engine backend, Message Passing Interface (MPI). JGrid has extended Jini's discovery infrastructure to cope wide with wide area discovery.

2.4 Discussion

We have presented a taxonomy for desktop grid systems. The taxonomy focuses on resource provision, scalability, organization, resource participation, resource utilization, task dependency, task generation pattern, supported quality-of-service, and deployment effort. We also survey some of the desktop grid system and map them using the taxonomy. On the basis of taxonomy and mapping, we present the current state of research in desktop grid systems. Then we identify the possible research gaps.

2.4.1 Desktop Grid Architecture

Desktop grid system provide high computational power at low cost by reusing existing infrastructure of resources in an organization. However, most of the existing desktop grid systems are built around centralized client-server architecture Fedak [2010]. This design could potentially face issues with scalability and single point of failure Fedak [2010]; Choi et al. [2007]. In volunteer desktop Grid systems, central task scheduler would become a potential bottleneck when scheduling large number of tasks Chakravarti et al. [2005].

In contrast to existing Grid paradigms, models based on peer-to-peer architecture offers an appealing alternative Liu and Antonopoulos [2010]. Most of the current research on peer-to-peer architecture do adapt centralized approach in one form or other. For example, JXTA can be considered as hybrid peer-to-peer system as it has the concept of super peers. Jini uses lookup service to broker communication between the client and service and this approach appears to be a centralized model.

For the announcement of new auction instances, their states and current market prices, Popcorn installs a central repository by means of a Web platform. As a disadvantage of this central information aggregation the Web platform becomes a communication bottleneck Schnizler [2007]. In Spawn, market information is propagated only to neighbors and new information is disseminated with delays. This imperfect knowledge could result in performance trade

offs Schnizler [2007].

2.4.2 Desktop Grid Adoption

Volunteer desktop Grid systems, in general, employ "many resource providers - few users" model, meaning that any user can join and offer resources, but only a selected few users can make use of those resources. Hence, volunteer desktop computing is highly asymmetric Rezmerita et al. [2007] in which volunteers contribute computing resources but do not consume any. Thus the asymmetric nature of desktop Grid systems acts as one of the primary impediment to widespread adoption by users. From our survey, we could see that CPM and OurGrid alone support symmetric resource participation. CPM is based on market economy model and OurGird is based on P2P architecture model. Hence, we adopt distributed computational economy framework for quality-of-service (QoS) driven resource allocation. Such framework provides mechanisms and tools that realize the goal of both resource provider and resource consumer. Resource consumers need a utility model, representing their resource demand and preferences. And resource providers need an expressive resource description mechanism.

2.4.3 Grid Application Development

Currently, most of the grid applications are implemented with the help of computer scientists. Providing MPI-like grid application programming language is not sufficient, as many scientists are not familiar with parallel programming language Jin [2005].

One of the major resource for researchers is existing application toolkits. However, many existing applications are developed for desktop computers. Hence, grid enabling existing application with minimal efforts could be useful to the researchers.

Grid computing not only have technical challenges but also have deployment and management of grid environment challenge.

2.4.4 Quality of Service

Desktop computing systems, for example Entropia and P2P systems, for example Gnutella provide high levels of quality of service for specialized service. But are too specific for generalized use; e.g., it would be difficult to see how the Gnutella protocol[11] could be useful for anything but searching for files or data content. A Grid should be able to provide non-trivial quality of service, for example, this is measured by performance, service, or data availability or data transfer. QoS is not just about aggregation of capabilities of participating resources in the Grid. QoS is application specific and it depends on the needs of the application. For example, in a physics experiment, the QoS may be specified

[11] http://rfc-gnutella.sourceforge.net/

DG System	Model	Description
Spawn	Vickrey auction	The Spawn system provides a market mechanism for trading CPU times in a network of workstations. Tree-based concurrent programs are sub-divided into nodes. Each node holds vickrey auction independently to acquire resources.
POPCORN	Auction	Each buyer (resource consumer) submits bid and the winner is determined through one of the three auction protocol implemented: Vickery, Double and Clearing House.
CPM	Various models	CPM supports various market model such as commodity market, contract-net/tendering and auction.

Table 2.4: Summary of Market based Systems

in terms of throughput, but in other experiments, the QoS may be specified in terms of reliability of file transfers or data content. From our survey, we could see that only Compute Power Market (CPM) provides quality of service using differentiated price service.

2.4.5 Scheduling Policy

First-Come-First-Served (FCFS) is the classical eager scheduling algorithm for bag-of-tasks applications, where a task is simply delivered to the first worker that requests it. This scheduling policy is particularly appropriate when high throughput computing is sought and thus it is commonly implemented by major desktop grid middleware like BOINC, Condor and XtremWeb. However, FCFS is normally inefficient if applied unchanged in environments where fast turnaround times are sought, especially if the number of tasks and resources are in the same order of magnitude, as it is often the case for local user's bag-of-tasks Kondo et al. [2004]. As an alternate distributed scheduling mechanism such as market-based scheduling can be considered. The evaluation results of computational and data grid environments demonstrate the effectiveness of economic models in meeting user's QoS requirements Abdelkader et al. [2009].

2.4.6 Limitations of Jini and JXTA

As both Jini and JXTA have Java based implementations, the features of the Java language for dealing with OS and hardware heterogeneity are inherited by both frameworks. Jini prescribes the use of leases when distributing resources across the network. The leasing model as a whole contributes to the development of autonomous self healing services that are able to recover from crashes and clear stale information without administrator intervention. The framework extends the Java event model in a natural way, incorporating measures that enable the developer to react to the intricacies of delivering events across a, possibly unreliable, network. Although Jini does not mandate the use of RMI for proxy to service communication, RMI is mandatory when interacting with the Jini event and transaction models. To interact with the reference implementation of the Jini lookup service and to renew leases, RMI is also required since the lookup service and lease objects are represented by a RMI proxy Vanmechelen [2003]. Java RMI is acceptable when transferring small or medium sized chunks of data over hig-speed data links but being slower than TCP. Another issue is the performance degradation imposed by the HTTP tunneling technique Vanmechelen [2003]. JXTA project lacks in the documentation, hence design and implementation decisions are not documented. Often, it requires inspection of the implementations source code to determine the different steps and policies used throughout the implementation of the JXTA protocols Vanmechelen [2003]. JXTA does have a good set of tools that allow using resources available in the network. However while working with this platform many mistakes can be noticed. The documentation for many important elements hardly exists. The tools for XML processing, used mainly with advertisements, are difficult to utilize. Despite of supposed popularity of JXTA there are still no professional products based on it. Implementation of JXTA protocols still has many undocumented elements mgr inż. Marcin Cieślak [2007].

2.4.7 Research Gap

Research in desktop grid system focus on two areas: desktop grid application development and desktop grid system development. Research on desktop grid application development focus on scheduling, resource management, communication, security, scalability, fault tolerance, trust, and architectural model.

The grid application development research focus on developing new applications suitable for desktop grid systems. However, writing and testing grid applications over highly heterogeneous and distributed resources are complex and challenging. Hence, the desktop grid system should support ease of application development and integration.

The strength and weakness of current approaches to desktop grid system development can be summarized as follows:

- Most of the current desktop grid system follow client-server or centralized peer-to-peer architecture. These approaches are easy to implement and

have high throughput. However, they are not scalable, have a single point of failure, and server component become a bottleneck.

- Distributed desktop grid system based on market approach has been proposed. However, use of multiple markets for resource sharing has not been explored.
- Most of the current desktop grid system focus on asymmetric resource utilization. However, support for symmetric resource utilization could motivate users to contribute their resources to the desktop grid initiative.
- Currently desktop grid system do not support quality of service based on non-trivial parameters.
- Current market-based resource management approaches in desktop grid computing focus on price-based mechanisms to allocate resources. In price-based mechanisms, the price represents supply/demand condition of resources in an economic market. Instead of price, a utility function can be used to represent these condition. However, this aspect has not been given adequate attention.
- Currently either thin or thick deployment technologies have been widely used. However, slender deployment fosters adoption of Gird by scientific community.
- Porting existing desktop application or development of new grid application should be possible with minimal efforts.

Apart from these issues, resource failure and lack of trust among participants are also important challenges to be addressed.

2.5 Infrastructure Awareness System

The success of a volunteer computing infrastructure depends on the contribution of the users. We have used awareness techniques developed by Juan et. al., Hincapié-Ramos et al. [2011] has been used to recruit contributors during deployment. The details of techniques and their impact on resource participation has been reported in Hincapié-Ramos et al. [2011].

2.6 Summary

In this chapter, we have surveyed different desktop grid systems currently available. We have analyzed the existing desktop grid system and identified the research gap.

Part II

The Mini-Grid Framework

Chapter 3

The Mini-Grid Framework

In this chapter, we present the first part of this thesis contribution, which is the programming API and runtime infrastructure for formation of ad-hoc "mini-grids" in a local area network. The main goal of the proposed infrastructure, based on a peer-to-peer architecture, is to manage a pool of resources; and owing to the infrastructure with applications having easy access to them.

3.1 Motivations and Design Objectives

The use of computers for data analysis and visualization of results are currently playing a fundamental role in the working methodology of many research groups. As a consequence, having access to high-performance computing has become essential for many applications. Many solutions have been proposed to use the widespread availability of desktop computers to address the computational requirements of a common class of applications, named *embarrassingly parallel*, or *bag-of-tasks* (BoT) applications. These are parallel applications that can be decoupled into a large number of independent tasks that do not need to communicative with each other.

In previous chapter, we presented the survey of existing desktop grid systems. The survey reveals the following short comings of existing desktop grid systems.

- Classical grid systems follow either centralized or hierarchical architecture and a regulated control for membership and access privileges. These grid architectures assume a dedicated administrative authority for policy enforcement, monitoring and access control. The participants in such grid share a synchronized and non-conflicting objective. There exists well defined collaborations based on some predefined usage policies and access privileges Choi and Buyya [2010].

- Current desktop grid systems follow either centralized client-server architecture or some form of centralization approach in peer-to-peer architecture. However, systems with the centralized approach can suffer from

high computational overhead and may result in overall system performance degradation. Further, the centralized server or root node of the hierarchical organization has the drawback of being a single point of failure. Current use of desktop grid computing is highly asymmetric, i.e., volunteers supply computational capabilities but do not consume computational capabilities.

- Current desktop grid systems do not support application specific quality parameters. For example, a certain application needs to be finished before a fixed deadline, another application does not want its task distributed to nodes at specific location, etc. However, current approaches in resource allocation focus on quality attributes related system centric policies such as time and cost. The role of market driven strategies has been identified by Jia Yu and Rajkumar Buyya for application specific quality of service metric Yu and Buyya [2005].

- Currently, grid application development focus on development of new applications rather than porting existing application toolkits.

Thus most of the current approaches to grid computing is very centralized both technically and in use. However, there are situation that requires a transient, short lived and one-time collaboration among participants. One example of such a situation is the scenario where a group of researchers want to contribute their computational resources for a common objective. In this example, every participant want to participate for a limited amount time, normally until the participant has some interest in the collaboration. For such a group it may be infeasible to build or participate in a computational grid infrastructure because of administrative overheads, cost, etc. In other words, existing grid models do not support the "ad hoc" collaboration among participants. In such environment, users may have energy-saving habits, such as setting computers to hibernate mode when not in use or turning off during weekends, resulting in intermittent resource availability.

3.2 Design Objectives

We build an extensible light-weight framework, called the *Mini-Grid Framework* having auction-based task allocation incorporating application specific quality parameters for resource selection, modeling resources and computational task based on their context information, and supporting formation of peer-to-peer grid infrastructure requiring minimal configuration and deployment efforts. The design objectives of the Mini-Grid Framework can be summarized as:

- To support symmetric resource usage by adopting peer-to-peer architecture.

- To address resource volatileness due to absence of centralized control by having market-based scheduling.

- To support application specific quality parameters for resource selection.
- To model capabilities offers of resource and capability requirements of grid applications using their context.
- To extend functionality provided by the framework.
- To minimize efforts required for integration of the framework with existing application toolkit and
- To minimize efforts required for deployment of grid application.

3.3 Conceptual Architecture of the Mini-Grid Framework

The Mini-Grid Framework can be used to create a peer-to-peer computational infrastructure for the execution of BoT applications. The framework leverages from the fact that most computers are not fully used on a continuous basis as researches alternate between application execution and performing experiments in the lab. The Mini-Grid Framework proposes the model of contributory computing system, where the users provide their own resources to be used collectively. The users of the system typically provide resources to the extent of their possibilities, as they wish to help the community. This will result in intermittent resource availability, as users can decide whether to contribute or not at any moment. Being provided by independent users, no homogeneity of resources can be guaranteed. The process of deployment and application execution should be made as easy and intituitive as possible. No complex administration must be required.

This section presents a conceptual architecture of the Mini-Grid Framework. First we introduce the fundamental concepts behind the framework design. Then we present the conceptual framework based on the above fundamental concepts and discuss the design space of the framework.

3.3.1 Concepts

In this section, we focus our attention on key concepts behind the Mini-Grid framework, which will be expanded upon in the rest of the chapter. The information contained in this section will provide a level of knowledge that will make succeeding sections more meaningful.

Resource - any physical entity such as a laptop, desktop, server, cluster, instruments, etc. that participates in the mini-grid environment. Resources have an IP address and are connected to a computer network. Thus a given resource can communicate with all other resource in Mini-Grid.

Client - a software entity that runs on a node participating in Mini-Grid. The client can play two types of roles namely resource provider and resource consumer. Resource providers have computational resources and are willing to

share them. Resource consumers need computational resources to perform their computational tasks. The client can play either roles and or both.

Resource consumer - a role played by a client interested in consuming other's resources. The components of resource consumer include *Submitter, Auctioneer, TaskBus, and MessengerComponent.* These components are explained in detail in section 3.5.1.

Resource provider - a role played by a client interested in providing its resources. The resource include computing power, storage space, network bandwidth, etc. The components of the resource provider include *Executor, Bidder, TaskBus, TaskExecutor, and MessengerComponent.* These components are explained in detail in section 3.5.2.

Monitored entity - an physical or conceptual object of interest in Grid environment. The monitored entities can be divided into computational, network, application and storage entities. For example, processor, disk, etc. are computational entities. The monitored entity has a set of attributes whose value represent its state and environment.

Context statement - a statement describing a feature or an attribute of an monitored entity. For example, "current CPU utilization is 25%" is a statement describing the current CPU load. Context statement can provide either static or dynamic information about monitored entity. For example, clock speed of a CPU, a static parameter that indicates the rate at which a CPU can execute instructions, does not change until the CPU is replaced. On the other hand, CPU utilization, a dynamic parameter that indicates the current load condition, changes over time. Each context statement is expressed in terms of Object-Attribute-Value triplets. Object corresponds to monitored entity and the attribute corresponds to properties of the monitored entity. The value can be a literal or another monitored entity.

Task - represents a unit of work in our framework such as file transfer or execution of a function in an algorithm. Tasks are submitted by a client acting as a resource consumer and are executed on clients acting as a resource provider. Each task contains an executable along with required data and all information required to execute the task in Mini-Grid, called *TaskContext.*

TaskBus - a distributed data structure that enables dynamic exchange of tasks between resource provider and resource consumer. The TaskBus offers an interface to the client to send and receive Tasks.

TaskContext - a data structure that contains all information required for execution of a task in Mini-Grid such as computational and memory requirements, network bandwidth requirements, operating system requirements etc. It can be considered as an alternative to *Resource Specification Language (RSL)* of Globus RSL [2007] or *Job Submission Description Language (JSDL)* proposed by the JSDL working group of the Global Grid Forum Anjomshoaa et al. [2007].

ResourceContext - a data structure that contains static (e.g. operating system, cpu speed) and dynamic (e.g. number of tasks being executed, amount of free memory) properties of a resource that enable selection of most adequate resource satisfying application requirements. It can be considered equivalent to *GLUE schema* Andreozzi et al. [2009] adopted by gLite middleware gLite or

classAds adopted by Condor Raman et al. [2004].

MessengerComponent - the message communication channel that provides basic message transport service. Currently both reliable (UDP, IP multicast) and reliable (TCP, reliable unicast) messages are supported. The messenger component offers an interface that can be used to send and/or receive messages.

Sensors - heterogeneous and distributed sources from which context information can be obtained. This context information can be raw and device-specific format. Sensors can be classified as two types:

- Hardware sensors: represent hardware sensors capable of capturing physical data. For example an RFID scanner scanning an RFID-enabled physical object.

- Software sensors: represent software programs that collect data from software components. For example, a SNMP agent - a software module in a managed network device responsible for maintaining local management informations and delivering that information.

However, currently the Mini-Grid Framework consist of software sensors only.

Context monitors - software components that gathers context information about a monitored entity from one or more sensors. For example, a "process monitor" detects changes in CPU characteristics of a desktop computer. Context monitors registers them self with a context interpreter and provide the gathered context information as a set of context statements.

Context interpreter - software component that converts context information in one format to another format. Context interpreters are used to achieve physical data independence. Physical data independence indicates that the physical storage structure used for storing context information could be changed without necessitating a change in the conceptual schema used by context monitors. The change is absorbed by conceptual/internal mapping.

Context base - a registry for storing context information about different monitored entities. Context information are expressed as context statements. Loosely speaking, a context statement describes an attribute variable of a managed object. These statements are collected into context base. Structure of context base can be defined by context models based on ontology.

Query interface - provides a query mechanism for context information by consumers of context information. The query interface supports ready-made mold or templates for providing context information in a typical situation. Query template are domain oriented context structures. Given a template name, the query interface, the query interface provides the necessary context of the situation represented by that template name. Query interface are used to achieve logical data independence. Changes to the query representing the template, such as addition or deletion of attributes, must be possible without having to rewrite context consumers. In other words, the context consumers that refer to

the template must work as before, after the query undergoes a logical reorganization.

3.3.2 Conceptual Architecture

Mini-Grid follows the general vision of a desktop grid computing system, turning a set of resources into a runtime environment for task distribution and execution. The Mini-Grid Framework aggregates resources such as desktop computers, laptop computers, clusters, etc., in a Local Area Network (LAN) for building a desktop grid environment called the "Mini-Grid" environment.

A conceptual illustration of the Mini-Grid Framework is shown in Figure 3.1. The Mini-Grid Framework consists of four logical components: *ResourceProviders, ResourceConsumers, TaskBus* and *MessengerComponent*. Each resource participating in Mini-Grid has a software entity called "client". On the one hand, a client can be used by a resource to consume the computational capabilities of other resources. On the other hand, it can also be used by a resource to provide its computational capabilities to other resources. That is, a client can play both "Resource Consumer" and "Resource Provider" roles. The clients in Mini-Grid communicate with each other using the messenger component. The computational tasks are exchanged between the clients using the task bus.

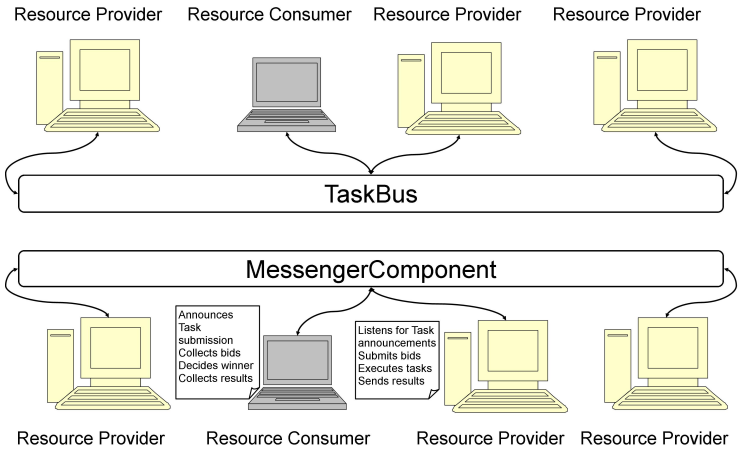


Figure 3.1: A conceptual model of the Mini-Grid Architecture including the TaksBus, MessengerComponent, ResourceConsumers and Resource Providers.

Each client participating in the Mini-Grid environment has three sub-systems namely the task scheduling sub-system, the context awareness sub-system and messaging sub-system. The task scheduling sub-system is responsible for handling issues related to task submission, distribution and execution. The context awareness sub-system is responsible for describing the context of tasks and for collecting and providing context information of resources. The messaging sub-

system is responsible for sending and receiving Mini-Grid messages according to the Mini-Grid Messaging Protocol defined in Section B.

As a distributed computing platform, the framework allows Mini-Grid application running on a client acting as resource consumer to submit tasks to the infrastructure which will be executed on clients acting as resource provider. Clients playing resource consumer role accepts tasks coming from Mini-Grid applications, distributes the tasks to resource providers according to a scheduling policy based on auction, transfers application code to executors, collects and stores task results and delivers task results to Mini-Grid application upon request. Client playing resource provider role listens for task announcements, participates in auction process, executes allocated task and returns completed task along with results.

3.4 Task Scheduling Model

The task scheduling model includes i) model that describes resource requirement, ii) model that describe available resource, iii) task distribution protocol, and iv) model that describe quality of service requirements of the application. A sequential application can be grid enabled by identifying the independent computations that can be executed concurrently. One way is task decomposition, in which the computations are a set of independent tasks that can be executed in any order. Another way is data decomposition, in which the focus is on the data rather than on the different operations applied to the data. The data is partitioned into certain number of parts and the operation can be performed on the different parts Mattson et al. [2004]. There are other patterns of computation do exist. However, these two decomposition are most common. The grid enabled application consists of a collection of independent task with a quality of service requirement. Certain task may require large amount of data to be transfered to or from a remote node before or after the execution of the task at the remote node. However, for better performance each task in the application should have a higher computational complexity index, a ratio between the execution time of the task to its data transfer time.

Each entity in Mini-Grid, for example *grid enabled application*, *task* and `resource` may have a set of characteristics or properties or features or properties. Such information can be called *context* of the entity. Context models can be used for storing and querying the context information associated with an entity and the relationship among entities. For example, task context model can describe the resource requirements (i.e., execution environment of the task). Further, application context model can describe the quality of service requirements and resource context model can be used to describe the capabilities of a resource. We discuss in detail about these context models in

3.4.1 Resource Discovery and Selection

Generally, task distribution in volunteer computing environment involves a "pull" or "push" model. In pull model, the workers (resource providers) request tasks from the master (resource consumer). On the other hand, in push model the master (resource consumer) distributes tasks to arbitrary workers. In the push model, the first task of the scheduler (on behalf of resource consumer) is resource discovery, i.e,. to identify a list of available resources. Once the list of possible resources is known, the second task is selecting those resource that most suitable based on constraints imposed by the grid enabled application or the user. Normally, a scheduling policy selects the suitable resources. The scheduling policy determines how to select appropriate resources. It can be classified into three categories: simple, model-based, and heuristics-based Choi et al. [2007]. In the simple approach, resources are selected by using First Come First Served (FCFS) or randomly. Further, the model-based approach can be categorized into deterministic, economy, and probabilistic models. The deterministic model is based on structure or topology such as queue, stack, tree, or ring. Resources are deterministically selected according to the properties of structure or topology. For example, in a tree topology, tasks are allocated from parent nodes to child nodes. In the economy model, scheduling decision is based on market economy (i.e., price and budget). In the probabilistic model, resources are selected in probabilistic manners (such as Markov model, machine learning or genetic algorithm). In the heuristic-based approach resources are selected by ranking, matching, and exclusion methods on the basis of performance, capability, weight, precedence, workload, availability, location, reputation/trust, etc. The matching method chooses the most suitable resources in accordance to evaluation function. The exclusion method excludes resource according to criteria, and then chooses the most appropriate one among the survivors. Ranking, matching and exclusion methods can be used together or separately. Workload, availability, location/timezone, reputation/trust, performance, capability and weight are used as criteria for selecting resources Choi et al. [2007].

The use of market concepts for resource management is not new. Different research projects Amir et al. [1998]; Lalis and Karipidis [2000]; Nisan et al. [1998]; Sherwani et al. [2004]; Waldspurger et al. [1992] used market concepts for resource management in computational clusters. Market-based resource allocation model which are based on trading and resource brokering policies between resource providers and resource consumers have been proposed in Abramson et al. [2001]; Wolski et al. [2001]; Buyya et al. [2002]; Gomoluch and Schroeder [2003]; Lai et al. [2005]. The mechanism proposed in market-based grid resource allocation include models such as: Auctions (based on outcome of a bidding process), Commodity markets (resource providers specify the charges and resource consumers pay according to the usage of resource), Tenders (based on contracting mechanism - that governs the agreement between resource providers and resource consumers), and Posted price (similar to commodity market except that there are special offers from time to time to attract more users) Buyya et al. [2002]. In addition to these economic models, different pricing schemes

apply to the markets such as: Flat price model (fixed price for a period of time), Competitive economic models (dynamic pricing scheme), Usage timing (peak/off-peak - like telephone services), Prediction-based (based on the ability to predict responses by competitors), Loyalty-based (special prices for regular and loyal users), and Advanced contract based (contract established before resource use determines the price) Buyya et al. [2002]. Economic models have been used in the context of resource allocation, although the important question about which model is the *most* appropriate for supporting resource and allocation in distributed environment is difficult to pinpoint.

Commodity markets rely on polling aggregate supply and demand repeatedly to calculate the equilibrium price and all allocations are performed at this equilibrium price. As in Mini-Grids the resources are not dedicated and supply/demand of resources is very dynamic, the complexity of implementing such centralized market which rely on the aggregate supply/ demand of resources becomes infeasible. Therefore, we have selected auction models as the platform for matchmaking of consumer and producer of resources in Mini-Grids. In the following section, we study different auction models.

3.4.2 Auction Mechanisms

Auctions have long been suggested as a means to allocate resources in a distributed system whereby clients initiate an auction to find the best offer of resources to execute a task Malone et al. [1988]. Auctions have been widely used in solving real-world resource allocation problems and well designed auctions achieve desired outcomes such as high allocative efficiency Klemperer [2002]. The basic philosophy behind auctions is that the highest bidder always gets the resources, and the current price for a resource is determined by the bid prices. In distributed computing systems using auction mechanism for resource allocation, resource providers auction their capabilities using a centralized auctioneer. The consumers bid for resources at prices worth to them.

Auctions can be classified into single-sided or close auction and double-sided auction or open according to whether the auction is static or dynamic. In open auctions, bidders know the bid value of other bidders. In closed or sealed auctions, the participant's bids are not disclosed to others. The value placed by the resource owners on the goods/service determines the pricing in auction based economic models. The consumer whose valuation comes closest to that of the resource owner determines the access to services Buyya et al. [2005]. In these models, there is no global information available about the supply and demand and buyers and sellers usually are not aware of the other's bids or asks and they decide on their local knowledge. An overview of the most popular auction mechanisms is provided below.

In single-sided auction one trader initiates an auction and a number of traders can make a bid. There are four main types of single-sided auction protocol; the English, Dutch, Sealed-Bid, and the Vickrey auction protocol. Resource consumers place their bid with auctioneer.

- In the English auction, the auctioneer announces a price for the service and increases that price incrementally as long as there are at least two resource consumers interested. When the second last resource consumer refuses to stay in the bidding process, the last resource consumer receives the service provided by the resource provider. The winner has to pay the price of second highest bid. The English auction follows a sequential bidding in which buyers take turns publicly to submit increasing bids. Buyers decide a private value depending on their requirements. A bidder stops bidding when its private value is reached.

- In the Dutch auction, the auctioneer calls at a price and lowers this price incrementally as long as no resource consumer is willing to accept it. Once a resource consumer accepts the announced price, it wins the auction and has to pay his bid. The rate of price reduction is up to auctioneer and it has a reservation price below which not to go. Dutch auction may terminate when the auctioneer reduces the price to reservation price and still there is no buyer.

- In The First-Price Sealed-Bid auction, resource consumers submit one single bid to the auctioneer without knowing other's bids. The resource consumer with the highest bid wins the auction and pays the amount of the winning bid.

- The Vickrey or second-price auction is similar to the Sealed-Bid auction, except that the winning resource provider pays the amount of the second best bid. If there is no second-highest bidder, then the price of the commodity is the average of the commodity's minimum selling price and the consumer's bid price.

All four auction protocols yield the same return in private value auctions hence the selection of an auction protocol usually depends on messaging and other implementation requirements Chard et al. [2010]. On the other hand, double-sided auction creates competition on both sides by allowing resource consumers and resource providers to compete with their group to obtain best resource provider (in case of resource consumer) or resource consumer (in case of resource provider) Chien et al. [2005]; Wieczorek et al. [2008]. There are two types of double auctions, continuous double action (CDA) and periodic double auction. Continuous Double Auction matches buyers and sellers immediately on detection of compatible bids. In this type, bids and offers may be submitted at anytime during the trading period. A periodic version of the double auction instead collects bids over a specified period of time, then clears the market at the expiration of the bidding interval Wurman et al. [1998]. Pricing policy adopted by auctioneer can be classified into uniform-price policy and discriminatory policy. In uniform policy, all exchanges occur at the same price determined in auction clearing stage. Whereas in discriminatory policy, the prices are set individually for each matched buyer-seller pair.

For resource allocation in Mini-Grids, we need an auction mechanism that supports simultaneous participation of resource providers/consumers, accommodate variations in resource availability, and can model resource requirements to fulfill the constraints placed by resource providers/consumers. English and Dutch auctions are sequential and are based on open-cry where each bid has to be broadcasted to all participants. This becomes a considerable communication overhead in the context of the Mini-Grid.

First-price auction, Vickery auction and Double auction are simultaneous and closed bid auctions. Besides in lightly loaded system, the buyer-initiated auction is proved to outperform the seller-initiated auction Eager et al. [1986]. We have modeled resource requirements of the application as offers and resource capabilities of the resource providers as bids. The resource that best suits the application requirement wins the auction. We have developed a distributed computational economy model based on auctions for quality-of-service (QoS) driven resource selection. The auction algorithm used for task distribution is explained in next section.

3.4.3 Task Distribution Protocol

The Mini-Grid task distribution protocol is based on auctions. The Mini-Grid Framework uses a 'resource-push' approach of auction-based dynamic resource provision rather than the traditional 'user-pull' approach. In this 'resource-push' approach, the participating resource providers express their interest in executing a computational task dynamically by participating in the bidding process and thus eliminates the requirement of resource discovery mechanism.

The major steps in each phase of task distribution is detailed below. The task distribution involves 3+4+2 steps. The first three steps involve resource discovery, next four steps involve resource selection and task allocation, and the final two steps involve execution management as detailed below.

1. A BoT application generates tasks with a Task Context description, and submits them to the Resource Consumer.

2. The Resource Consumer announces each task to all Resource Providers currently listening to announcements using Messenger Component.

3. On receiving the announcement, the Resource Providers verifies using its local Resource Context to see, if it can submit a bid. If not, it ignores the announcement.

4. The Resource Provider computes the bid based on the announced bidding strategy and its local Resource Context.

5. The Resource Provider submits the computed bid. Once the "Time-to-Bid" (TTB) – prescribed time by which a bid must be submitted for consideration – period elapses, the Resource Consumer proceeds.

6. The Resource Consumer evaluates the submitted bids and selects the optimal Resource Provider for execution of the task.

7. The Resource Consumer announces the winner. The winning Resource Provider gets the task from the TaskBus and executes it. The winning Resource Provider sends an acknowledgment to the winner notification.

8. On completion of the task execution, the Resource Provider sends a task completion notification, and return the task including the result of the execution to the Task Bus.

9. Once the Resource Consumer gets the notification that the task has been executed, it collects the result from the Task Bus. The Resource Consumer sends an acknowledgment to Resource Provider for task completion notification. The submitter informs the application that task execution has been completed through Task Listener. The Task Listener retrieves the completed task from the Submitter.

The pseudo-code for task allocation in the Resource Consumer is shown in algorithm 1. The algorithm proceeds if there is at least one Resource Provider interested in executing the task. If there is no Resource Provider, it times out and informs the application that it cannot schedule the task in Mini-Grid.

Algorithm 1: Task allocation on the Resource Consumer.

```
Input: Submitted task.
Output: Winner notification.
begin
    forall the submittedTask do
        TaskContext taskContext = submittedTask.getTaskContext();
        CALL auctioneer.createAuction(taskContext);
        time ⟵ 0;
        while time > timeToBid do
            submittedBids = CALL auctioneer.getSubmittedBids();
            winningBid = Min(submittedBids);
        end
        CALL messengerComponent.send(winnerNotification);
    end
end
```

The pseudo-code for determining client participation in Resource Provider is shown in algorithm 2. The client invokes the algorithm on receiving a task submission notification.

Since Mini-Grid is ad-hoc and extremely volatile, participating resources can leave at any time. In order to handle cases where an executing resource (a Resource Provider) fails or go off-line, a simple failure handling mechanism has

Algorithm 2: Algorithm for determining client participation in auction.

Input: Task submission notification.
Input: Set of context statements detailing resource context.
Output: Boolean value representing client in Resource Provider participation in bidding. TRUE when client can participate and FALSE otherwise.

```
begin
    forall the taskSubmissionNotification do
        TaskContext taskContext = notification.getTaskContext();
        ResourceContext resContext = ctxManager.getResourceContext();
        Avail ⟵ Set of ContextStatement in ResourceContext;
        Req ⟵ Set of ContextStatement in TaskContext;
        if ∀ContextStatement ∈ Req matches
        ContextStatement ∈ Avail then
            return true;
        else
            return false;
        end
    end
end
```

been implemented as part of the framework. Along with the task submission, the application specifies a value for Time-to-Live (TTL) parameter. TTL is a limit on the period of time. The resource consumer expects the resource provider to complete the execution of the task before this time elapses. If the resource provider does not send a task completion notification before the TTL elapses, the resource consumer assumes that resource provider has left Mini-Grid. Then the resource provider informs the application that it cannot schedule the task in Mini-Grid. The application can run the task locally on the resource consumer or reschedule the task in Mini-Grid.

The default auction strategy is a 'First Price Sealed Bid Auction' Klemperer [2004] in which bidders are not aware of each others' bid value and runs only a single round. First-price sealed bid auctions has minimal communication overhead and hence easy to implement. However other types of auction can be used by extending the framework.

3.5 Module Architecture View

In this section, we describe in detail the software architecture of the Mini-Grid Framework. We show what components are involved in the resource provider and the resource consumer and how they interact.

3.5.1 Resource Consumer - Module View

An UML diagram of the core concepts/classes in the client acting as a resource consumer is shown in Figure 3.2. The core software component is the Submitter deployed on each resource consumer, implementing the functionality of resource consumer. The submitter is responsible for resource (provider) discovery, resource (provider) selection, task allocation and execution management. Resource selection is an automatic process of finding resources capable of executing a given task based on the capabilities of participating resources. In Mini-Grid, we use auction based distributed task allocation mechanism explained in detail in section 3.4.3. Resource selection is also an automatic process of finding suitable resource by matching the capabilities of discovered resources to the requirements of task. Task allocation and execution management involves transferring task to remote resource provider and management of its execution. The resource consumer has the following sub-components: TaskBus, TaskListener, MessengerComponent, BidEvaluator and Auctioneer as shown in the Figure 3.2.

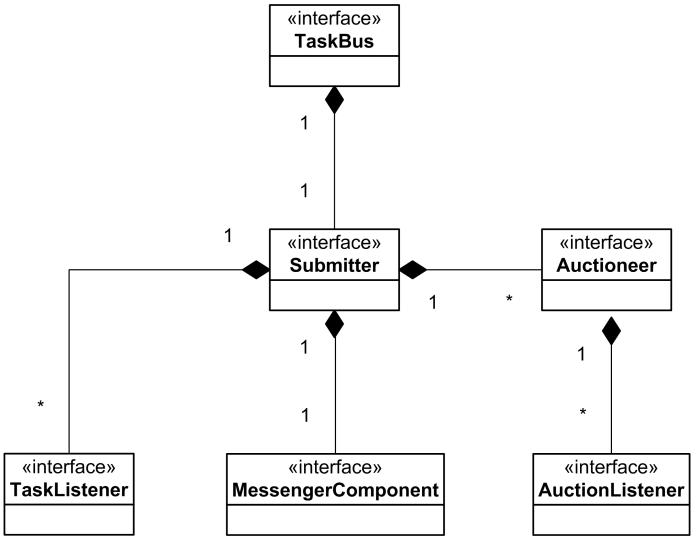


Figure 3.2: Module view of resource consumer.

A Mini-Grid application submits a task to the submitter along with maximum life time of the task i.e., "Time-to-Live" (TTL). The task includes a Task Context description, which includes minimally information about execution environment such as the target software and hardware required and a bidding strategy. The bidding strategy specifies the type of bid to be submitted by the resource provider in the auction process. The auctioneer uses Bid Evaluator to determine the selection of best candidate among the available options and the selected resource provider will be offered with the execution of the submitted task. The submitter puts the task into the task bus and informs the auctioneer

to call for bids. Interested clients acting as resource provider submit bids to auctioneer based on the announced bidding strategy. Once the TTB expires, the auctioneer determines the winner and notifies the same. Based on the current network conditions, the framework determines TTB value. On the other hand, the application can define the optional time-to-live value for each submitted task. The submitter gets the executed task from the task bus. In case, the remote resource provider completes the task execution, the submitter informs the application through the task listener. The application can obtain the completed task encapsulating the result from the submitter using task listener. After the TTL expires the application can query submitter for an executed task.

In case the remote resource provider does not complete the execution of the task within the TTL period, then the submitter tries to ping the remote resource provider. If the ping attempt fails, then the submitter learns that the remote resource has failed or has left Mini-Grid and hence informs the application through task listener about the failure of task execution. The application can either reschedule the task in Mini-Grid or run locally.

The submitter component uses the messenger component to send and receive Mini-Grid messages detailed in Section B.

3.5.2 Resource provider - Module view

An UML diagram of the core concepts/classes in the client acting as a resource provider is shown in Figure 3.3. The core software component is the Executor deployed on each resource provider, implementing the functionality of resource provider. The executor is responsible for participating in the task distribution process and execution of task on the resource provider. The resource provider has the following sub-components: Bidder, TaskBus, TaskExecutor, MessengerComponent and ContextManager as shown in the Figure 3.3.

The executor component of the client receives a bid request from the auctioneer. On receiving the bid request, the executor interacts with the context manager module to get its resource context. Then the resource context verifies if a task can be executed on this resource provider using the obtained context information. The executor rejects or ignores the bid request if the task is not suitable for execution. The bid request contains the bidding strategy used by the auctioneer to decide the winner. Based on the bidding strategy, the executor queries the context manager to get resource context information for calculating the bid. The executor uses, the bidder to determine the bid for the advertised task based on the requested task context and the resource context. The bidder submits the bid to the auctioneer. The auctioneer informs the bidder, if it were the winner in the bidding process. If the bidder is a winner then the bidder informs the executor about winning the bid. The executor retrieves the task from the taskbus and asks the task executor to schedule the task. Once the task executor completes the execution, informs the executor. The executor retrieves the result from the task executor and returns the executed task with results to the taskbus.

The Executor components uses the MessengerComponent to send and receive Mini-Grid messages detailed in section B.

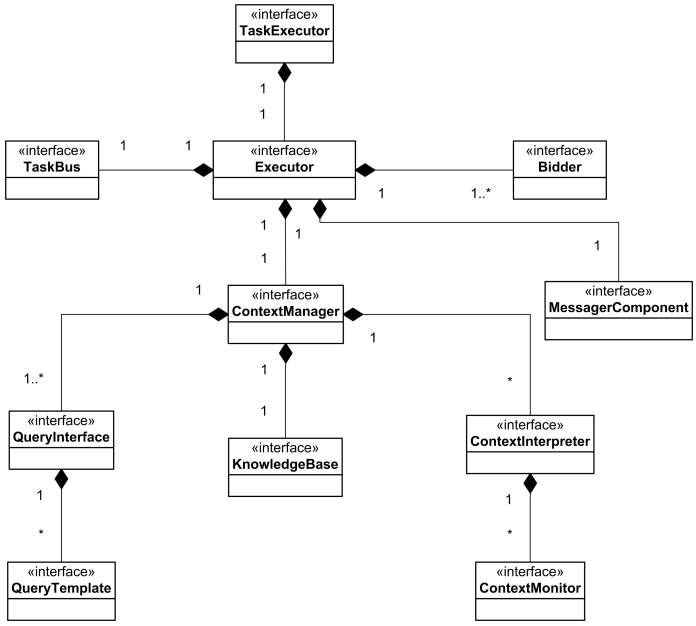


Figure 3.3: Module view of resource provider.

3.6 Execution Architecture View

The Mini-Grid runtime infrastructure consists of resources connected in a peer-to-peer setup, each playing either resource provider or resource consumer or both roles. Each resource has a configuration file having details of multicast IP address and port numbers required for communication. In this section, we discuss the execution architecture of the framework.

3.6.1 Task Submission

Mini-Grid applications use the Mini-Grid Framework for distributing their computational tasks in Mini-Grid. The interaction diagram in figure 3.4 illustrates the task submission at resource consumer. A client acting as a resource consumer can submit one or more tasks for execution. The client calls `submit` method in `Submitter`. The application need to specify the maximum life time of the task (i.e. TTL); the task context information detailing the capability requirements in terms of static and dynamic resource information; and a bidding strategy that needs to be used for selecting the resource provider along with the submitted task. The TTL determines the maximum amount of time that

the submitter can wait for completion of task execution on a remote resource provider. Before the TTL expires the submitter should notify the client the status of the task. If the Task is completed then the results are available for the client otherwise an exception occurs. The application need to handle the exception. It can decide to resubmit the application once again for execution in Mini-Grid or it can schedule locally.

The bidding strategy permits the application developer to model quality of service (QoS) constraints as bids. For example, if the user is interested in scheduling the tasks, generated by the application, on secured resources, then he can define a bid called "SecureBid". The secure bid can be described in terms of security parameters, for example, availability of intrusion detection system and firewall at the resource provider. The quantified numeric values of these attributes can be used for selecting the best resource provider among the available ones.

The `Submitter` puts the `submittedTask` into the `TaskBus` and instructs the `Auctioneer` to conduct an auction to find the best resource suitable for executing the `submittedTask`. The `Auctioneer` initiates the auction by asking `MessengerComponent` to send a `taskSubmissionNotification`. The notification travels in the TCP/IP network and reaches all the clients acting as resource provider and currently available. The `MessengerComponent` on remote resource providers receives the notification. On receipt of the notification, the `MessengerComponent` informs its `Executor`.

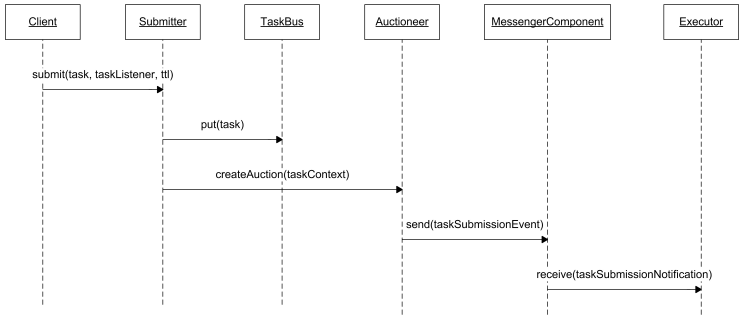


Figure 3.4: Interaction diagram for task submission.

3.6.2 Bid Submission

A client acting as a resource provider has a set of `ContextMonitor`s, both dynamic and static. These `ContextMonitor`s describe the resource capabilities in terms of static and dynamic resource information. For example, a `CPUContextMonitor` can provide static information such as "CPU Clock Frequency", "CPU Model", "L1 Cache", "L2 Cache". The `CPUContextMonitor` can also provide dynamic information such as the average CPU load during the last one, five and fifteen minute periods as "CPU Load (1 Min)", "CPU Load

(5 Min)", and "CPU Load (15 Min)" respectively. The `ContextMonitor`s registers them self with a `ContextIntepreter` along with the information about Monitored Entity that they are monitoring. The information provided by `ContextMonitor`s are converted into `ContextStatement`s by the `ContextIntepreter` and are stored in a `KnowledgeBase` in a `ContextManager`. The `ContextManager` has a query interface that can be used to query required context information. The `ContextMonitor`s provide static context information about the monitored entity after their registration with the `ContextIntepreter`. The `ContextMonitor`s provide dynamic information either periodically or immediately once the context of the monitored entity changes. For example, the "CPU Load (15 Min)" information changes (i.e., gets updated) every 15 minutes. On the other hand, queue length of the `Executor` gets updated when ever a task gets scheduled for execution or a task completes its execution.

The interaction diagram in figure 3.5 illustrates bid submission at resource provider. The `MessengerComponent` of a resource provider, on receiving a `taskSubmissonNotification` informs the `Executor`. The `Executor` calls the method `isSameAs()` of `ResourceContext` with the `TaskContex` advertised in the notification. The `ResourceContext` queries the `ContextManager` to see if the resource provider has sufficient capabilities requested. If the resource provider has sufficient capabilities, then informs the `Executor` that it can participate in the bidding process. The `Executor` then instructs the `Bidder` to find out the `Bid` that need to be submitted. The `Bidder` based on the advertised bidding strategy queries the `ContextManager`. The bid needs to be submitted before the TTB expires.

The `Submitter` determines the value for TTB using congestion determination algorithm inspired by the TCP congestion control slow start approach (RFC 5681) Allman et al. [2009]. The algorithm starts with a default minimum threshold value for TTB and doubles every time when the submitter sees that no bids are available for a given task announcement until it reaches a maximum threshold value. Once it reaches the maximum threshold value, the submitter halves the value every time when it sees bids for a given task announcement until it reaches the minimum threshold value. Otherwise remains at the maximum threshold value. Thus the processor load at the client and the network load determines the threshold value. Currently, the minimum threshold value defaults to 150 milliseconds and the maximum threshold value defaults to 60000 milliseconds.

3.6.3 Winning / Loosing the Auction

The interaction diagram in figure 3.6 illustrates winning a auction process at resource consumer. The `MessengerComponent` of a resource consumer, on receiving a `bidSubmissionNotification` adds the submitted bid by calling `addBid()` method of `Auctioneer`. The `AuctionTimer` notifies the auctioneer, once the TTB expires. The `Auctioneer` calls `evaluateBids()` method of `BidEvaluator` object. The `BidEvaluator` object implements the resource selection algorithm to determine the winner. The default resource selection algorithm selects the

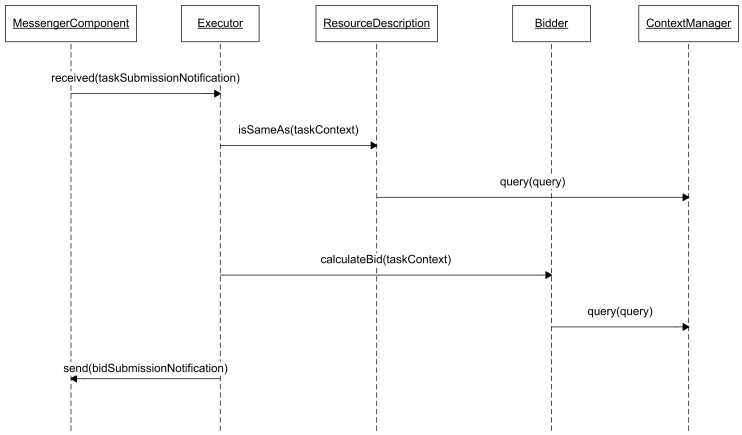


Figure 3.5: Interaction diagram for did submission.

client that has submitted the lowest bid. In case of a tie, two submitted bids having the lowest value, the winner is determined based on the submission time of the bids. The application developer is free to incorporate any task allocation algorithm suitable for his application requirement. Thus task allocation strategy can be application specific. Once the winner is decided the winning client is notified by the `AuctionListener` by calling `send()` method of the `MessengerComponent` with winner notification message as parameter. If the winning client does not receive the winner notification, then it has lost the auction. The clients do not maintain any state information and hence they do not wait for a winner notification and the messages are exchanged asynchronously.

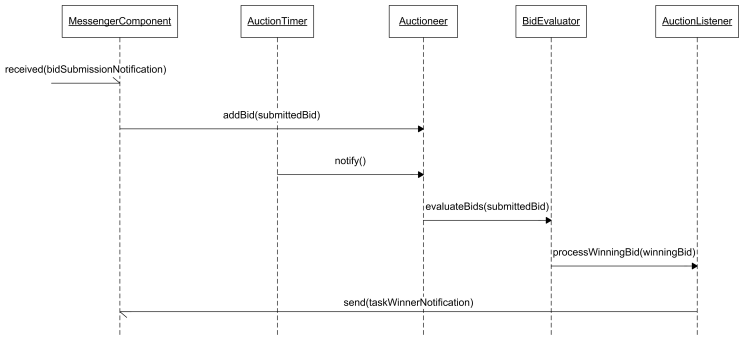


Figure 3.6: Interaction diagram for determining winner.

3.6.4 Remote Task Execution

The interaction diagram in figure 3.7 illustrates remote task execution at a winning resource provider. The `MessengerComponent` of the winning resource

provider, on receiving a `taskWinnerNotification` informs the `Executor`. Then it obtains the task from the `TaskBus` by calling the `getTask()` method and schedules the task on the `TaskExecutor`. The `TaskExecutor` on completion of task execution, informs the `TaskListener`. The `TaskListener` in turn informs the `Executor` about completion of task execution by calling the method `processTaskCompletion()`. The `Executor` gets the completed task from the `TaskExecutor` and puts the completed task into the `TaskBus`. The completed task contains the result of computation. Then the `Executor` instructs the `MessengerComponent` to send a `taskCompletionNotification`.

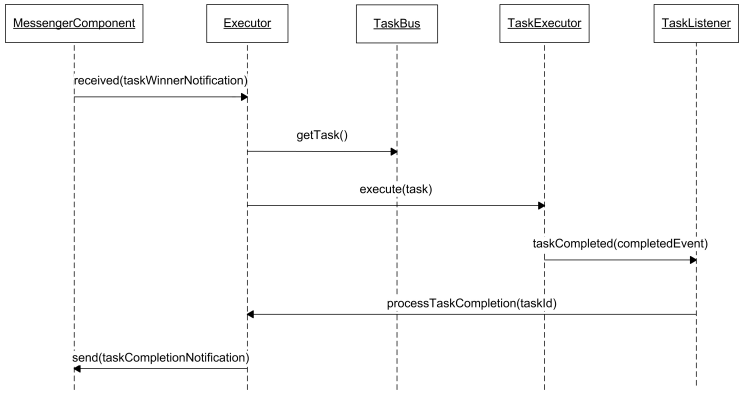


Figure 3.7: Interaction diagram for remote task execution.

3.6.5 Providing Results

The interaction diagram in figure 3.8 illustrates handling task completion and obtaining result at a resource consumer. The `MessengerComponent` of the resource consumer that submitted the task, informs the `Submitter` on receiving a `taskCompletionNotification`. Then the `Submitter` obtains the task from the `TaskBus` and informs the `TaskListener` about completion of task execution. The `TaskListener` can then get the completed task along with results.

In case, the remote resource provider could not complete the execution of the task, then the `Submitter` times out and informs the `TaskListener` about failure of task execution.

3.7 Summary

In this chapter we have presented the programming API and runtime infrastructure for formation of ad hoc Mini-Grids in a local area network. We have stated the motivation behind this work and the various design objectives for the framework. We have highlighted the use of various market based resource management approaches in the Grid. We have presented the need for modeling

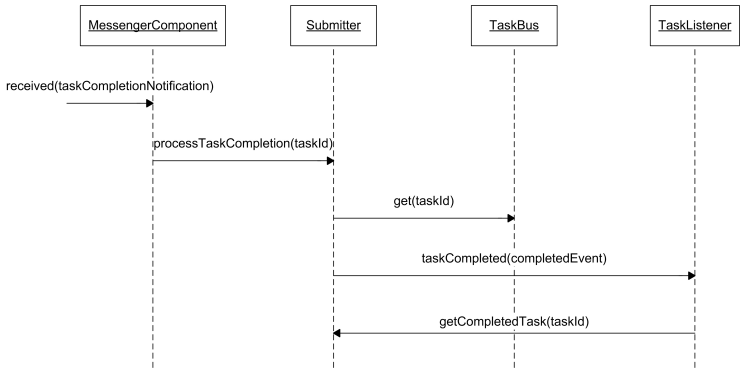


Figure 3.8: Interaction diagram for providing results.

computational tasks and resources. We have described in detail the software architecture of the Mini-Grid Framework. We shown the components of the resource provider and the resource consumer and how they interact. Currently, the communication between resource provider and resource consumer happen via serialized messages. However, it is quite easy to know what information is exchanged between them. Hence, encryption technique should be used ensure security but current implementation does not use any encryption technique.

Chapter 4

Context-Awareness for Quality of Service

Quality of Service (QoS) support refers to the possibility of a resource provider to offer a performance level on its computational capabilities that satisfies the requirements of a resource consumer. Such offers can be provided as two types of guarantees: *strict* guarantees, involving digital contracts and *loose*, best-effort usage. Strict guarantees require the establishment of a formal agreement, named *Service Level Agreement* (SLA) and *Service Level Specification (SLS)*, a set of attributes and values describing the profile requested performance level via signaling and negotiation between resource providers and resource consumers. In peer-to-peer grids - without a centralized administrative infrastructure - the implementation of QoS negotiation is difficult.

Loose guarantees are delivered on a best-effort basis, and for this reason, they do not require to prior establishment of a SLA and the consequent negotiation. The provisioning of loose guarantees consists of the capability of resource consumers to select service providers that provide a best-effort QoS profile that meets the resource consumers. In this case, resource discovery and selection depends on critical performance parameters over time.

The delivery of loose guarantees relies on three functional components.

- Availability of resource discovery mechanism that allow resource consumer to select the most appropriate resource provider by comparing different QoS profile of interest to resource consumer.
- Availability of gathered sensor data to resource consumers for making effective decisions about suitability of available resources.
- Availability of sensors for the monitoring of the QoS performance profile (SLS) of various Grid resources.

In Grid, resource provisioning can be implemented using either "resource-push" or "resource-pull" model. In resource-push model, a "resource manager"

or "scheduler" selects the best resource provider to run the tasks on behalf of a resource consumer, for example, the condor match maker Frey et al. [2002]. On the other hand, in the resource-pull model, resource providers periodically request the tasks from the resource manager or scheduler, for example, the BOINC scheduling server Anderson [2004]. In the resource-push model, to make appropriate decisions, schedulers need accurate information in real time about resource providers of interest. This information can be made available using either "information-push" or "information-pull" model. In the information-push model, information is send periodically (adopted, for instance, by the Globus Toolkit MDS Laszewski et al. [1997]). On the other hand in the information-pull model, information is collected from resource providers on demand (adopted, for instance, by the Legion resource management system Chapin et al. [1999]). The information-push model may cause schedulers to use stale (i.e. inaccurate) information.

Different types of resources can provide similar capabilities but with varying degrees of quality of service. Hence, the resource capabilities are required to be presented in such a way that resource providers can evaluate their capabilities against the requested capabilities for task execution, and the resource consumers can find optimal resources. A powerful discovery mechanism can be built if resource's capabilities and requirements are described explicitly, precisely, and unambiguously.

In existing Grid infrastructures, the capabilities of Grid resources can be obtained through Grid information services (such as the Monitoring and Discovery Service von Laszewski et al. [1997]). Currently, attribute-based resource description is used in Grid middlewares, for example Condor uses symmetric description language class advertisements (ClassAds) Raman et al. [2004]. However, these resource description mechanisms have several short comings: lack of expressiveness, extensibility, and symmetric nature. Attribute-based resource description mechanisms are symmetric, that is, both resource information providers and resource information consumers need to agree on syntax and semantics of language used for describing the resources. Thus they lack independence between the resource provider and resource consumer for expressiveness. Flexibility in the resource description framework is needed that allows to add new types of resources on the fly. It is necessary to have a meta-language that allows encoding relationship between existing resource types and new types of resource that will be added in future. Resource requirements need richer query interfaces to accurately describe what they want to discover.

Schedulers can obtain, at best, capability information such as the size of main memory, number of CPU's, and their nominal speed from information service like MDS Tsouloupas and Dikaiakos [2006]. However, application-specific characterization of Grid resources improves resource selection. In ad hoc grid environments, like in Mini-Grid, it is important to have a mechanism that allows the resource consumer to scope the resource selection based on the quality of offer and consume the resource capability that best matches their application requirements. Therefore, characterization of available resources for improving the resource selection is needed. In this thesis, the use of context information

for modeling the computational task and the resources is proposed.

4.1 Context-Awareness for Task and Resource Modeling

The coordinated resource sharing supplies a "context" that can be used to describe resources and requests. The context information, any information that can be used to characterize Grid entities, when modeled and managed introduces context-awareness into the Grid Jean et al. [2004]. This introduction of context-awareness into the Mini-Grid Framework enables provision of context-aware quality of service to the Mini-Grid applications.

4.1.1 Context Definition and Classification

In his survey, Dey Dey [2001] presents alternative views on context and its definition. Addressing the limitations of early definitions of context, Dey provided the following general definition, which is perhaps now the most widely accepted:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between the user and the application, including the user and the applications themselves Dey [2001]".

This human computer interaction (HCI) centered definition clearly states that context is always bound to an entity and that information describing the situation of an entity is context. However, from our point of view, this definition has some significant shortcomings. Firstly, the context is not just data or information about an entity but it also includes knowledge. That is associating meaning with the data transforms the data into contextual information. This contextual information can be converted into knowledge by applying interpretation rules.

For instance, a term `Unix` can be defined in our context model as a class of `OS`, then one can extend the term `Unix` to `Linux`, and `Unix` to `IRIX`. When a task requires `IRIX` operating system as one of the parameters for execution environment, and a particular resource provider has `Unix` operating system. It could also become one of the potential candidate as a reasoner could infer that `IRIX` is a subconcept of `Unix`. Having such domain knowledge with the inherent relations between concepts can assist in resource matching process, so that queries can be properly interpreted according to their meanings.

Secondly, the definition states that the context is relevant to the interaction between the user and the application. However, the contexts exist independent of the interaction between the user and the application. For example, the context exists for a CPU of a computational resource independent of the user or the application interaction with the computational resource.

After Day, there have been multiple attempts to define context in the context-awareness research community. At the same time, the research on context definition's shifted focus from a user and his activities towards entities participating in these activities. For instance, Wei et al. Xing et al. [2006], define context as any information concerning user's mobile device and its capabilities. da Costa et al. da Costa et al. [2005] define context as a storage, network, power and memory parameter of user's devices. Chalmers Chalmers [2004] define context as information relevant to the user along the time scale; the context which is "now" and the "past" context. J. Strassner presents a more abstract and extensible definition of the context as follows: "The Context of an Entity is the collection of measured, and inferred knowledge that describe the state and environment in which an Entity exists or has existed". Strassner proposes an approach whereby the context of any entity is modeled as a collection of data, information, and knowledge resulting from gathering measurements of and reasoning about that entity. J. Strassner definition overcomes the limitations of Dey's definition and hence we hence adopt it.

We have used temporal properties of the context information to classify the context information(other types of context classification are also possible MA Razzaque [2005]). We classify context information as configured, measured, derived, and archived type. Configured context information is static and is valid until the life of the entity. Archived context information refers to historical context information useful in reasoning and predicting. Measured context information is more dynamic and their accuracy depends on the measurement technique used. Derived context information is dynamic and their accuracy depends on the base information from which it is derived. Table 4.1 summarizes the various forms of context information and their features.

4.1.2 Context Modeling Techniques

Context modeling refers to structuring contextual information contained in the system in an abstract form on the level of data structure as well as the semantic level. There are several approaches that differ in complexity of syntax and expressiveness to model context Strang and Linnhoff-Popien [2004]. The most simple and widely used data structure for modeling contextual information is *key-value* pairs. Key-value pairs Schilit et al. [1994] are easy to manage; however, they do not support hierarchies/namespaces for separating identically named keys used in different context components. Further, queries only based on exact match are possible and they do not support sophisticated reasoning technique in context retrieval Strang and Linnhoff-Popien [2004]. Markup languages, for example XML, are characterized by a hierarchical data structure using a combination of tags with attributes and content. Context is modeled as a set of schema. Context modeling based on markup languages Licia et al. [2001] are highly expressive but often are proprietary or limited to a small set of contextual aspects or both. Graphical context models are derived from generic modeling methods such as UML Bauer [2003] and ORM Karen et al. [2003]. However, they do not support validations and lack formality. Object-

Type	Source	Nature	Accuracy	Example
Configured	Manufacturer, Administrator, and User	Static	More accurate but subjected to human errors	CPU vendor information
Measured	Benchmark tools	Dynamic	Depends on tool	CPU load
Derived	Inference tools	Dynamic	Depends on used inference technique	CPU usage pattern
Archived	Historical	Static	More accurate	CPU utilization

Table 4.1: Types of context information and their properties

oriented context model use object-oriented mechanisms such as encapsulation, inheritance and re usability to represent contextual knowledge Cheverst et al. [1999]. However, object-oriented models require low-level implementation agreement between applications to ensure interoperability and are thus not suited for knowledge sharing in open and dynamic environments Chen et al. [2004]. In a logic-based model, the context is defined as set of facts, expressions, and rules. Their level of formality is very high and they can be composed and distributed.

Ontologies are a powerful tool to specify concepts and the relationship between them. They provide a uniform way for specifying the context model concepts, subconcepts, relations, properties and facts, altogether providing the means for sharing of contextual knowledge and reuse. The contextual knowledge can be interpreted and evaluated by using ontology reasoning. Ontology reasoning allows software programs to compare contextual facts and infer new context information from existing contextual facts. Thus, We have adopted an ontology-based formal context model to address critical issues including formal context representation, knowledge sharing, and logic-based context reasoning.

4.1.3 Context Modeling Using Ontology

In the context of knowledge management, ontology is referred as a specification of a conceptualization Tim et al. [2001]. The ontologies are used to express the more or less complete knowledge about concepts, and their attributes, as well as their interrelationships. There exist several languages that are used to describe ontologies in recent years. These languages should be capable of describing concepts, attributes, and relations in a precise and traceable manner. Further,

it should be capable of creating effective queries towards reasoning. OWL L. and van Harmelen Frank [2004], which is part of the semantic web's ontology language based on XML and RDF/S McB [2004] has the above property. Further, wide range of tools are available for creating and validating ontology fragments developed in OWL. Also logical reasoning mechanisms can be used to deduce high-level conceptual context from low-level raw context. Research on ontology-based context models, which are able to share context information and reason context by defining contexts using these ontology languages, have been conducted Wang et al. [2004]; de Almeida Damiao Ribeiro et al. [2006]; Pessoa et al. [2007]. Currently, a number of efforts have focused on ontology in Grid context Brooke et al. [2004]; Tangmunarunkit et al. [2003]; Pinar et al. [2006]. Most ontologies proposed so far, are application-specific Grid environment, and have been developed for special purposes, for example as in Earth Grid System II Pouchard et al. [2003]. However, the definition of an upper-level domain ontology, Core Grid Ontology (CGO), for Grid-infrastructure-related information has been presented in Xing et al. [2006].

Ontologies are asymmetrically extensible (i.e., resource providers and consumers need not have to agree on certain terminology while extending the ontological concepts), resource providers can extend them without loosing the semantic soundness. For instance, a term `Unix` can be defined in our context model as a class of `OS`, then one can extend the term `Unix` to `Linux` and `Linux` to `ScientificLinux`. A reasoner can infer that `ScientificLinux` is a specialization of `Linux` and `Unix`. In the Description Logics, the fundamental reasoning of `concept expression` is a subsumption, which checks whether one concept is a subset (or superset) of an other concept.

Ontologies provide structured and extensible vocabularies that demonstrate the relationships between different terms allowing intelligent agents to flexibly and unambiguously interpret their semantics. For example, a resource context ontology might include the information that the terms `Pentium` and `Celeron` are `Intel` Processors, that `Intel` is not `AMD` or `SPARC`, and that an `Intel` system includes a `Pentium` or `Celeron` processor. This information allows the term "Computer with Pentium or Celeron Processor" to be unambiguously interpreted (e.g. by a resource consumer) as a specialization of `Intel` System.

In this thesis, we have adopted ontology-based semantic modeling of Grid resources on the basis of context, similar to many existing initiatives such as Core Grid Ontology. However, we have adapted scenario-based ontology development and evaluated the adequacy of the resulting ontology.

4.2 Ontology Design

Any proposal for a new ontology or the extension of an existing ontology must describe the motivating scenario, and the set of intended solutions to the problems presented in the scenario that arise in the applications. Then competency questions are used for the evaluation of a new ontology or the extension of an existing ontology M. and M. [1995]. Here, we present the Mini-Grid ontology by

developing concrete concepts to model the computational task and the resources in Mini-Grid.

The goal of this ontology is to present resource capabilities such that resource providers can match them against the requested capability, and the resource consumers can rank the resource providers in order to select the best candidate. The scope of this ontology is limited to its use in Mini-Grid. The level of granularity is directly related to the competency questions and terms identified.

Fundamental Grid domain concepts, vocabularies, and relationships have already been defined in Core Grid Ontology (CGO) and Grid Laboratory Uniform Environment (GLUE) schema Andreozzi et al. [2009]. The ontology presented here has been derived from the above generic domain ontologies. Only relevant concepts from these generic domain ontology have been reused, for example, "Processor" concept and more details specific to the Mini-Grid application have been added such as "Security Tool" concept.

4.2.1 Motivation Scenarios

The motivating scenario is a detailed narrative about a situation in the domain based on selected problems that need to be addressed. We have presented the scenarios using the templates proposed in Graciela et al. [2006], similar to use case templates in object-oriented methodology. The template describes: the name of the scenario, actors who participate in the scenario, a brief scenario description, and a list of possible terms related to the scenario. The motivation scenarios tabulated in Table 4.2 show the optimization criteria and resource requirements when the Mini-Grid users schedule their bioinformatics algorithm in Mini-Grid. These scenarios have been identified by interviewing the users and brain storming among project members.

4.2.2 Competence Questions

Competence questions derived from the motivation scenarios determines the scope of the ontology. The competence questions can also be used to verify if the ontology contains enough information to answer these questions. It also determines the level of granularity required in the ontology. A concept in the domain ontologies stated earlier is not included in this ontology if there is no competency question that needs it. Thus, the competence questions determines not only the concept included in this ontology but also the concepts that are excluded. And the same rule applies to properties in the ontology.

The competence questions states the problems that arise from the scenarios. Besides, scenario analysis contains a set of solution to these problems. Thus, the scenarios lead to primary competence questions and analysis of scenario leads to ancillary questions. In Table 4.3, we have presented a list of competence questions that were driven by scenario analysis. Let us discuss each scenario and possible solution to address the scenario. **Response Time:** In general users wish to use time optimization in scheduling of task in Mini-Grid, that is, the response time should be minimum. The response time depends on the

Scenario	Actors	Description	Terms
Time optimization	Mini-Grid application, `Task`, `Submitter`, `BidEvaluator`, and resource providers	The Mini-Grid application acting on behalf of a user submits a high computational task to the `Submitter`. The `Submitter` schedules the task, based on auction process, on the resource that is suitable to execute the job. Possible criteria used by the `BidEvaluator` are resource availability, the current load of the resource, queue lengths, and computational capacity of the resource.	Availability, Task queue, Queue length, Computational capacity, Resource load
Security	Mini-Grid application, `Task`, `Submitter`, `BidEvaluator`, and resource providers	The Mini-Grid application acting on behalf of a user submits a task to the `Submitter` that need to run on a secured resources. The `Submitter` schedules the task, based on auction process, on the resource that is suitable to execute the job. Criteria used by the `BidEvaluator` are security measures provided by a resource.	Security measures
Location	Mini-Grid application, `Task`, `Submitter`, `BidEvaluator`, and resource providers	The Mini-Grid application acting on behalf of a user submits a task to the `Submitter` that needs to run on resources located in a particular location. The `Submitter` schedules the task, based on auction process, on the resource that is suitable to execute the job. Criteria used by the `BidEvaluator` is location of a resource.	Location
Stability	Mini-Grid application, `Task`, `Submitter`, `BidEvaluator`, and resource providers	The Mini-Grid application acting on behalf of a user submits a task to the `Submitter` that need to run on stable resources. The `Submitter` schedules the task, based on auction process, on the resource that is suitable to execute the job. Criteria used by the `BidEvaluator` is the stability of a resource.	Stability

Table 4.2: Context modeling scenarios

Competence questions	
What is the resource response time?	
Ancillary questions	How many cores does the resource have? What is the processing power of each CPU? What is the current and recent processor utilization? What is the current queue length of local resource management system? What is the current average task completion time? What is the current average transfer time of per unit data?
How secure is the resource?	
Ancillary questions	What kind of security tools are available? Is the resource secured from external / internal attacks? Can sensitive data be transferred securely? What kind of security policies are practiced?
Is the resource stable?	
Ancillary questions	What kind of resource it is? What kind of network connection does it has? Does the resource belong to individual or shared by a group?
Where is the resource located?	
Ancillary questions	Is the resource located in "Room 4D06"?

Table 4.3: Competence questions

Security risk	PKI	IPSec	SSL	Firewall	Biometric Auth	SSH	IDS	Digital certificates
Credential theft	✓		✓		✓			✓
Unauth. admin access	✓				✓	✓		✓
Network credential theft		✓	✓					✓
Data theft		✓	✓		✓			
Denial of service attack		✓		✓			✓	

Table 4.4: Individual security approaches and their use in dealing with security threats and risks

number of logical processors (or cores) a resource has, processor clock frequency, current processor load, number of tasks being executed on the resource, number of tasks waiting for execution, the time taken to transfer the task from resource consumer to resource provider and so on. Thus we have a list of ancillary questions that represent the solution domain.

Security: Various classes of security-related risks and threats need to be considered before scheduling a task on remote resource providers. The common risks or threats of concern to Mini-Grid users are: credential theft, data theft, and denial of service attack.

- Credential theft describes the type of threat in which an entity gains unauthorized access to the application, administration access and network devices.
- Data theft describes the type of risk in which sensitive data or information is accessible or modified by unauthorized entities.
- Denial of service attack refers to an attack in which the performance of underlying network or system degrades.

In recent years, a wide variety of industry standards and technologies have emerged to address the above said theft or attack. Current security approaches include: security standard and policies, and library and tools. Standards and policies are the international best practices and approaches proposed and followed by industry that when applied would be able to minimize or prevent specific security risks. Examples include IPSec, a security standard, dealing with IP network-level security using public-key cryptography. Libraries and tools include the technologies that would be able to provide protection against security risks. Examples include firewall or packet filter dealing with unauthorized communication.

Thus, a secure resource can be identified by quantifying the security measures implemented on the resource by its user. In Table 4.4, we have presented individual security approaches and their ability to address the theft or attacks identified earlier. Table 4.4 provides a list of security technology and standards that can address one or more security risks. A tick mark (✓) indicates that a particular technology or standard is applicable to, or effective, in handling the security risk. However, stated security technologies are by no means complete; they serve as examples of current technologies and standards. The identified security threats are specific to Mini-Grid and may need to include other types of security threats based on experience.

A resource can be said secure, if it has a set of security approaches that can address all the security theft or attacks. For example, if a resource (say R1) implements PKI and IPSec and when a resource consumer selects this resource for execution then it is secure because these two security technologies together address all the identified security threats. If the resource addresses all the security threats using a set of security technologies and in addition has additional security measures, it is said to be more secure. For example, a resource (say R2) implements security technologies PKI, IPSecm and SSL, then it is more secure than resource R1.

Location: User may want to schedule tasks on resources that are physically located at a particular place. For example, during interaction with users, one user said he was interested in scheduling the application only on resources in his lab.

Stability: By stability, the user means guaranteed long-term access to the resource. In an ad hoc environment, it is not possible to guarantee such access to the resource. However, one can indirectly infer the stability of a resource by the device type and the type of network connectivity it has. Resources can be a 'server machine" or "shared desktop computers" or "user laptop computers" or "office desktop computers". Network connectivity can be of two types: wired and wireless. A device of type desktop computer and having wired network connectivity can be assumed to be available in Mini-Grid for longer duration than a device of type laptop and having a wireless network connectivity. Thus, by knowing the type of device and network connectivity, one can judge the availability of the resource.

4.2.3 Object Oriented Data Model

In this section, the core concepts of the proposed mini-gird ontology are presented. Though it has been designed to represent knowledge of Mini-Grid system, it can be extended. Therefore, we have adopted the Web Ontology Language OWL-DL L. and van Harmelen Frank [2004] to describe the terms and classes in the Mini-Grid ontology. We first define a set of core classes and then mention few sub-classes. After that, we define the relationships and constraints among these classes (represented by links).

In Figure 4.1, the core concepts of the proposed ontology are presented in the form of a UML class diagram. In the proposed context model, we view

Concept / Class	Description
Resource	An entity providing a capability
Software	One of the capabilities available in a resource
Middleware	A software component for forming distributed computing environment
Location	The geographical location of resources
Security Mechanism	Security policies and protocols implemented by a resource
Network Connectivity	Network connectivity that a resource has

Table 4.5: The core concepts / classes in the Mini-Grid context model.

Mini-Grid as a collection of entities and these entities are monitored to know their context and hence we have monitored entity as the base class. The monitored entities can be computational or storage devices. However, other types of devices can also participate in Mini-Grid. Such devices can be sub classed from `MonitoredEntity`. It has the attribute's entityId, which is a unique identifier that identifies the entity being monitored, and entityType, which is the type of the entity being monitored.

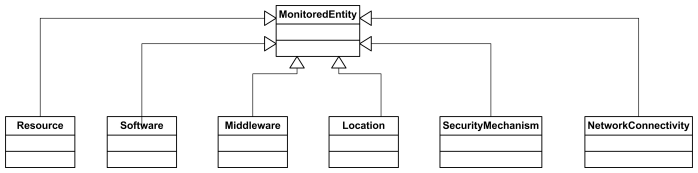


Figure 4.1: An UML diagram of core ontology concepts.

In Table 4.5, the core concepts classes of the Mini-Grid context model are presented. Out of these concepts, software and security mechanism needs more explanation.

- Software: Refers to softwares that are currently installed in a resource. We consider availability of a software as one of the capabilities of a resource. Software can be Mini-Grid application, a user library, a security software installed on the resource, and so on.
- Security Mechanism: Refers to security policies and protocols adopted by a resource. Resources can adhere to a specific security policy that can address a security risk or threat. Resources can implement certain security protocols such as IPSec, Kerberos, and SSH. Ability to address a security threat or risk can be considered as one of the capabilities of the resource.

4.3 Context Modeling

Based on the scenarios detailed earlier and the competence questions, we have identified two main abstractions: tasks and resources and a range of user preferences. In this section, we present the task and context modeling based on their context. We also present how the framework performs matching the capability requirements of the Mini-Grid application to the capabilities of the available resource providers.

4.3.1 Resource Modeling

In Figure 4.2, we present the conceptual context model of the computing device (i.e., resource providers). The computing device is based on the core monitored entity `Resource`. It has various software entities including the Mini-Grid application and the Mini-Grid middleware, network connectivity, and an execution environment. The computing device can implement certain security protocols and security policies, and is located in a particular physical location.

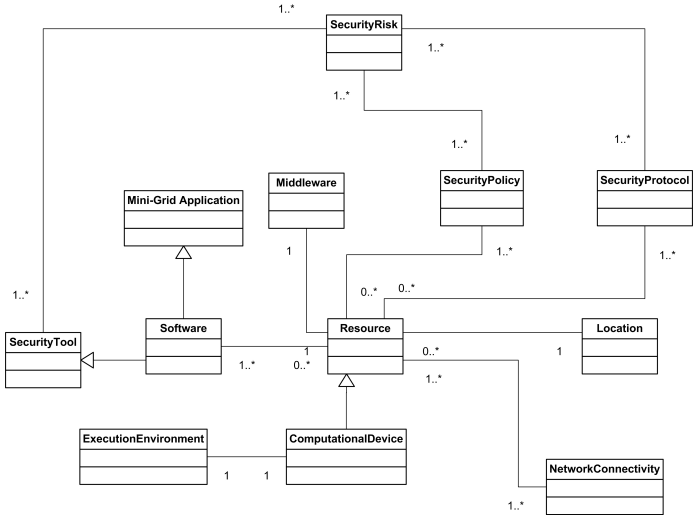


Figure 4.2: Concepts and relationships for resource.

In Figure 4.3, we present the conceptual context model of the computing environment. It uses `Processor, Memory, DiskSpace, OperatingSystem,` and `TaskExecutor` concepts. The properties of these concepts are tabulated in Table 4.6.

4.3.2 Task Modeling

In figure 4.4, we present the conceptual context model of the computational task. A Mini-Grid application can contain one or more computational tasks. A

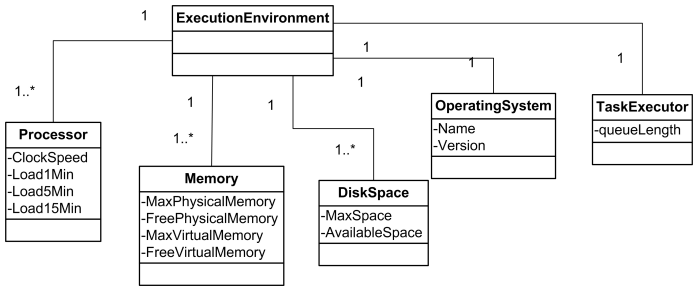


Figure 4.3: Concepts and relationships for computing environment.

Concept	Property	Description
Processor	ClockSpeed	CPU clock frequency
	Load1Min	1-minute average processor load
	Load5Min	5-minute average processor load
	Load15Min	15-Minute average processor load
Memory	MaxPhysicalMemory	Total physical memory
	FreePhysicalMemory	Unallocated physical memory
	MaxVirtualMemory	Size of configured virtual memory
	FreeVirtualMemory	Available virtual memory
DiskSpace	MaxSpace	Size of configured disk space
	AvailableSpace	Available disk space
OperatingSystem	Name	Name of the operating system
	Version	Operating system version
TaskExecutor	QueueLength	Number of tasks in the queue

Table 4.6: The properties of concepts in computing environment.

computational task has an execution environment, a computational complexity index value, requires certain Mini-Grid middleware configurations, and uses certain bidding strategy for scheduling.

4.3.3 QoS Modeling

QoS requirements of an application can be modeled as bids in the auction process. For example, consider a scenario where the application wants to schedule tasks on resource that can execute faster than others. In this case, the application developer has to give a name to the strategy that need to be used, for example `SPEEDBID`. During task announcement, the resource consumer would announce that it would like to have the bidders bid using ‘ SpeedBid ’. Further, the application developer has to define what speed bid means. Currently, looking at our context model we have the information about CPU clock frequency, current cpu load, and queue length at the `Executor`. Using these information we formulate the speed bid. Using CPU clock speed and the name of the proces-

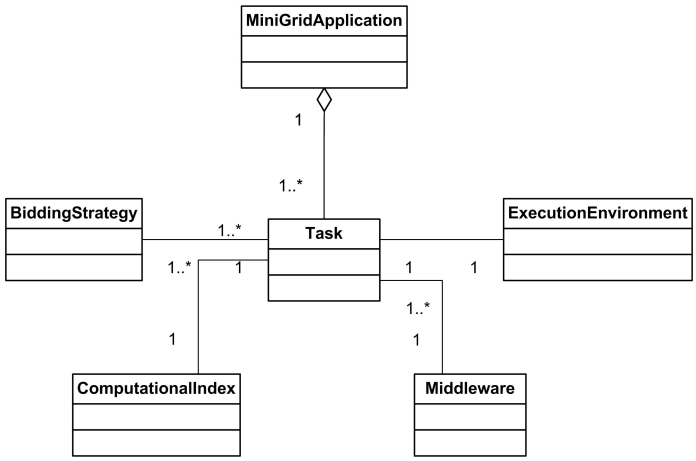


Figure 4.4: Concepts and relationships for a task.

sor, we can find the average MIPS rating of the processor. The average MIPS ratings for different processor have been published in different web-sites, for example Wikipedi[1] or we can use bench mark tools, for example Open Source Open Benchmark (OCB)[2] to determine this. The MIPS rating for a processor is determined from a set of small programs known as the MIPS benchmark. The ration of the geometric mean of the performance of these programs to that for the same programs run on the VAX 11/780 provides the MIPS rating. The VAX 11/780 is defined as a MIPS-1 machine, meaning that it is capable of executing one million instructions per second. If a processor is rated at 10 MIPS, it supposedly performs the given benchmarks ten times after does than a VAX 11/780.

Now, we define a scoring function that ranks the resources based on average MIPS rating, current cpu load and number of processors.

- CPU Speed (MIPS) - MIPS rating per CPU
- CPU Count (N) - Number of CPUs available
- Current Load (L) - The machine utilization information at the time of scheduling. For example, if current cpu load is 10%, then this parameter will have a value 0.1.

$$Rank = MIPS \times N \times (1 - L) \tag{4.1}$$

In Mini-Grid, tasks can be queued at the remote executor for execution. The above ranking function in Equation 4.1 does not incorporate this information.

[1] http://en.wikipedia.org/wiki/List_of_Intel_microprocessors
[2] http://sourceforge.net/projects/opencpubench/

Further, in Mini-Grid, the average completion time of an application is not known at the time of scheduling. Hence, we follow a greedy approach that does a simple ranking using the Equation 4.2.

$$Rank = \frac{MIPS \times N \times (1 - L)}{Q} \tag{4.2}$$

where `Q` denotes the queue length at the `Executor`. The ranking function used is simple but useful. However, more accurate ranking functions can be developed using CPU load prediction techniques Zhang et al. [2008]. The bidder component of the resource provider has to query its local context manager using a query template to compute the bid. Hence, the application developer need to define the query template name and its corresponding SPARQL query as shown in below code snippet. On executing the query, the context manager returns the value of the required parameters. Using these parameters, the bidder would compute the bid that need to be submitted.

```
PREFIX  mg:   <http://minigrid/elements/1.0/#>

SELECT  ?speed ?cpuload ?qlength
WHERE {
  ?node          mg:deviceType        mg:ComputationalDevice ;
                 mg:hasProcessor      ?processor ;

  ?processor     mg:cpuSpeed          ?speed .
  ?processor     mg:cpu15Minload      ?cpuload .
  ?processor     mg:hasTaskExecutor   ?taskexecutor .
  ?taskexecutor mg:queueLength        ?qlength .
}
```

On receiving the bid, the auctioneer using `evaluateBids()`, determines the best bid. The evaluate bid method sorts the bids according to the Equation 4.2 and determines the best bid. Bids can be sorted as they are comparable objects.

User may be concerned not only about trunaround time but also about security issues. Some resource consumers may feel that it would be better to schedule the task on resources that are secure. For example, a user can say " How do I prevent my data leaving from the building? ". This translates into hoq to prevent data leaving from my network. A correctly configured firewall can prevent unauthorized traffic from entering your network and keep data from leaving your newtork. Finally, the user's concern get translated into ensuring that the resource has a firewall installed. Looking at the Table 4.4, one could see that firewall has been identified as one of the security approach. As detailed earlier, the application developer has to define a bid, its corresponding query, and evaluation approach.

4.3.4 Evaluation

In order to verify and validate the ontology regards to competency questions, we used the Simple Protocol and RDF Query Language (SPARQL) Prud́hommeaux and Seaborne [2008]. SPARQL is a syntactically SQL-like language for querying RDF graphs via pattern matching. The language's features include basic conjunctive patterns, value filters, optional patterns, and pattern disjunction. To implement these queries, we used the Jena Framework, which provides an API for creating and manipulating the RDF model.

We present an example to illustrate how we have validated the ontology with respect to competency questions. Let us assume that a user has defined speed bid parameterized by the cpu clock frequency. His computational tasks require a memory of 4GB, then he can issue the following SPARQL to find suitable resource. We create the context model, and different instance using Jena API and then issue the query to see if the output matches our manual inference.

```
PREFIX  mg:   <http://minigrid/elements/1.0/#>

SELECT  ?speed
WHERE {
  ?node       mg:deviceType        mg:ComputationalDevice ;
              mg:hasProcessor      ?processor ;
              mg:hasMemory         ?memory .
  ?processor  mg:cpuSpeed          ?speed .
  ?memory     mg:maxPhysicalMemory ?phyMemory .

  FILTER ( ?phyMemory > 4096 )

}
```

4.4 Context-awareness Sub-system

In the next section, we present the design of the context-awareness sub-system inspired from the Java Context Awareness Framework (JCAF) Bardram [2005a,b,c]. JCAF is a Java-based service oriented runtime infrastructure and programming API for creation of context-aware applications. The runtime infrastructure is an environment of distributed services for acquiring context information (through Context Monitors and Context Actuators) and which enables interested application components to subscribe to relevant context events through an event-based mechanism. Context transformers reside in a transformer repository and constitute application-specific implementations of aggregators of context information.

The JCAF application programming interface enables modeling context of entities as a set of context items encapsulating context information. Entities can themselves be context items of other entities and thus maintains relations among entities. However, JCAF uses RMI for communication between entities.

In general, Java RMI has been designed for creating distributed client/server application. The communication between services and clients as well as among the clients themselves is based on Java RMI. Java RMI is acceptable when transfering small or medium sized chunks of context data over hig-speed data links but being slower than TCP. However, when transferring larger chunks of data the RMI performance degrades Alisa [2010]. For comparison, RMI needs 5.1 ms to transfer 10 B of data versus 165.7 ms to transfer 50 KB Gümüskaya et al. [2008]. Further, JCAF does not have a context reasoning mechanism.

We have adopted the idea of context acquisition using context monitors in our framework. We have also adopted the event-based architecture of JCAF. However, we have used reasoning technique to acquire knowledge from existing context information. Further, we have ontology for context modeling and RDF store for context management.

4.4.1 Conceptual layers

Implementing context management system using modular approach contributes to the system being loosely coupled and ensures component reuse and extension, without incurring unnecessary development time. The architecture of context management system based on layered design approach is illustrated in Figure 4.5.

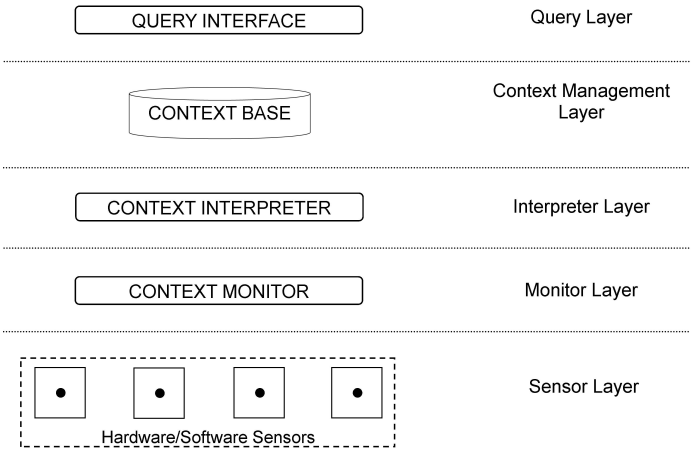


Figure 4.5: Conceptual layers of context management system

Sensor Layer: This layer consists of physical and/or logical sensors. They provide context information in device-dependent raw format. Proprietary protocols may be needed for interacting with sensors. Physical sensors are subjected to failure and hence unreliable.

Monitor Layer: This layer acts as a wrapper to sensor layer. It may be embedded into the sensors or can be a separate component. If implemented as a separate component, then reliability is ensured. It also provides uniform access

to context information.

Interpreter Layer: This layer consists of context interpreters. These interpreters have the ability to convert context in one form into another. This layer ensures physical data independence.

Context Management Layer: This layer provides management functionalities. They provide interface to manage context information. Also, they provide means to define context models specific to an application. They can also aggregate context information.

Query Layer: This layer provides access to context information by context consumers, for example, resource provider. Context information can be obtained by polling technique. This layers consists of query interface explained earlier and ensures logical data independence.

4.4.2 Context Management System - Modular View

An UML diagram of the core concepts/classes in context management is shown in Figure 4.6. The core software component is the ContextManager implementing the functionality of a context management system. The ContextManager is responsible for context retrieval and dissemination, structured storage of context information, and acquiring and transforming the context information. The ContextManager has the following sub-components: ContextMonitors, ContextInterpreter, KnowledgeBase, and QueryInterface as shown in Figure 4.6.

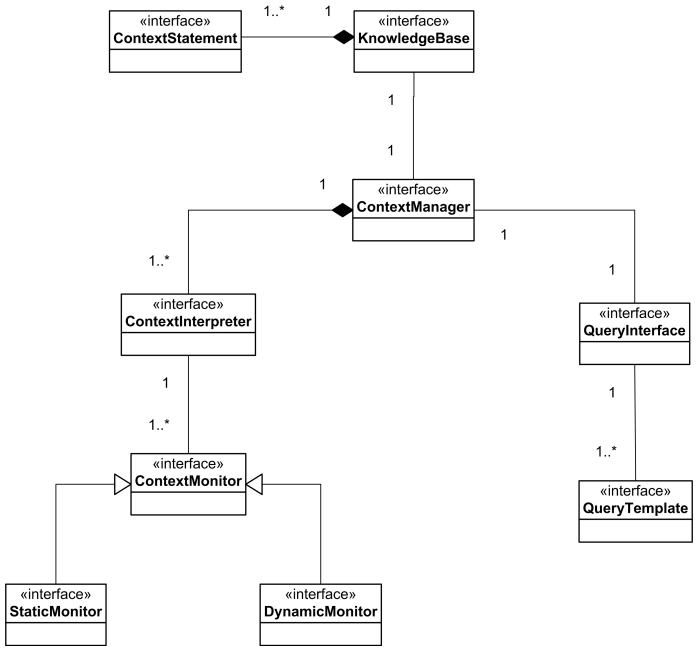


Figure 4.6: Module view - Context management system

ContextMonitors: The context information is mostly distributed over different physical devices or embedded in the application environment. Thus, sensors are the source of context information and a context monitor can receive all data in raw format and delivers the necessary information to context interpreters. Context monitors and sensors can communicate using proprietary protocol and data format. For example,as illustrated in Figure 4.7 a desktop computer is monitored by a CPU context monitor to provide context information about processor. The desktop computer – a monitored entity – has an SNMP agent, a software sensor, which provides information such as softwares installed and the processor details. The CPU context monitor and the SNMP agent can communicate using the SNMP protocol J.D. et al. [1990]. Each context monitor has a unique identifier.

Figure 4.7: An example context monitor

ContextInterpreters: A context monitor can register and unregister them-self with a context interpreter. While registering with the context interpreter, they provide information about the entity they are monitoring. Each entity observed by the context management system has a unique identifier. Apart from the unique identifier, each entity can be identified to one of the entity types. For example, a CPU context monitor can provide context information such as current CPU load to an RDF context interpreter. In this case, the CPU context monitor registers itself with the RDF context interpreter that it is providing context information about processor. In this case, the monitored entity would be "processor-01" and type of entity would be "Processor". Context monitors provide information as context statement. For example, to inform the context interpreter that the processor has a speed of 2194 MHz, it uses the key-value pair {"processor-speed", "2194"}. The RDF context interpreter interprets this information as a context statement with subject "processor-01", predicate "processor-speed", and object "2194" as shown in the Figure 4.8. The context consumer application needs to define mapping functions that can convert the context information in raw format into a specific format. For example, in the above case we need to provide the information that the key "processor-speed" needs to be mapped to RDF property.

KnowledgeBase: The Knowledge base contains data store to store context information about different monitored entities. The context-aware application need to define a context model to describe a set of monitored objects. For example, an ontology can be used to define the properties of a processor and an RDF store to store the ontology and its instance.

The below code snippet shows processor ontology in RDF/Turtle format.

```
<http:// ..../processor.owl> rdf:type owl:Ontology .
```

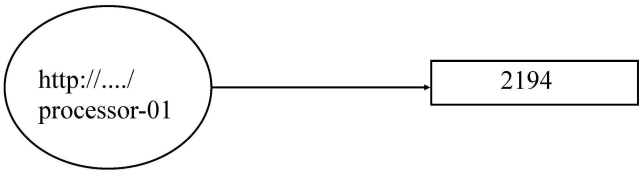


Figure 4.8: RDF graph representation of context information

```
:Processor rdf:type owl:class .
```

The context base provides functionalities to add and remove context information. The context base can provide both persistence and in-memory storage.

Query interface: A context consumers can use the query interface to query context information from the context base. The query interface provides support to specify query template once and then to use that template multiple times. For example, if we have a SPARQL query interface, then the context-aware application developer can define SPARQL queries for anticipated situation in the application. The templates are associated with a name and this name can be used to query rather than the actual query statement. In the Mini-Grid Framework, `Executor` and `ExecutorContext` act as context consumers and queries the context base.

QueryTemplate: A query template is an encapsulation which includes user-application quality of service description or resource capability description and their mapping to the corresponding SPARQL query statement. A template allows a domain expert to express domain-specific quality and resource capability description and corresponding mapping to the SPARQL query. Among the available templates, the application developer can choose templates relevant to the application.

4.4.3 Context Management System - Dynamic View.

Defining Query Template: An executor can associate a specific query template name to a query representing an application-specific bidding strategy by calling the `addTemplate()` method in `QueryInterface`.

Consuming Context Information: The interaction diagram in Figure 4.9 illustrates the consumption of context information by an executor. Once the executor receives a bid submission notification, it queries the query interface for context information with corresponding query template name based on the type of bidding strategy. The query interface would use the corresponding query to query the knowledge base. The knowledge base returns a set of context statements that matches the query. The executor uses these context statements to calculate the bid.

Providing Context Information: The interaction diagram in Figure 4.9 illustrates the provision of context information by a context monitor. Initially a

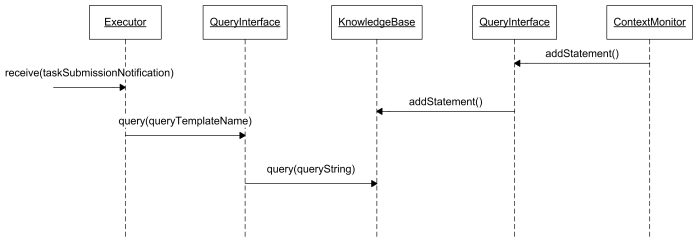


Figure 4.9: Context information consumption at an executor.

context monitor registers itself with a context interpreter. When the context of a monitored entity changes, the context monitor notifies the context interpreter by calling `addStatement()` method of context interpreter. The context interpreter in turn calls the `addStatement()` method of the knowledge base and the context statement gets added to the knowledge base.

4.5 Summary

Support of Quality of Service (QoS) in desktop Grid environment requires unambiguous description of resource capabilities available and required. We have presented the various context modeling technique and their advantages and disadvantages. In this chapter, we have presented the use of context information for unambiguous description of resource capabilities. we have adapted scenario-based ontology development and evaluated the adequacy of the resulting ontology.

Chapter 5

Programming API

In this chapter, we provide some insights in the use of the Mini-Grid framework in developing a sample Mini-Grid application. The Mini-Grid Framework has been developed i) to support computational task distribution to participating resources in Mini-Grid, ii) to model capabilities of participating resources and capability requirements of a Mini-Grid application, iii) to extend the framework with application specific quality of service parameters for scheduling computational tasks, and iv) to extend the framework to develop application specific auctioning protocols.

The framework aims to provide a general and expendable functionalities stated above. The framework has been implemented as Java classes and interfaces that can be used in building Mini-Grid applications. The framework target at three different users i) Mini-Grid application developer, ii) application toolkit developer (grid enabling existing application toolkit) and iii) domain expert to implement application specific auction protocols and bids.

5.1 Sample Application

One of the commonly used parallel programming model is the data parallel model. In this model, each computational task executes the same piece of code but on a different part of the data. This involves splitting of application data among available computational resources. The data set is typically organized into common data structure such as an array Barney. Figure 5.1 shows an example scenario where the data set is stored in an array (say A). The parallel version contains three tasks each executes same code but on different parts of the data.

Our sample application finds all the prime numbers from 1 to an upper limit, say `UPPERLIMIT` as shown in the below code snippet. The `UPPERLIMIT` is an input to the program.

```
public ArrayList<Integer> calculatePrime(int UPPERLIMIT)
```

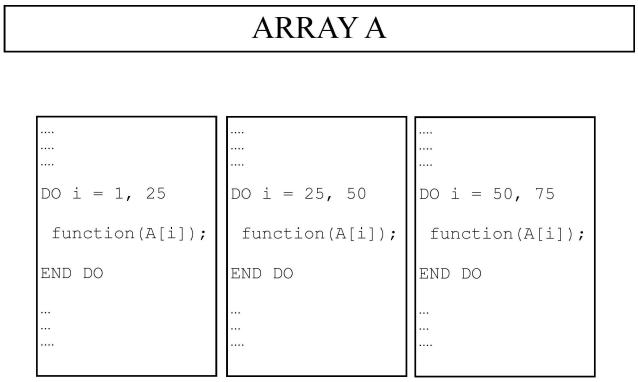


Figure 5.1: Data parallel model.

```
{
   ArrayList<Integr> primes = new ArrayList<Integer>();

   for (int i = 2; i < UPPERLIMIT; i++)
   {
    boolean prime = false;
    int limit = (int) Math.ceil(Math.sqrt(i));

    for (int j = 2; j < limit && prime==false; j++)
    {
         if (i%j == 0) prime = true;
      }

      if (prime)
         primes.add(i);
   }

   return primes;
}
```

Now, we develop a parallel version of the sample application using the data parallel model. The parallel version demonstrates a common technique used in parallel algorithm – solving a smaller case of the same problem to speed the solution of the full problem. An important aspect of developing a good parallel algorithm is designing one whose work is close to the time of a good sequential algorithm that solves the same problem. Without this condition we cannot hope to get good speedup. Parallel algorithms are referred to as work efficient relative to a sequential algorithm if their work is within a constant factor of the time of the sequential algorithm. Finding prime numbers applications aims to develop a

work-efficient algorithm Blelloch [1996]. The below code snippet illustrates the parallel version of the code. This program takes two input parameters LLIMIT and ULIMIT. Let us assume that we wish to calculate prime number between 1 to n. We split the computation into m tasks. Now, each task computes n/m numbers, i.e., first task computes in the range 1 to n/m, second task computes in the range n/m+1 to 2n/m and so on. The upper and lower limit of the range becomes input to the function `calculatePrime()`. The calculation in each task are independent of one another and hence leads to embarrassingly parallel situation.

```
public ArrayList<Integer> calculatePrime(int LLIMIT, int ULIMIT)
{
   ArrayList<Integr> primes = new ArrayList<Integer>();

   for (int i = LLIMIT; i < ULIMIT; i++)
   {
      boolean prime = false;
    int limit = (int) Math.ceil(Math.sqrt(i));

      for (int j = 2; j < limit && prime==false; j++)
      {
         if (i%j == 0) prime = true;
      }

      if (prime)
         primes.add(i);
   }

   return primes;
}
```

The Grid applications are either computationally intensive or data intensive. The computational intensive application uses most of the CPU power and use moderate amount of data. Here, we have used a simple application to demonstrate how a parallel version can be developed. Though it is simple, from parallelization point of view serves as a good example. Further, the Mini-Grid Framework targets at application that are computational intensive and that can be parallelized easily. However, any application that can be converted into a Bag-of-task application can be deployed in the Mini-Grid environment.

5.1.1 Grid Enabling Sample Application

This section illustrates how the sample application can be grid enabled using the Mini-Grid Framework. The core modeling abstractions in the Mini-Grid framework are the interfaces and classes: `Task, TaskContext, Submitter,`

and `TaskListener`. An UML diagram of the relationship between these classes and interfaces are shown in Figure 5.2.

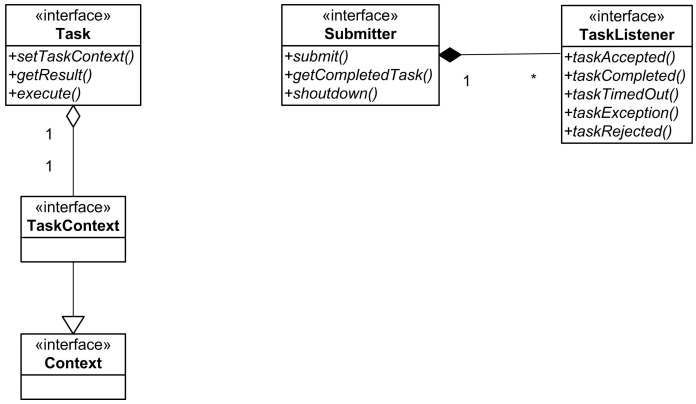


Figure 5.2: An UML Diagram of Task and Related Classes and Interfaces.

The basic modeling concept in the Mini-Grid framework is the `Task` interface. This interface defines the computational unit that need to be distributed in Mini-Grid. The parallel tasks in the sample application need to implement this interface. Further, the method `execute()` contains the business logic of the parallel tasks. A simple example implementation of a `PrimeTask` is listed below:

```
public void execute()
{
   for (int i = LLIMIT; i < ULIMIT; i++)
   {
    boolean prime = false;

      int limit = (int) Math.ceil(Math.sqrt(i));

         for(int j = 2; j < limit && prime == false; j++)
         {
            if (i%j == 0) prime = true;
         }

         if (prime)
            primes.add(i);
   }
}
```

In the above code snippet, the variables “LLIMIT” and “ULIMIT” are parameters to be the constructor. An instance variable “primes” holds the prime

numbers in the range LLIMIT to ULIMIT. Further, the task needs a task context as detailed in Section 5.2.2.

The interface `Task` encloses results of its execution at a remote resource provider. The application submitting `PrimeTask` to the Mini-Grid environment calls the method `getResults()` to get the results of execution. Hence the class `PrimeTask` need to implement the `getResults()` method and the below code snippet illustrates it.

```
public Object getResult()
{
    return primes;
}
```

The application need to be notified once the results are available. The framework uses the "Observer pattern" to notify the application important events during the execution of a task. The framework provides an interface `TaskListener` that acts as subject and needs to register with a `Submitter` for notifications. It has `taskAccepted(), taskCompleted(), taskRejected(), taskStarted()`, `taskTimedOut()`, and `taskException` methods. The `Submitter` is notified when a task gets accepted for execution, gets rejected, gets completed, generates exception during execution and times out on a remote resource provider. The application need to implement this interface and should have the logic for handling the appropriate situation.

The below code snippet illustrates a sample implementation of `TaskListener`. This task listener, logs in the various events that happen during the remote execution of a task.

```
public class SampleListener implements TaskListener
{
   private Submitter mgSubmitter;

   private LocalExecutor localExecutor;

   public SampleListener(Submitter submitter, LocalExecutor lEecutor)
   {
      mgSubmitter = submitter;
      localExecutor = lExecutor;
   }
   public void taskCompleted(TaskEvent event)
   {
      logger.info("Execution of task completed"+event.getTaskId);

      //Obtain completed task from Mini-Grid Submitter
      mgSubmitter.getCompletedTask(event.getTaskId());
```

```
      //Process completed task
   }

   public void taskStarted(TaskEvent event)
   {
      logger.info("Execution Started"+event.getTaskId());
   }

   ...

   public void taskRejected(TaskEvent event)
   {
      logger.warn("Task could not be scheduled in Mini-Grid"
                                              +event.getTaskId();

      localExecutor.schedule(event.getTaskId());
    }

    ...
}
```

Now, the sample application consists of a set of `PrimeTask` that can be distributed in Mini-Grid for execution. In the code snippet `LocalExecutor` corresponds to an object that is responsible for scheduling failed tasks on the resource consumer.

5.1.2 Distributing Tasks in the Mini-Grid Environment

Client acting as a resource consumer joins the Mini-Grid environment by creating objects of class implementing `Submitter`. The clients can leave the Mini-Grid environment by calling `shutdown()` method in `Submitter`. The sequence of steps involved in the resource consumer joining the Mini-Grid environment are:

- Create and initialize a transport component.

```
TransportComponent transportComponent =
           TransportComponent.getInstance();
transportComponent.init();
```

- Create and initialize messenger component.

```
MessengerComponent messengerComponent =
         MessengerComponent.getInstance(transportComponent);
messengerComponent.init();
```

- Create and initialize submitter object.

```
Submitter minigridSubmitter =
                new DistributedSubmitter(messengerComponent);
```

- Inform submitter the auction and evaluation method to be used.

```
minigridSubmitter.setAuctioneer(new FPSBAuctioneer
               (minigridSubmitter, new SimpleBidEvaluator()));
```

The sample application running on a resource consumer can submit a task by calling the method, `submit()` method. If the user has an idea on the life time of the individual task, then he can specify it as a parameter in `submit()` call, otherwise the framework uses a default value for TTL. In the code snippet, `FPSBAuctioneer` corresponds to the implementation of First-Price Sealed-Bid auctioneer.

```
LocalExecutor localExecutor = new LocalExecutor();

TaskListener taskListener
       = new SampleListener(minigridSubmitter,localSubmitter);

for (int i=0; i< UPPERLIMIT/nofTask; i++)
{
   Task tempTask = new PrimeTask(taskContext, i, i*UPPERLIMIT/nofTask);
   minigridSubmitter.submit(task, taskListener);
}
```

5.1.3 Providing Results

Once the task gets completed at remote resource provider and is ready for use by the sample application, the "DistributedSubmitter" object implementing `Submitter` interface notifies the application by calling `taskCompleted()` method of the class implementing the interface task listener (i.e., `TaskListener` object). The application calls `getCompletedTask()` method to obtain the results as shown in the below code snippet.

```
minigridSubmitter.getCompletedTask(taskIidentifier);
```

5.1.4 Participating in Mini-Grid - Resource Provider

A client willing to share its capabilities join the Mini-Grid environment and executed the task allocated to it by participating in a bidding process. The sequence of steps involved in the resource consumer joining the Mini-Grid environment are:

- Create and initialize a transport component.

```
TransportComponent transportComponent =
                          TransportComponent.getInstance();
transportComponent.init();
```

- Create and initialize messenger component.

```
MessengerComponent messengerComponent =
         MessangerComponent.getInstance(transportComponent);
messengerComponent.init();
```

- Create and initialize executor object.

```
Executor minigridExecutor =
               new DistributedExecutor(messengerComponent);
```

- Inform the executor how to obtain its resource context.

```
minigridExecutor.setContext(new DefaultExecuterContext());
```

- Set the different types of bidders the executor can use.

```
minigridExecutor.setBidder(new SpeedBidder());
```

After the initialization, the client receives task winner notification and can participate in the bidding process at detailed earlier in Section 3.6. The client can leave Mini-Grid as demonstrated in the following code snippet.

```
minigridExecutor.shutdown();
```

In this section, we started with a discussion on how an algorithm can be parallized using a simple application. Then it presents how a resource can participate in the Mini-Grid and distribute computational task in the Mini-Grid. In the next section we would look into implementation of some of the core abstractions used in the Mini-Grid Framework.

5.2 Defining Context

Resource providers and resource consumer need a context model to annotate their resources and to describe their requests. Furthermore, the `Bidder` component uses this context model to obtain information required for formulating a bid.

In order to explain how resource and task context can be stated in Mini-Grid environment, we use the following example context model illustrated in

Figure 5.3. The figure shows fragments of a ontology defined to annotate the resources in the Mini-Grid. The ontology consists of only core concepts, their properties and the relationship between them that are necessary for the discussion. The key top-level concept *ResourceType* consists of classes and properties that describe *Computing, Storage and Network* elements. The classes used to describe *ComputingDevice* include *Processor, Memory, DiskSpace, Software and TaskScheduler*.

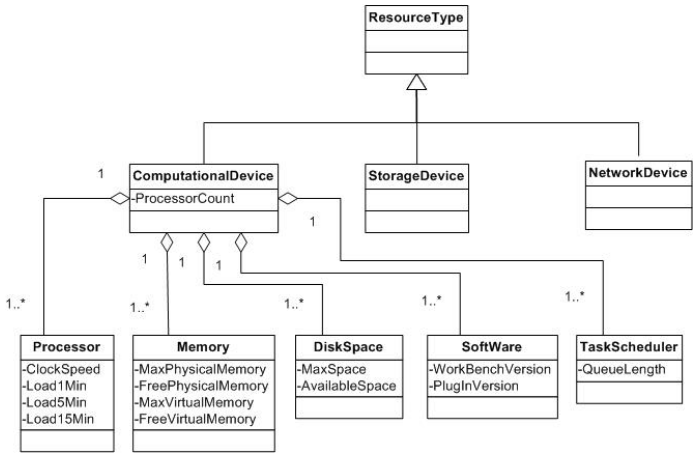


Figure 5.3: Fragments of different concepts used in Mini-Grid resource description using UML.

The Processor class has clock frequency (*ClockSpeed*), average processor load in 1, 5, and 15 minutes (*Load1Min, Load5Min, Load15Min*) as its properties. The Memory class has maximum physical memory (*MaxPhysicalMemory*), currently available physical memory (*FreePhysicalMemory*), maximum virtual memory (*MaxVirtualMemory*), and currently available virtual memory(*FreeVirtualMemory*) as its properties. The DiskSpace class has maximum storage space (*MaxSpace*) and currently available storage space (*AvailableSpace*). The Software class has currently installed CLC Work Bench version (*WorkBenchVersion*) and currently available Mini-Grid plug-in version (*PlugInVersion*) as its properties. The TaskScheduler class has count of currently running and waiting tasks (*QueueLength*) as its properties.

5.2.1 Defining resource context

The capabilities of a resource provider are stated as a set of `ContextStatement`s. By looking at the example context model, it is clear that there are static as well as dynamic context information used to describe a resource. For example, average processor load in 1, 5 and 15 minutes are dynamic context information. On the other hand, processor clock frequency is static context information. In mini-gird environment, context information about resources are provided by

`ContextMonitors`. Hence, we need to define context monitors as explained earlier.

5.2.2 Defining task context

`TaskContext` refers to the capabilities required to execute a task on a remote resource provider. For example, the task needs to be executed as early as possible and requires physical memory 2048 m bytes to execute the task. This can be stated as shown in the below code snippet.

By default, we have `GenericTaskContext` class implementing `TaskContext` interface and `GenericTaskContextStatement` class implementing the interface `ContextStatement`. However, the user is free to have his own implementation.

```
TaskContext taskContext = new GenericTaskContext();

taskContext.addStatement(new
         GenericTaskContextStatement(TaskContext.BID_TYPE,
                                              BidName.SPEED_STR));

taskContext.addStatement(new GenericTaskContextStatement
                         (TaskContext.MEMORY_REQUIRED, "2048"));
```

This section is concerned about implementing core abstraction in the Mini-Grid Framework. These implementation are sufficient for normal functioning of the framework. However, the framework permits possible extensions for specific needs of the application. For example, application specific bids and bidders can be developed by extending the concepts `Bid` and `Bidder` respectively. Some of the possible extensions are explained in the next section. However, default implementation essential abstractions have been implemented and are available to the application developer.

5.3 Framework Extension

One of the key feature of the Mini-Grid Framework is its extensibility. The framework can be extended to create custom classes for supplementing or replacing functionality supplied by the framework. The framework provides various extension points. Application specific nature of the computational tasks can be modelled by extending the interface `Task`. In a pervasive computing environment, context information have quality Buchholz and Schiffers [2003] such as precision, accuracy, etc. The interface `ContextStatement` can be extended to include these quality parameters. A domain expert can develop his own context monitors and context interpreters by extending the interfaces `ContextMonitor` and `ContextInterpreter` respectively.

A domain expert can also define his own utility function for modeling the bid by extending the interface `Bid`, used in task distribution process. Not only bid, but he can also define other auction protocols by extending the interfaces `Auctioneer, Bidder, BidEvaluator` and `AuctionListener`. In this section, we explain some of the possible extensions to the framework.

5.3.1 Defining a Context Monitor

A context monitor is responsible for acquiring context information from a sensor and associate it with a monitored entity. The context monitor registers itself with a context interpreter by stating the monitored entity it is monitoring. This process helps to associate the context information with specific monitored entity.

Let us assume that currently `ResourceProvider(A)` is available in MiniGrid. Further, we assume that among other context monitors, we have the `CPUStaticMonitor` and the `CPUDynamicMonitor` providing context information about processor of `ResourceProvider(A)`. After the deployment of context monitors at the resource provider, they need to register them self with a `ContextInterpreter`. While registering they need to specify the entity that they are monitoring and its type. Here both the monitors are monitoring the processor of `ResourceProvider(A)` and hence they register with a context interpreter with their individual unique identifier and stating that they are monitoring `CPU00` of type `ResourceType.ComputationalDevice.Processor`. The following code snippet illustrates the registration process. Here the variable `resourceId` contains the unique identifier of the resource.

```
String myComputerUri = "http://www.minigrid.org/#" + resourceId;

MonitoredEntity monitoredEntity =
                new HashtableImplEntity(myComputerUri,
                                "ComputationalDevice.Processor");

contextInterpreter.register(monitorId, monitoredEntity);
```

Static context monitors, just describe context information while registering them self with context interpreter. On the other hand, dynamic monitors provide context information at the time of registration as well as when ever the context information changes they update by calling `contextChanged()` method. This method in turn calls `addContextStatement()` method of context interpreter to which it has registered itself.

The following code snippet illustrates the provision of static context information by a monitor.

```
int count = Runtime.getRuntime().availableProcessors());

contextInterpreter.addStatement(monitorId, "ProcessorCount",
                                                        count);
```

The following code snippet illustrates the provision of dynamic context information by a monitor. Here the variable `load15Min` contains average 15 minutes processor load.

```
public void contextChanged()
{
   contextInterpreter.addStatement(monitorId, "Load15Min",
                                                    load15Min);
}
```

5.3.2 Defining a Context Interpreter

The context interpreters provide context information as context statements and these statements are stored in the `KnowledgeBase` of the `ContextManager`. The `ContextManager`s provide information in device specific format and hence we need to define mappings that convert device specific format into standard format. For example, Microsoft Windows API provides functionality to query system information. It uses the keyword "CPUSpeed" to refer the clock frequency of the installed processor. On the other hand Linux operating system provides system calls to read system information from `proc/info` file. It uses the keyword "cpu MHz" to refer the clock frequency of installed processor in MHz. Hence, two different operating system provide same information using different types of keywords, formats and units. Hence, we need a mapping function that can map context information in different syntax and semantics to a uniform syntax and semantics.

Also, we can define context interpreters that can map context information in device specific format into RDF or OWL format. Let us assume that we are using RDF to represent context information. RDF uses RDF Resource to define a class and RDF Properties to define the properties of a class in our context model. When a context monitor registers with a context interpreter, it uses a simple keyword to denote entities or resources that they are monitoring, for example `ComputationalDevice.Processor`. Hence we need to define a "RDFContextInterpreter" that can understand these keyword and convert into appropriate RDF format.

The following code snippet illustrates how mapping function can be used. "DeviceType" is an RDF vocabulary that defines the classes and properties in our context model.

```
 contextInterpreter.addMapping("ComputationalDevice.Processor",
                                              DeviceType.PROCESSOR);
```

The "DeviceType", RDF vocabulary contains among other the following statements.

```
public class DeviceType {
```

```
protected static final String uri ="http://www.minigrid.org/#";

  private static Model m = ModelFactory.createDefaultModel();

  public static final Resource PROCESSOR =
                          m.createResource(uri + "Processor");

  public static final Property PROCESSORVENDOR =
                       m.createProperty(uri, "ProcessorVendor");

    ...
  }
```

5.3.3 Defining query template

We need to define query templates that can be used by `ExecutorContext` and `Executor` to query context information. For example, we define speed query that can be used to query information that quantify speed. The below code snippet illustrates a speed query template that uses the clock frequency of the processor to formulate "SpeedBid".

```
String speedQuery = prolog + NL
  + "SELECT ?speed ?load ?queueLength "
  + "WHERE  {"
  + "?node mg:deviceType mg:ComputationalDevice ."
  + "mg:ComputationalDevice
            + mg:hasExecutionEnvironment ?executionEnv ."
  + "?executionEnv  mg:hasProcessor ?processor ."
  + "?executionEnv  mg:hasMemory ?memory ."
  + "?executionEnv  mg:hasTaskExecutor ?taskExecutor ."
  + "?processor mg:clockSpeed ?speed ."
  + "?processor mg:load15Min ?load ."
  + "?taskExecutor mg:queueLength ?queueLength ."
  + "}";

queryInterface.addTemplate(BidName.SPEED, speedQuery);
```

Later, when `Executor` receives a `bidSubmissionNotification` and has sufficient capabilities required for executing the advertised task, it queries the `ContextManager` to get context information related to bidding strategy "Speed", as shown in the below code snippet.

```
if (executorContext.isSameAsTaskContext(
                    taskSubmissionNotification.getTaskContext()))
{
```

```
   TaskContext taskContext =
                  taskSubmissionNotification.getTaskContext();

   String bidType = taskContext.getObject
            (TaskContext.DEFAULT_SUBJECT, TaskContext.BID_TYPE);

   BidName queryTemplateName = BidName.toBidName(bidType);

   Iterator<ContextStatement> contextStatements =
                       queryInterface.query(queryTemplateName);

   ....
}
```

5.3.4 Defining Bid

As stated earlier "bid" quantifies the utility function that a Mini-Grid application user has interest in. The framework comes with a default implementation that is sufficient for normal usage. However, application specific bids can be defined by implementing the interface `Bid`.

The interface `Bid`, a comparable and serializable

```
public class SpeedBid implements Bid (

   ...

   private int bidValue;
   ...

   public int compareTo(Bid bid) {

      if (!(bid instanceof SpeedBid))
         throw new ClassCastException("Invalid Object for Comparision");

      SpeedBid tempBid = (SpeedBid) bid;

      if (bidValue > tempBid.getClockSpeed())
         return 1;

      else if (bidValue < tempBid.getClockSpeed())
         return -1;

      return 0;
```

```
   }
   ...
}
```

5.3.5 User Defined Bidder

The bidder component determined the bid value based on the capabilities of the resource provider. For each type of bidding strategy used by the application, the framework user has to define bidders. The following snippet illustrates a simple bidder that calculates the bid based on the resource provider's clock speed.

The user defined bidder has to implement the interface `Bidder` and has to implement the calculate bid method. In this example, the user defined bidder queries the context manager to obtain a list of context statement that defines the capabilities of the resource provider.

```
public Bid calculateBid(TaskContext taskContext) {

  String processorSpeed = null;

  /*
   * Bidder queries context manager
   *
   */

  while (statementList.hasNext())
  {
   ContextStatement tempStatement = statementList.next();

   if (tempStatement.getPredicate().contains("ProcessorSpeed"))
       processorSpeed = tempStatement.getSubject();
  }

  /*
   * Calculate bid
   */

 int clockSpeed = Integer.valueOf(processorSpeed).intValue();
 SpeedBid bid = new SpeedBid();
 bid.setClockSpeed(clockSpeed);
 return bid;
}
```

In Section 5.1, we discussed how parallel applications can be developed. Then it presented how a resource can join or leave the Mini-Grid; how they can distributethe computational tasks over the Mini-Grid; and then to collect the

final results. We presented the implementation of the core abstractions of the Mini-Grid Framework in Section 5.2, and then the possible extensions to the framework in the Section 5.3. So far we have discussed how a new application can be developed using the Mini-Grid Framework. In the next section, we discuss how an existing application can be Mini-Grid enabled.

5.4 Mini-Grid Enabling Application Toolkit

Mini-Gird enabling an existing application toolkit requires two elements: grid enabling algorithm and changes to user interface of the application toolkit. In grid enabling algorithm, we need to design concurrent version of the algorithm, implement the concurrent version of the algorithm by implementing or extending the core abstractions such as `Task`, `Context`, etc.; and distribute the computational tasks to the Mini-Grid and present the integrated results to the user. Secondly, we have to make changes to user interface of the existing tool to use the new algorithm. This section and its subsections describe the basic tasks involved in grid enabling existing application.

5.4.1 Grid Enabling Algorithm

- ***Design Concurrent Version:*** If we have a sequential code that we want to transform into a concurrent version, we need to identify the independent computations that can be executed concurrently. Iterations of loops and function call within the code that can be executed autonomously are two instances of computations that can be independent. There are two concurrent design models – task decomposition and data composition. The former exploits functional parallelism features and is achieved by distributing task on different nodes of the distributed system. The later exploits data parallelism by distributing data across the nodes of the system. In *task decomposition*, the computations are a set of independent tasks that can be executed in any order. On the other hand, in *data decomposition* the application process large amount of data and can compute every element of the data independently.

- ***Design Implemention:*** Implement the decomposed algorithm using the required abstractions of the Mini-Grid Framework as detailed in Section 5.1 and 5.2.

- ***Result Visualization:*** Integrate the individual results of the computation and present it to the user. We can make use of data visualization support provided by the existing application.

5.4.2 User Interface Changes

- ***Configuration Management:*** The Mini-Grid Framework requires initialization of certain parameters, for example the tcp port numbers to

be used for communication among participating resources. The configuration parameters are made available through a text based configuration file. The framework comes with a default configuration file having all parameters set to default values. However, the user can define his own values by making changes to values of these parameters. The user can be asked to use any text editor to make the changes. However, to prevent the user from making mistakes, the Mini-Grid enabled application can provide a *configuration* user interface to make these changes. For example, the Mini-Grid PlugIn has a configuration user interface to set nick name to the resource and user e-mail id as shown in Figure 5.4. The workbench user has to an option **File** → **Collaboration Network (minigrid)** → **Access Minigrid Info ...** in the upper left drop down menu to invoke the configuration user interface.

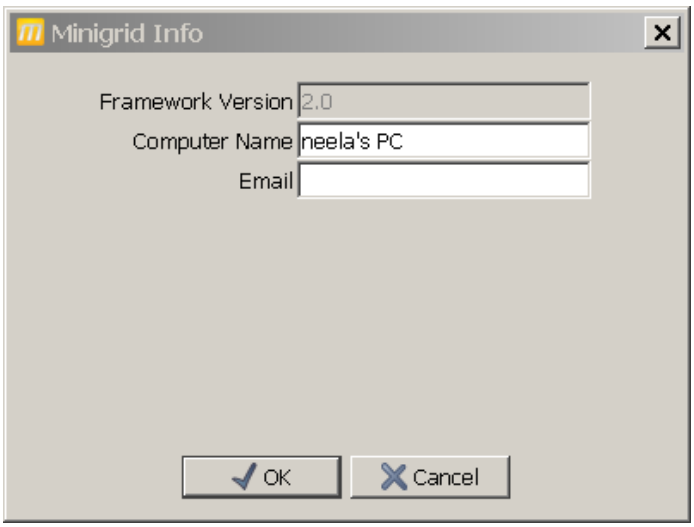


Figure 5.4: The Configuration User Interface.

- ***Resource Participation:*** When a community member can decide to participate in Mini-Grid the framework should be invoked. A Mini-Grid enabled application should allow the user to express his interest to participate or leave the Mini-Grid environment. For example, the Mini-Grid PlugIn has an user interface to achieve this. After plugin installation, a workbench user has an option **File** → **Collaboration Network (minigrid)** → **Join Collaboration Network** in the upper left drop-down menu to join the MiniGrid and has an option **File** → **Collaboration Network (minigrid)** → **Leave Collaboration Network** to leave the Mini-Grid as shown in Figure 5.5.

- ***Application Invocation:*** The user interface of the toolkit should provide facility for the user to execute a grid-enabled algorithm in Mini-Grid.

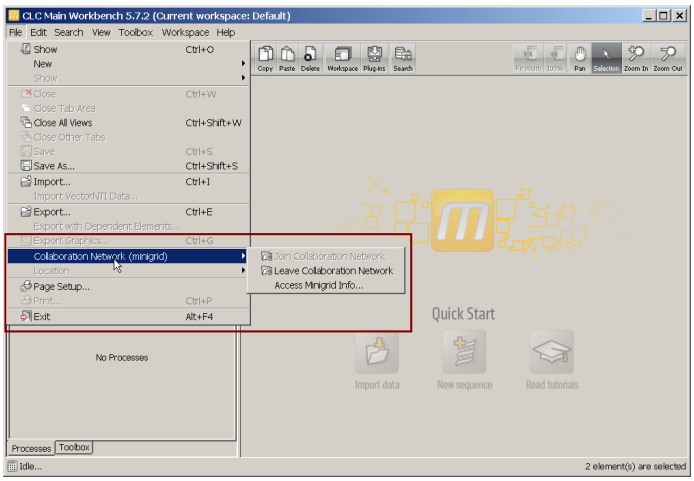


Figure 5.5: Join/Leave Mini-Grid User Interface.

For example, the Mini-Grid PlugIn makes the grid-enabled algorithm as yet another tool as shown in Figure 5.6. After plugin installation, a workbench user has an option **Toolbox)** $\rightarrow$ **Fold with PPfold** in the upper left drop-down menu or in the **Toolbox** section of the user interface.

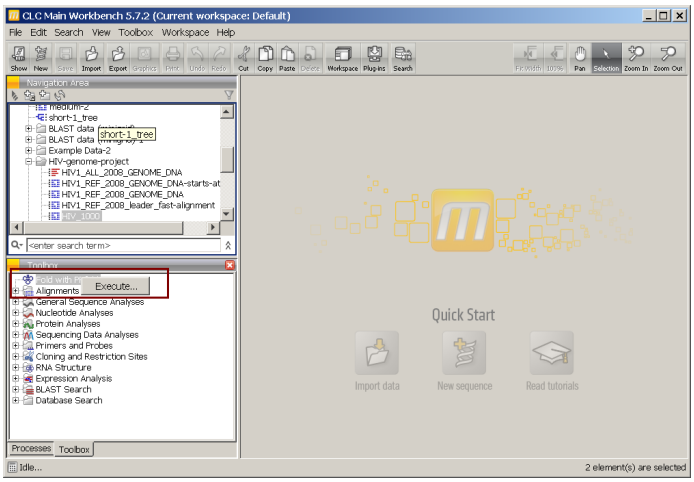


Figure 5.6: Application Invocation User Interface.

- The toolkit should have the necessary changes in its user interface for getting input for the selected algorithm. These modification are algorithm specific and are not discussed here. However, Section 8.5.2 provides an example implementation for PPfold algorithm.

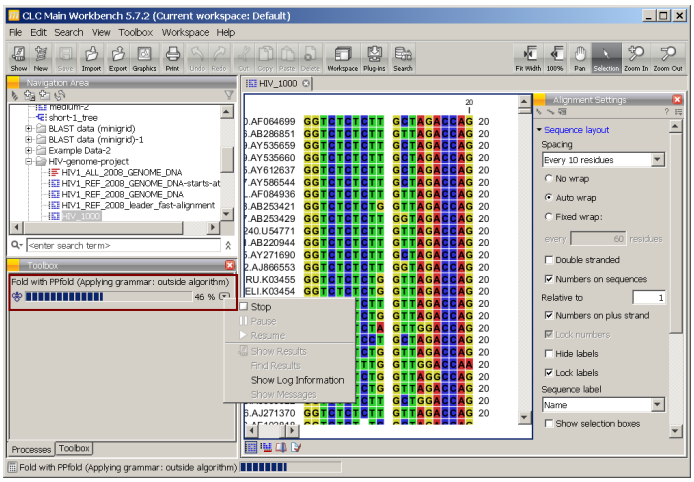


Figure 5.7: Progress of Computation in Mini-Grid.

- ***Application Monitoring:*** When a user executes an application on Mini-Grid, he should be kept informed about the progress of computation. The progress bar control provides user with a visual indicator of the progress of the application execution in the Mini-Grid. The progress bar can greatly improve the usability of the application. Informed users are less likely to do something bad, such as rebooting their resources. Progress bar also provide another essential piece of information: the program is still running and has not crashed. For example, the Mini-Grid PlugIn uses a progress bar to indicate the completed percentage of computation. After plugin installation, a workbench user has an option **Processes** in the bottom left tabbed-pan to show the progress of computation as shown in Figure 5.7.

5.5 Summary

In this chapter, we discuss about designing a parallel version of simple sample application. Then we describe how the abstractions can be used to implement the parallel version. Finally, we show how this application can be integrated into an existing application toolkit provide facilities to distribute the task while taking away the burden of becoming deeply involved with grid technology.

Part III

Evaluation and Discussion

Chapter 6

Experimental Evaluation

In order to evaluate the effectiveness of the Mini-Grid Framework compared to a single machine normally available to the user and its related overhead introduced by the auction based scheduling strategy that slows down the execution of the application in Mini-Grid, we carried out a set of performance tests.

The Mini-Grid Framework has been designed to address the computational requirements of a common class of applications, named *embarrassingly parallel*, or *bag-of-tasks* (BoT) applications. These are parallel applications that can be decoupled in a large number of independent tasks that do not need to communicative with each other. Thus, they can be simultaneously executed in a large number of desktop computers. The comparisons we present here are based on the average completion time of the BoT application. The average completion time of the BoT application is defined as the time elapsing between the submission of a bag of task and the termination (either successful or not) of all its tasks. Specifically, we are interested in studying the following:

- Overhead, measures the additional time introduced into the execution time of individual tasks from various sources in the framework.
- Speedup, measures the potential speedup of the application obtained by executing an application in the Mini-Grid environment.

6.1 Experimental Setup

In this section, we present the performance evaluation of the Mini-Grid Framework by deploying simple applications such as: prime number calculation application and application that searches for a key in a large data set in a controlled environment.

6.1.1 Application

We have considered a bag-of-task application and hence the application consisted of a fixed number of tasks and all the tasks were generated at the submis-

sion time. We have also assumed that all the tasks had the same execution time for simplicity. Some of the data parallel model applications have this property. Generally, for simulation purpose BoT applications are considered to be composed of homogeneous Lee and Zomaya [2009]; Bertin et al. [2011]. However, in real world application the execution time would be a variable. The application contained 50 independent tasks having an average run-time of 70 seconds each. The tasks were finding prime numbers between in a fixed range. The experiments were repeated until 95% confidence level was achieved and the average value was considered. This application has been used in all experiments unless otherwise stated.

6.1.2 Testbed

In this section, we present the experimental set-up used to evaluate the various aspects of the Mini-Grid Framework. This experimental set-up was used for all experiments unless otherwise stated.

For our study, the Mini-Grid ran on a set of 21 identical Intel Core Duo 2.33GHz desktop PCs with 2GB of RAM, running MS Windows XP. All these desktop PCs were connected by Gigabit Ethernet and were located on the same physical switch. Each node could play the role of both resource provider and resource consumer, but for simplification one node was configured to act as a resource consumer, and the rest as resource providers.

For each auction, the resource providers submitted `SPEEDBID` quantified by processor clock frequency and current workload at the individual resource providers. The value of the speed-bid is determined by multiplying the clock frequency of the processor and the number of cores available and then dividing it by the number of tasks queued up for execution. The bid evaluation strategy determines the winner based on the highest submitted bid value during the auction process. Dedicated access to the desktop PCs was obtained for the duration of these tests, that is, only idle resources can participate in the experimental set-up.

6.2 Average Completion Time

In this experiment, we compare the performance of Mini-Grid with a single machine under the same workload by measuring the average completion time of the application. Initially, we executed the application on a single machine with Intel Core Duo 2.33GHz desktop PCs with 2GB RAM and running MS Windows. Then we executed the same application in the experimental testbed detailed in Section 6.1.2.

Comparison of the average completion time of the computational-intensive BoT application in a single machine and the Mini-Grid environment is shown in Figure 6.1. In Figure 6.1, the X-axis shows the number of machines that participated in Mini-Grid and the Y-axis shows the average completion time of the BoT application. The average completion time (in seconds) of the BoT

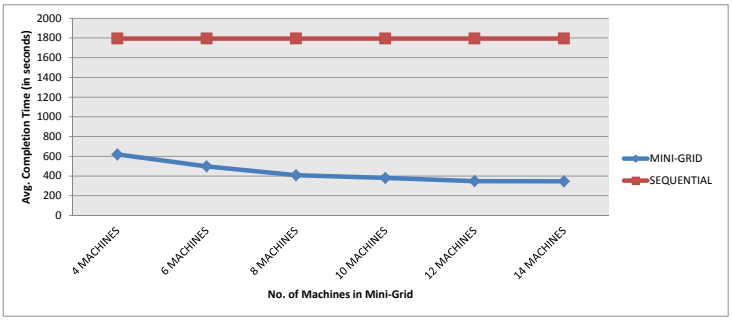


Figure 6.1: The average completion time of BoT application (in seconds) - Single machine Vs Mini-Grid

application decreases in proportion to increased participation of the resource providers. For example, the average completion time of the application on a single machine was 1793.9 seconds and the average completion time of the application in Mini-Grid with 4 machines was 619.0 seconds ($1/3^{rd}$ of the average completion time on a single machine).

However, Figure 6.1 does not show a linear speedup in Mini-Grid. In a parallel system, equal work load among all processing elements leads to linear speedup. However, unevenness in the workload partitioning makes the load of one processing element heavier than that of others. This heavily loaded processing element determines the completion time of the BoT application. Hence, we are not seeing a linear speed-up. Also scheduling the application in Mini-Grid has other sources of overhead as detailed in the next section.

6.3 Overhead

The overhead that occurs during execution of an application in Mini-Grid originates from various sources. The sources of overhead fall under three categories: auction overhead, data transfer overhead and load imbalance. The auction overhead represents the time required by the resource consumer to make a scheduling decision, that is, selection of suitable resource provider. The data transfer overhead represents the time required for transferring input/output data between the resource consumer and the remote resource provider. The load imbalance overhead occurs in the context of one or more available resource providers being idle due to the portioning of a set of task among the available resource providers. For example, if we have a set of 'n' resource providers and 'm' tasks to schedule on them, then an overhead occurs if 'm' is not a multiple of 'n'.

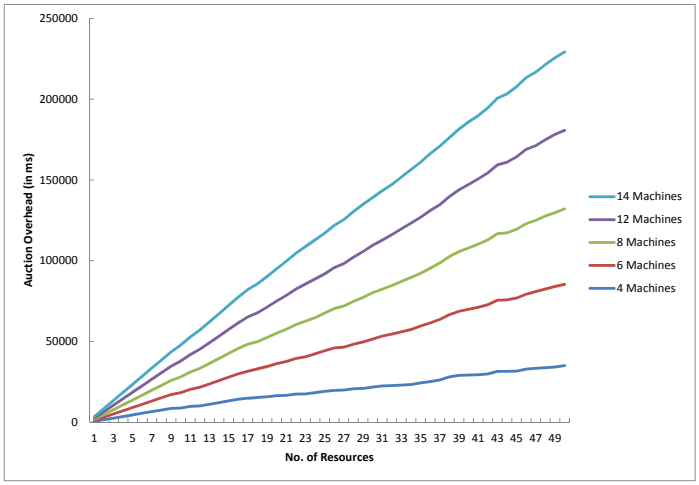


Figure 6.2: Auction overhead in Mini-Grid - Effect of queue length and resource participation

6.3.1 The Auction Overhead

The auction overhead depends on the number of messages exchanged between the auctioneer and the potential bidders, the amount of data transferred in each message, the round-trip delay (i.e., round-trip-time) of the underlying network, the current processor load at the resources, and the task generation rate. In the current implementation, messages exchanged between the auctioneer and the bidders consumes one TCP/UDP packet. The round-trip delay of the underlying network was between 50 and 120 milliseconds depending on the current network traffic and type of network.

We examine the time taken to complete the auction process by using the experimental set-up and the application detailed in Section 6.1.2. The experiment was performed for varied queue length (1-50) and for varied number of resources participating in the auction process (4, 6, 8, 12 and 14 machines). The time taken for auctioning was measured as the elapsed system clock time from the time of submission of the task to the resource consumer by the application to the time of completion of the auction process.

The time taken for auctioning each task for a varied queue length and number of resources participating in Mini-Grid is shown in Figure 6.2. In Figure 6.2, the X-axis shows the task queue length at the auctioneer and the Y-axis shows the average time elapsed (in milliseconds) for conducting the auction. For a fixed queue length, the auction time increased as the number of resources participating in Mini-Grid increased. For example, the auction time increased by 33.6% when the number of nodes was increased from 4 to 8 (50% increase in

resource participation) for a fixed queue length of 50. From Figure 6.2, it can be observed that increase in the auction time is significant (roughly 40% to 50%) when the resource participation increased from 4 to 6 due to increased message processing time (50%). However, when the resource participation is higher and for a smaller percentage of increase in the resource participation, the proportional increase in the auction time is minimal. For example, the auction time increased by 0.2%-1.45% when the resource participation is increased from 12 to 14 (16.66% increase in resource participation) due to change in message processing time being lower (13.3%).

Figure 6.2 illustrates that the auction time increases linearly in proportion to queue length. This is due to the waiting time introduced by the sequential single-item auction (i.e., tasks are auctioned one at a time) incorporated in the framework. However, combinatorial auctions where all tasks are auctioned and the bidding happens based on groups of tasks, can be implemented, but with increased complexity and communication overhead.

6.3.2 The data transfer overhead

The overhead due to stage in, the time for transferring input data required for computation from the resource consumer to the remote resource provider, and stage out, the time for transferring the output data of computation from the remote resource provider to the resource consumer, contributed to data transfer overhead. We were interested in studying the overhead introduced by this data transfer by using the experimental set-up detailed in Section 6.1.2. Firstly, we performed experiments to determine the impact of data transfer on the completion time of a BoT application. We ran an application (say "application A") containing a bag of 10 tasks and each task having a run-time of 70 seconds in the Mini-Grid environment with 10 resource providers. Next, another application (say "application B") having a bag of 40 tasks and each task having a run-time of 70 seconds and involving 80 MB of data transfer was run in Mini-Grid with 10 resource providers. Hence the applications have the same run-time and the same experimental set-up, but one application involves data transfer and the other does not involve any data transfer. Application A took 177 seconds to complete and application B took 277 seconds. Thus there was 50 seconds (28.2%) increase in run-time due to data transfer. Application A corresponds to sequential execution and application B corresponds to execution of parallel version of the application in Mini-Grid. Thus the experiment provides an idea on what kind of application is suitable for Mini-Grid. An application can speed up considerably in Mini-Grid only when the application has longer run-time compared to the amount of data transfer involved in the application.

When decomposing an application for parallelization, one approach is to logically partition the problem into a set of parallel tasks. However, while partitioning one has to consider the amount of data transfer involved. The computational intensiveness index is a ratio between the execution time of the task and the time taken for data transport. This index should have a value higher than 3 (i.e., the execution time must be more than three time the time taken for

No	Amount of data transfer involved in each task (in MB)	Total number of tasks in the set
1	20	40
2	40	20
3	80	10
4	160	5

Table 6.1: Application granularity

data transport). This value has been calculated for Local Area Network (LAN) by conducting various experiments. However, for other type of networks this value need to be calculated. Thus one has to choose the right size for partition, that is, granularity to optimize its performance in Mini-Grid. We performed experiments using the experimental set-up detailed in Section 6.1.2 to study the impact of application granularity of the BoT application on their completion time. However, the application used in this experiment was different. When scheduled in Mini-Grid, the application had to transfer 800MB input data to remote resource providers. The application was scheduled in Mini-Grid with varied granularity as shown in Table 6.1. The application was also scheduled in a single machine without using the Mini-Grid Framework and it involved no data transfer as the required data were present in the same machine.

The average completion time of the BoT application in Mini-Grid and in a single machine is shown in Figure 6.3. In Figure 6.3, the X-axis shows granularity of the application and the Y-axis shows the average completion time (in seconds) of the BoT applications with varied granularity. The average completion time of the BoT application decreases when scheduled in Mini-Grid compared to its execution in a single machine. For example, the average completion time of the BoT application on a single machine was 1793 seconds and the average completion time of the same application involving 800 MB of data transfer split into 40 tasks was 224 seconds ($1/12^{\text{th}}$ of the average completion time on a single machine). However, 20 machines participated in Mini-Grid and 20 MB of data was moved from the resource consumer to resource provider for each task. Even for application involving data transfer performs better in Mini-Grid provided sufficient number of resource providers are available and tasks are distributed evenly to all the resource providers.

As stated earlier the experiment was repeated with varied task granularity but involving the same amount of data transfer, and the results are shown in Figure 6.3. The average completion time of the BoT application increases as the granularity increases. For example, the completion time for 40 tasks, each requiring 20 MB of data transfer, takes 224 seconds and 20 tasks, each requiring 40 MB of data transfer, takes 291 seconds. Thus there is 30% increase in completion time due to increased run-time of each task (i.e., in the first case each task has 70 seconds run-time and in the second case each task has 140 seconds run-time). However, the first case had to schedule 40 tasks and hence has additional overhead.

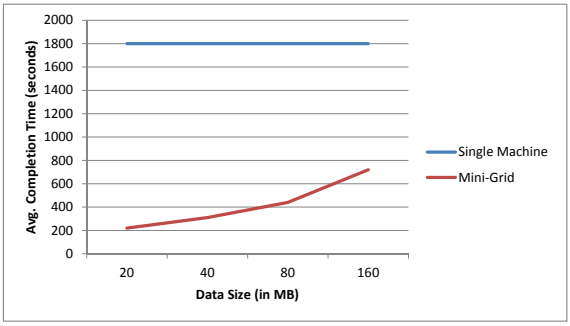


Figure 6.3: Average completion time - Mini-Grid Vs Single machine

6.3.3 The Context Processing Overhead

Each resource provider participating in the auction process need to define the bid that it is going to submit. To define the bid, the resource provider needs to use the resource context. This involves context query and processing, which consumes CPU time. In our implementation context information are stored in a RDF triplestore, a database for the storage and retrieval of subject-object-predicate triples. The triplestore needs to load data and respond to query over a knowledge base. Hence, time taken by the resource provider to define the bid depends on the query response time of the triplestore. Query response time varies widely with different implementation. A review of the literature can provide many studies on performance of RDF triplestore that are available. For example, Florian Stegmaier et al. Stegmaier et al. [2009] have stated that the execution time of query against database containing 100,000 triple set lasts for 28 milliseconds in Jena. However, other implementations like Oracle's Semantic Technologies consume even less time. This execution time would be 5-6% of the time-to-bid value. Hence, context processing time does not contribute much to the auction overhead. When the context processing time is higher than time-to-bid, the bidder (resource provider) would submit a delayed bid. In the mean time, once time-to-bid expires, the auctioneer (resource consumer) would ignore the delayed bid. Delyaed bid can be identified by the combination of resource identifier and task identifier contained in the bid.

6.4 Speed-up

According to Amdahl's law M. [1967], speed-up can be defined as the time taken for completing an embarrassingly parallel application on a single processor system divided by the time taken for completing the same application in Mini-

Grid. If T_s is the time taken to execute the application on a single processor system and if T_p is the time taken to execute the application in Mini-Grid, then

$$Speedup = T_s/T_p \tag{6.1}$$

The above equation assumes that the parallel application consists of 100 percent code that can be run in parallel. But practically, there would be certain percentage of execution that has to be in serial. Considering this factor, the Amdahl's law can be restated as

$$Speedup \leq \frac{1}{(1 - pctPar) + \frac{pctPar}{p}} \tag{6.2}$$

where $pctPar$ is the percentage of execution time that will be run in parallel, and p is the number of machines (i.e., CPUs) on which to run the parallel application. To compute speedup, the formula has taken the serial execution time and normalized it to 1. The time of the parallel execution is estimated in the denominator to be the percentage of serial time $(1 - pctPar)$ and the percentage of execution that can be run in parallel divided by the number of machines (i.e., CPUs) to be used $(pctPar/p)$. Amdahl's law gives us an upper bound on the speedup we might expect to achieve from the parallelization of a serial application. However, it ignores real-world circumstances like overhead due to scheduling management, communication, etc.

We examine the speed-up that can be obtained in Mini-Grid by using the experimental set-up and application detailed in Section 6.1.2 using the equation 6.1. The speed-up that can be obtained in Mini-Grid is shown in Figure 6.4. In Figure 6.4, the X-axis shows the number of machines that participated in Mini-Grid and the Y-axis shows the obtainable speed-up factor in Mini-Grid. And in Figure 6.4, we also show the speed-up that can be obtained theoritically using equation 6.2, assuming 5% of code requires sequentail execution. From Figure 6.4, we could see that there is an increase in speed-up with increase in the number of resource providers available in Mini-Grid. However, the speed-up is not linear, as one would normally expect as per Amdahl's law. This is due to the fact that the overheads are not incorporated in the model. We have seen that the overheads are directly proportional to the number of resources, for example the auction overhead. Now incorporating this overhead into equation 6.2, we get

$$Speedup = \frac{1}{(1 - pctPar) + \frac{pctPar}{p} + Kp} \tag{6.3}$$

where K is a fixed factor.

Now, we calculate the speedup using the equation 6.3, and assuming $K = 0.057$ (a constant value, for simplicity), as shown in Figure 6.4. One can observe that the performance of the Mini-Grid gets closer to the theoretically calculated speedup with overhead incorporated in the model. However, there is slight variation. This is due to the scheduling mechanism adopted by the operating

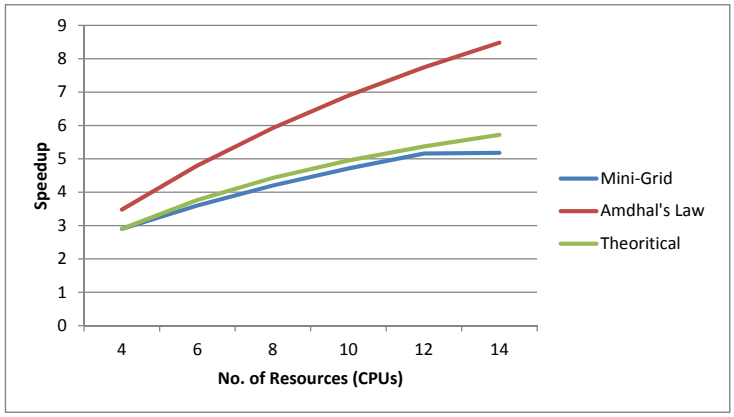


Figure 6.4: Speed-Up in Mini-Grid

system for multi core machines. We observed that the time taken to complete 'n' and 'n+1' tasks on a machine having dual core and Windows XP operating system is the same, where n is a positive odd integer. That is, the machine takes the same amount of time to complete a set of 7 and another set of 8 tasks. Though the experiments were performed with systems having Windows XP, other operating systems such as Ubuntu and Mac Os X exhibit similar property on dual core machines.

6.5 Discussions

Effectiveness of the market-based mechanism over the simple queuing algorithms like round-robin algorithm is proved in Gomoluch and Schroeder [2003]. A comparison of different auction protocols as a design choice for resource allocation in an ad hoc grid under different network conditions has been investigated and presented in Behnaz and Koen [2008]. Now, we compare the performance of our framework with Globus Toolkit, a de facto Grid system middleware and Entropy system, the commercial enterprise desktop Grid.

6.5.1 Mini-Grid vs Globus

Globus offers Grid information services via an LDAP-based network directory called Metacomputing Directory Services (MDS) Laszewski et al. [1997]. Globus MDS service consists of two components: *Grid Index Information Service* (GIIS) and *Grid Resource Information Service* (GRIS). GRIS provides resources discovery services on a Globus based Grid. The directory information is provided by a Globus component running on a resource or other external information

Grid System	Response Time (1 Query / 1 Auction) (m sec)	Response Time (10 Query / 10 Auction) (m sec)	Response Time (50 Query / 50 Auction) (m sec)
MDS2	1219	5534	29,175
Mini-Grid	603	9635	45,758

Table 6.2: Comparison of resource discovery performance in the Globus and the Mini-Grid environments

providers. The resource information providers use a push protocol to update GRIS periodically. GIIS provides a global view of the Grid resources and pulls information from multiple GRIS to combine into a single coherent view of the Grid. Thus Globus can be considered to follow "information-push" model since the resource information are periodically pushed from resource providers. Resource discovery is performed by querying the MDS.

Experimental study conducted in Schopf et al. [2005] presents the performance of MDS (v. 2.4.3) without caching. Their testbed consisted of a set of 5 client nodes and one server node, all connected by the same Gigabit Ethernet physical switch. The server machine had a dual Intel (hyperthreaded) Xenon running at 2.20GHz with 1GB of RAM. Each client machine had a dual CPU 1133MHz Pentium III machine with 1.5GB of RAM. The performance evaluation has been conducted with sequential queries from different clients against one Grid Information Service running on one server machine. The entries in the registry were updated once in 10 minutes. The index had 10 entries (i.e., 10 resource providers were present in the Globus Grid and registered them selves).

We have used the experimental setup described in Section 6.1.2. The resource discovery mechanism in Globus is querying the MDS service and the mechanism in Mini-Grid is auctioning. Thus we can compare the average auction time incurred in the Mini-Grid with query performance of the MDS index service. The two testbed setups were slightly different. However, the network environment in both testbed setups are similar and the computational capacity of server machine in Globus and resources in the Mini-Grid testbed are comparable. Table 6.2 shows a rough comparison of resource discovery mechanism in the Globus Grid environment and Mini-Gird.

At first glance, by looking at Table 6.2 one could conclude that the Globus MDS performs 60-70% better than the Mini-Grid approach. However, it is important to note that the time taken by Globus MDS is only for resource discovery. On the other hand, in Mini-Grid, it is not only resource discovery but aslo for resource selection and resource allocation. Also, it is important to note that Globus MDS provides recent but not guaranteed to be absolute latest resource information. For example, in the Globus experiment, resource information was updated once every 10 minutes. When the frequency of resources joining and leaving the grid system increases, Globus MDS could provide stale information.

6.5.2 Mini-Grid vs Entropia

Resource availability traces collected from the deployment of Entropia desktop Grid at San Diego Super Computer Center (SDSC) reports an average small gap length of 35.9 seconds Kondo [2005]. These short gaps occur exclusively in between the termination of a task and the beginning of a new task on the same host. The sources of this delay include various system costs of receiving, scheduling and sending a task as well as an actual built-in limitation that prevents the system from sending tasks to resources too quickly. That is, the Entropia server enforces a delay between the time it receives a request from the worker and the time it sends a task to that worker Kondo [2005]. In Mini-Grid, the tasks are made available to resource consumers (i.e., workers) through TaskBus. The TaskBus is a kind of Distributed Hash Table (DHT). Once the resource consumer receives a task from an application, it puts the task into the TaskBus. Once the resource provider receives the winner notification, it retrieves the task from the TaskBus and executes it. Thus, we do not have any forces delay between the time the resource consumer sends the winner notification and the time the resource provider retrieves the task from the TaskBus.

6.5.3 GridFTP vs TCP

One of the important source overhead in any Grid computing environment is due to data transfer. Currently, the Mini-Grid Framework uses TCP/UDP-based socket communication for data transfer. Globus based Grid environments uses GridFTP protocol William et al. [2005] for data transfer. Now, we compare the performance of the GridFTP protocol and TCP/IP-based socket communication. Even though the GridFTP protocol uses underlying TCP based socket communication, it uses multiple TCP streams in parallel. A recent performance evaluation reported in Butler [2010] points out that the use of an optimized modern TCP stack in an uncongested lossless network makes it possible to achieve high data rates without employing GridFTP. Further, it points that using an un-optimized TCP stack in an uncongested network it is possible to achieve high data rates by employing parallelization technique.

In the Mini-Grid Framework implementation, we have used up to a maximum of five parallel TCP connections for data transfer. However, we have not optimized buffer sizes used in these TCP connections.

6.5.4 Semantic Vs Keyword Based Matchmaking

Semantic based matchmaking results in greater "hits" than conventional keyword based matchmaking. The semantic based matchmaking using subsumption relationship returns not only exact matches but also related matches. But conventional keyword based matching returns only exact matches. However, conventional keyword based matching has less overhead in the matching process compared to semantic approach Amarnath et al. [2009].

In the Mini-Grid Framework, semantic based matchmaking does not introduce additional overhead. Each resource provider has a context manager that evaluates the suitability of the resource provider to execute the task and then executes a query to calculate the bid value. Since, we use simple context model that has relatively less number of concepts the overhead introduced by the semantic based technique falls below the time-to-bid (TTB) value. However, it has one time initial memory overhead during loading the ontology. We have used Java Simple Object Notation (JSON) for serializing task context information and transferring the context information between resource provider and resource consumer. Further, overhead in loading ontology and querying ontology depends on the RDF store used. Also main memory based RDF store perform better compared to their database based RDF store. We have used the Jena framework which requires significant improvement in query processing as reported in literature on evaluation of RDF Querying framework Schönberg and Freitag [2009].

6.5.5 Conclusion

Scheduling BoT application in the Mini-Grid environment does not show a linear speedup with increase in number of resources participating in the Mini-Grid. Unevenness in workload partitioning and other sources of overhead prevent a linear speedup. The auction overhead increases with increase in queue length at the resource provider and the number of resources participating in the Mini-Grid. For data intensive applications, speedup can be expected to increase linearly only when the execution time of a task is higher than the time required for performing data transfer. The speedup depends on the local scheduling algorithm used in multi-core desktop computers. Semantic-based resource matchmaking introduces additional overhead compared to traditional keyword-based approach. However, the matching algorithm provides higher hit count compared to traditional keyword-based matching algorithm.

6.6 PPfold Lab Deployment

In this section we present the performance evaluation of the Mini-Grid Framework when used in a real application, a parallel version of RNA secondary structure prediction algorithm (PPfold).

6.6.1 Application

Ribonucleic acid (RNA) is an important molecule that performs a wide range of functions in biological systems. In particular, it is RNA that contains genetic information of viruses such as HIV and thereby regulates the function of these viruses. Hence, it is important to obtain RNA structural information. RNA molecules have two sets of structural information: the primary structure of

RNA and tertiary structures. RNA secondary structure is a restricted subset of tertiary structure Wang and Zhang [2002].

Many computational methods have been proposed in an attempt to predict RNA secondary structures. In this study, we focus on pfold B and J [1999, 2003], a Stochastic Context Free Grammar (SCFG) based RNA secondary structure prediction algorithm. The algorithm uses phylogenetic stochastic context free grammar (phylo-SCFG) that predicts RNA secondary structures. SCFG are probabilistic models that consists of symbols and production rules with associated probabilities that can capture primary sequence features as well as long range interaction between base pairs in a RNA secondary structure. Pfold uses phylo-SCFG to predict the most likely common secondary structure based on a model of secondary formation combined with a phylogenetic analysis of the observed substitution pattern. Pfold use inside-outside method to estimate parameters for structural prediction of the SCFG. The inside-outside method, a forward-backward algorithm used to compute the probabilities that a given production will be used in a random derivation of a sequence. Once we have the SCFG, we can train it by using Expectation Maximization using the inside-outside method to estimate the maximum likelihood probabilities. We use a parallel version of the algorithm. The algorithm will be discussed in Section 8.2.2.

6.6.2 Testbed

For our study, Mini-Grid ran on a set of six heterogeneous desktop PCs having clock frequencies ranging from 1.1GHz to 2.1GHz and physical memory ranging from 1GB to 2GB. All these desktop PCs were connected by Gigabit Ethernet and were located on the same physical switch. Each node could play both the role of resource provider and resource consumer, but for simplification one node was configured to act as a resource consumer and the rest as resource providers.

For each auction, the resource providers submitted SPEEDBID quantified by processor clock frequency and current workload at the individual resource providers. Dedicated access to the desktop PCs was obtained for the duration of these tests (i.e., only idle resources can participate in Mini-Grid).

6.6.3 Distributing PPfold in Mini-Grid

The algorithm generates intermediate results that need to be stored on the resource consumer node. Thus, the algorithm puts high main memory requirements, for example to fold HIV genome of 10000 base pairs it requires 5-6GB of RAM. Hence for our experiment, we have assumed that the resource consumer has the capability to store all the data. On the other hand, the memory requirements for resource providers are not so high, as they need to store only the data necessary for execution of the current calculation. The data necessary for each task depends on the number of division, a parameter to the algorithm. As explained in Sükösd et al. [2011], even for a smaller number of divisions the algorithm places memory requirements that a resource provider cannot meet.

To avoid overloading resource providers, we have utilized the context modeling features of the Mini-Grid Framework. As stated earlier, a task has task context detailing the requirements of that task. The algorithm provides an estimation of memory requirements as part of the task context. The submitter announces these memory requirements as part of task context information and only executors having sufficient memory could participate in the bidding process. This avoids overloading the resource providers. For our experiments we have used the utilization function based on the ability to complete the task at the earliest (i.e., the bid is modeled using the clock frequency of the executor and its current workload). The submitter selects the executor that can complete the task quickly and send the task to that executor for execution.

Algorithms such as PPfold put another challenge for deploying them in Mini-Grid involving large amount of data transfer between the resource consumer and resource provider. The performance depends on the amount of data transfer involved. The framework's overhead increases by a factor proportional to the amount of data transfer. To overcome this issue, the algorithm reports an estimation on the amount of data transfer required for each task and their run-time. Then based on this information the framework makes a decision on scheduling the task in Mini-Grid. We calculate the computational intensiveness of the task, a ratio between the run-time of the task and the time required for transferring the data necessary for that task. If the computational intensiveness index falls below a threshold value, then the framework schedules the task locally instead of distributing it in Mini-Grid. The threshold value for computational intensiveness index has been determined based on the trial runs in Mini-Grid.

6.6.4 Theoretical Speed-up

The algorithm might not scale well even with an increase in the number of resources participating in Mini-Grid. We illustrate this in Figure 6.6, using the inside part of the algorithm in PPfold as an example. Each task takes a certain number of "units of time" for execution (relative), which depends linearly on its height in the triangle. Tasks in each row are assumed to be executed simultaneously if additional resource providers are available. The theoretical speedup for n cores is expressed as

$$s_n = \frac{\text{total units of time on } n \text{ cores}}{\text{total units of time on one core}}$$

Further, the theoretical optimal speedups vary widely with the chosen number of divisions (N) and the available number of cores (n), for both the inside and the outside parts of the algorithm. The expectation value calculations are analogous to the inside algorithm and hence their theoretical performance is expected to be the same. Figure 6.5 illustrates the complexity of how the theoretical maximum speedups for execution on a single machine with two-cores versus distributed in Mini-Grid with six-cores vary as a function of divisions for the different parts of the algorithms. In Figure 6.5, the X-axis shows the number of divisions chosen in SCFG part and the Y-axis shows the theoretical

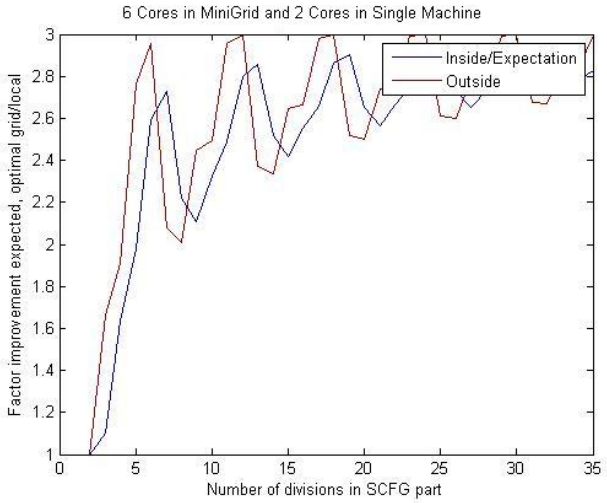
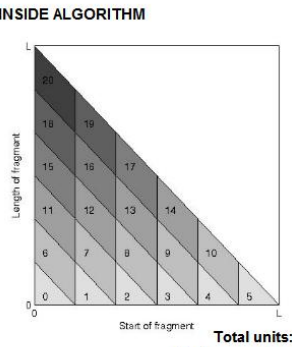


Figure 6.5: Theoretical speedup calculation - Comparing dual core Computer and six-core Mini-Grid.

speedup factor obtainable. From the figure one could observe that periodically the expected factor of improvement peaks. The periodicity in the figures is related to the divisibility of the number of cores with the number of chosen divisions. In the limit where $N \to \infty$, the factor speedup approaches n. This has been observed in practice on a shared-memory infrastructure Sükösd et al. [2011]. The stated theoretical model assumes that all tasks are distributed all the time with zero data transfer, as in a shared-memory environment. This is not a realistic model for Mini-Grid; therefore its performance in Mini-Grid is expected to decrease significantly.

Theoretical speedups can be calculated in a similar manner for any number of divisions, for the outside and the expectation parts of the algorithm. In Figure 6.6, we illustrate the theoretical speed-up factor that can be obtained for the inside part of the PPfold algorithm by varying the number of cores available in Mini-Grid. For this illustration, we have considered six divisions (corresponding to the 6 tasks in the bottom row), input parameter to the algorithm. The first column in the table corresponds to the theoretical run-time of the inside part of the PPfold algorithm in Mini-Grid with single core. When an additional core is available, the algorithm is expected to complete in 62.5% of the time taken when executed on one core. With the availability of four additional cores, the algorithm is expected to complete in 42.6% of the time taken when executed on single core.



	Units of time on 1 core	Units of time on 2 cores	Units of time on 5 cores
	1 x 30	1 x 30	1 x 30
	2 x 24	1 x 24	1 x 24
	3 x 18	2 x 18	1 x 18
	4 x 12	2 x 12	1 x 12
	5 x 6	3 x 6	1 x 6
	6 x 1	3 x 1	2 x 1
Total units:	**216**	**135**	**92**
Factor speedup:	**100%**	**62.5%**	**42.6%**

Figure 6.6: Theoretical speed-up.

Experiment	Parameters		No. of cores in Mini-Grid
1	12	9	6
2	9	12	6
3	3	7	6
4	7	14	6
5	12	12	12
6	24	24	12

Table 6.3: Experiment Details

6.6.5 Experimental Results

The scope of the controlled experiments has been limited to a single biological problem, namely the folding of an alignment of 3000 nucleotides in length. This alignment represents a sequence that is somewhat longer than the large ribosomal subunit, and thus presents a realistic scenario of a problem that a biologist could solve on their own laptop, but would probably prefer to speed up with the aid of Mini-Grid. Table 6.3 shows various controlled experiments that were carried out and the parameters used in those experiments. We have varied the phylogenetic divisions as shown in column 2 and FCFG divisions as shown in column 3 and the number of cores used in Mini-Grid for the experiment.

The results of these experiments on a single machine with dual core are provided in Table 6.4. The second, third, fourth and fifth columns of the table provide the execution time (in seconds) of phylogenetic part, inside part, outside part and expectation part of the algorithm respectively.

The results of the experiments outlined in Table 6.3 in Mini-Grid are provided in Table 6.5. The second, third, fourth and fifth column of the table provide the execution time (in seconds) of phylogenetic part, inside part, outside part and expectation part of the algorithm respectively.

Figure 6.7 compares the speedups achieved in the Mini-Grid environment

Experiment	Phylo. Part	Inside Part	Outside Part	Expectation Part
1	386.5	668.4	1406.3	188.9
2	408.7	623.2	1296.7	175.3
3	468.2	678.2	1482.3	201.5
4	415.2	592.1	1253.7	169.0
5	415.2	592.1	1253.7	169.0
6	375.2	591.3	1231.1	164.3

Table 6.4: Execution Time on Single Machine (2 Core).

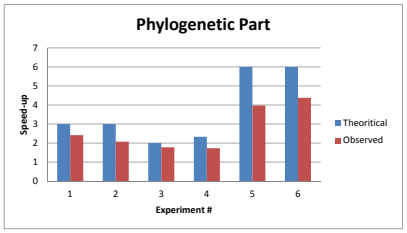
Experiment	Phylo. Part	Inside Part	Outside Part	Expectation Part
1	160.0	413.0	612.2	176.4
2	197.1	371.0	545.9	148.2
3	263.7	522.9	766.3	178.5
4	240.0	358.4	550.7	155.0
5	97.2	349.7	415.6	145.9
6	85.7	332.4	354.3	168.7

Table 6.5: Execution Time on Mini-Grid.

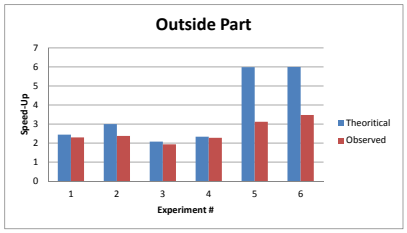
with the theoretical maximum speedup obtainable. Considering the fact that the theoretical speedup calculations do not consider the overhead due to data transfer, the speed-up observed in the experiments as shown in Figure 6.7a and 6.7b is consistent with the theoretical obtainable speed. However, the Figure 6.7c and 6.7d shows that the speed-up observed in the experiments decline sharply compared to the theoretical obtainable speed. This is due to the fact that a fraction of the tasks were not having the required computational complexity and hence were not distributed to the Mini-Grid environment and they were run sequentially. For example, for experiment# 1 detailed in the Table 6.5 out of 45 tasks generated 9 tasks were not distributed to the Mini-Grid ($1/5^{\text{th}}$ of total tasks generated). In summary, the Mini-Grid framework performs better when the individual tasks of the application have higher computational complexity index.

6.7 Summary

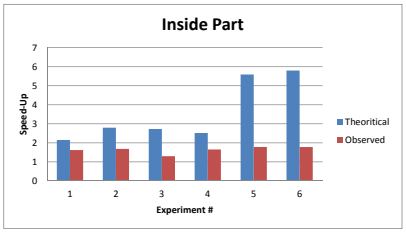
This chapter has presented performance evaluation using various experiments. First, we present the evaluation using a simple sample application. We have presented the speedup factor achieved in the Mini-Grid in comparison with theoretical upper bound on the speedup using Amdahl's law. Next, we examine the performance degradation due to different sources of overhead. Then, we present the performance comparison of the Mini-Grid with other popular alternatives. Secondly, we present the performance evaluation using controlled lab experiments using the ppfold application. We compare and discuss the performance



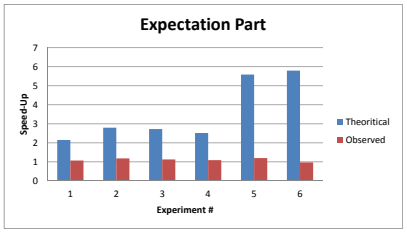
(a) Phylogenetic Part



(b) Outside Part



(c) Inside Part



(d) Expectation Part

Figure 6.7: Speed-up: Theoretical Vs Observed

of the Mini-Grid with theoretical obtainable speedup. We conclude that the Mini-Grid performs better if the tasks have a higher computational complexity index.

Chapter 7

Simulation

Intuitively, auction-based task allocation should achieve better performance than traditional centralized scheduler. To validate this intuition, we developed a discrete event simulator (based on SimJava libraries F and R [1998]) and used to compare the performance of auction-based and centralized scheduling strategies for variety of different workloads. The comparison we present here is based on the two metrics:

- Average number of *task execution failures*, defined as the number of tasks that failed because of resource failure.
- Average number of *task scheduling failures*, defined as the number of tasks that failed because of unavailability of resources to execute a task.

7.1 The architecture of the simulator

In the Figure 7.1, we illustrate the various entities involved in our simulation of traditional centralized approach: the *ResourceBroker* is the entity that receives the task from a BoT application and makes a scheduling decision and decides the *ResourceProvider* represented as R_1, R_2, etc. The Grid Information system (GIS) queries the participating resources about their availability and provides resource status information to the ResourceBroker on request. The *Resources* are responsible for execution of the task. When a task is submitted to the ResourceBroker, it queries the GIS and decides the Resource that executes the task. The Resource notifies the ResourceBroker after it has completed the task execution. The GIS queries periodically for status of resources. We have used a query interval interval of 600 seconds and 0.19 milliseconds for query response time in our simulation. These parameters are obtained from the study reported in Laszewski et al. [1997].

We have also developed a Mini-Grid simulator that has all the Mini-Grid components such as Resource Provider, Resource Consumer, Auctioneer, Bidder, TaskBus, TaskExecutor, Messenger, and Context-Management sub-module.

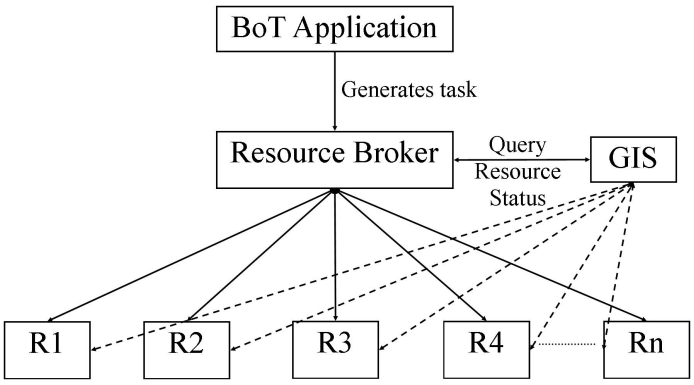


Figure 7.1: Centralized Resource Scheduling - Simulator Components.

The application generates BoT tasks and submits them to ResourceConsumer. The ResourceConsumer schedules the task on the available resource. After completion of the task, the resource notifies the ResourceConsumer about the result of execution. The ResourceConusumer distributes the task to the ResourceProvider based on the auction mechanism. The ResourceProvider has ContextManager that can provide context information about the ResourceProvider. The ResourceProvider also has two types of ContextMonitors: the CPU monitor and the TaskExecutorMonitor. They provide static information such as CPU processor and dynamic information such as current CPU workload. All these components are developed as entities using SimJava library. We have used 600 milliseconds for Time-to-Bid and assumed 200 milliseconds of network latency.

7.2 Desktop Grid Configurations

The space covering the relevant parameters for our study is too large to explore exhaustively. Therefore, we fix a number of system characteristics. For our study, we have considered a fixed value for the total computational power of the Grid in all the configurations. The total computational power P of a configuration is defined as the sum of the computing powers P_i of the individual resources. However, individual Desktop Grid configurations differs from each other in the availability of the participating resources. The computing powers P_i of the individual resources, expressed as a real number whose value is directly proportional to the speed of the resource (i.e., a resource i with $P_i = 2.5$ is twice faster than a machine j with $P_j = 1.25$). For this study, we have fixed the value of P to be 1000 for all configurations and the actual computing power of the individual resource to be 10 (i.e., $P_i = 10$).

We consider four distinct desktop grid configuration with different resource availability levels (i.e., the percentage of time that a given resource is available for computation in spite of preceding failures) as detailed by Joshua and Henri Joshua and Henri [2007] . The *fault time* (i.e., the time elapsing be-

Sl.No.	DG Name	Shape	Scale	Availability
1	Condor	275599	0.545	93.17%
2	CSIL	282700	0.554	93.17%
3	Entropia	13190	0.550	39.18%
4	UCB	444	0.355	0.06%

Table 7.1: DG Configuration.

tween two consecutive failures) is assumed to be a random variable with a Weibull distribution as reported by Daniel Nurmi et. al. Daniel et al. [2005]. And the *repair time* (i.e., the time elapsing between the occurance of fault and the time when the resource becomes operational again) is assumed to be uniformly distributed between 120 and 600 seconds for a reboot as reported by Brievik et. al. J. et al. [2004] or exponentially distributed with a mean of 2 days for a hardware crash as reported in by Soonwook Hwang and Carl Kesselman Hwang and Kesselman [2003]. The mean of the fault time (mean time between faults - MTBF) is equivalent to the mean of the Weibull distribution, defined as $MTBF = WeibullMean = scale * G((shape + 1)/shape)$ where G is the complete gamma function.

In simulation, we have fixed the MTTR value by assuming the mean reboot time to be 360 seconds (the mean between 120 and 600 seconds) and the mean recovery time to be 172800 (expressing 2 days in seconds). Therefore, MTTR is defined by $MTTR = 0.2 * MeanRecoveryTime + 0.8 * MeanRebootTime$. In our scenario, the MeanRebootTime = 360 seconds (the mean between 120 and 600 seconds) and the MeanRecoveryTime = 172800 (the 2 days in seconds). Therefore, the MTTR is $MTTR = 0.2 * 360 + 0.8 * 172800$. Thus the MTTR has a constant value 34848 seconds.

The availability α of the machines can be obtained from MTTB and MTTR values using the relation $\alpha = MTBF/(MTBF + MTTR)$. The various desktop configurations used in the simulation and their resource availability are tabulated in the Table 7.1. The table provide shape and scale parameter of Weibull distribution that characterize the resource availability. Using these two parameters we can calculate the mean of the distribution that corresponds to availability of resource in the respective configurations.

7.3 Workloads

For our study, we considered various workloads representing different types of BoT applications with different task submission rate. The type of application corresponds to the degree of parallelism represented by task granularity. Further, tasks of same granularity have approximately the same mean execution time.

In general, on week days, a daily cycle with a high submission rate during the working time and low submission rate during the night has been reported

Sl.No.	Task Granularity	Task Count	Application Size
1	1000	3600	3600000
2	5000	720	3600000
3	25000	144	3600000

Table 7.2: Workload.

by Dror Feitelson and Bill Nitzberg Dror and Bill [1995]. The peak is in late morning with noticeable drop during the lunch. This pattern can be represented by assuming exponential distribution for task interarrival rate (i.e., the number of tasks submitted per unit of time). We have assumed that task submission rate to be exponentially distributed with inter-arrival rate 270 seconds. In our simulation study, we have considered three different granularity vales 1000, 5000, and 25000 seconds, respectively. These three granularity levels correspond to three different types of application. The actual execution times of individual tasks have been assumed to be uniformly distributed. In our study, we have assumed a constant application size in all workload configuration and hence the number of tasks in the individual application depends on the task granularity. The number of tasks in a BoT application can be determined by adding tasks to the BoT until their execution time reach the above application size as tabulated in the Table 7.2. Task granularity represents the size of the individual tasks in the BoT application. For example, a task with granularity 1000 takes on an average 1000 seconds on a machine with $P_i = 1$. Each row in the table represents a BoT application. However, the total execution time of the BoT application remains a constant.

7.4 Validation of Simulation Results

In order to validate the results of our simulation, we have used a validation technique called *Fixed Values* Sargent [1995]. Fixed values are used for various input parameters and internal variables and the results of the simulation are compared to easily calculated values. Further, trace files have been used to walk through the events occurred during simulation and validate manually the events. Since, the calculated values and the simulated values are equal and walk through identifies all the events during simulation to be as expected, we can be confident with correctness of our simulator.

7.5 Results and Discussion

7.5.1 Impact of Resource Failures on Scheduling

In this section, we would be presenting the impact of failure on scheduling in the centralized approach and the Mini-Grid approach. The results of the experiments conducted using the centralized approach are tabulated in the Table 7.3

Sl.No.	Granularity	Submitted	Completed	Failed	DG Configuration
1	1000	3600	431	3169	Condor
2	5000	720	580.5	139.5	Condor
3	25000	144	122.9	14.1	Condor
4	1000	3600	432.4	3167.6	CSIL
5	5000	720	580.3	139.7	CSIL
6	25000	144	129.9	14.1	CSIL
7	1000	3600	373.9	3226.1	Entropia
8	5000	720	423.3	296.7	Entropia
9	25000	144	85.6	58.4	Entropia
10	1000	3600	172.6	3427.4	UCB
11	5000	720	102.1	617.9	UCB
12	25000	144	11.7	132.3	UCB

Table 7.3: Results for Centralized Approach.

and the experiments using the Mini-Grid approach are tabulated in the Table 7.4. The experiments have been repeated by varying the granularity and the desktop grid configuration. The experiments were repeated until 95% confidence level was achieved and the average value was considered. Each row corresponds to an experiment and provides information about the task granularity and the desktop grid configuration used in the experiment. Then, we provide information on number of tasks submitted, number of tasks completed and number of tasks failed.

From the Table 7.3 and Table 7.4, we could see that the number of task failed to execute decrease with increase in the task granularity and increases with decrease in resource availability. The task failures can be because of two reasons: failed during execution because of resource failure and failed during scheduling because of non-availability of resources. In the centralized approach, the clients update the registry or central scheduler or resource broker only at certain frequency, for example MDS of Globus toolkit. Further, the clients contact the ResourceBroker or central scheduler to inform about their availability only when they have completed the execution of currently scheduled task, for example BOINC client. But in the Mini-Grid approach, each time the ResourceConsumer schedules a task, it announces to all and the resources independent of what they are doing participate in the auction process. Hence, tasks can be queued at the ResourceProviders in the Mini-Grid approach. Further, the ResourceConsumer can select the resources based on their current workload. In the Mini-Grid approach, the failures increases slightly with increase in task granularity. In both cases, the number of failure increases with decrease in availability of resources. However, the Mini-Grid approach performs better than the centralized approach as shown in Figure 7.2. This is because centralized approach mostly uses stale information for scheduling and the Mini-Grid uses current information.

Sl.No.	Granularity	Submitted	Completed	Failed	DG Configuration
1	1000	3600	3436.3	163.7	Condor
2	5000	720	683.4	36.6	Condor
3	25000	144	133.9	10.1	Condor
4	1000	3600	3443.5	156.5	CSIL
5	5000	720	684.4	35.6	CSIL
6	25000	144	134.7	9.3	CSIL
7	1000	3600	2897.1	702.9	Entropia
8	5000	720	564.1	155.9	Entropia
9	25000	144	98.7	45.3	Entropia
10	1000	3600	1347.3	2252.7	UCB
11	5000	720	241.6	478.4	UCB
12	25000	144	32.6	111.4	UCB

Table 7.4: Results for Mini-Grid Approach.

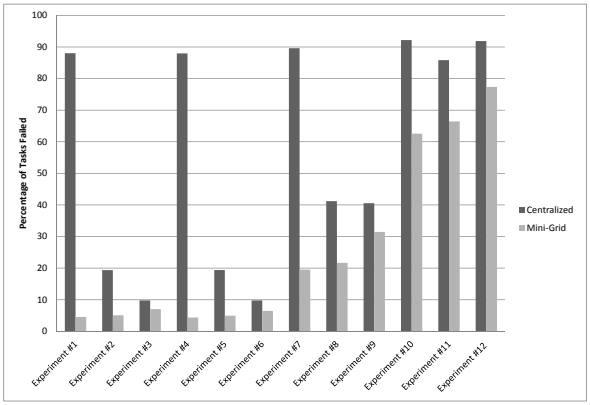


Figure 7.2: Failures : Centralized Resource Scheduling Vs Mini-Grid.

Sl.No.	RC Count	RP Count	Submitted	Completed	Failed	%of Completion
1	1	100	1024	1010.8	13.3	84.23
2	2	100	512	502.3	9.7	83.70

Table 7.5: Multiple Resource Consumers - Impact of Failures.

7.5.2 Impact of Multiple Resource Consumers

The experiment used for studying the impact of presence of multiple resource consumers is slightly different from the workload presented earlier. For this study, we have used an application that has 1200 tasks having an average execution time of 1000 seconds. The actual execution time of the individual application have been assumed to be uniformly distributed. The Mini-Grid environment consists of 100 resource providers and 2 resource consumers. The resource providers can participate in both the auctions conducted by the two resource consumers. The actual computational power of the individual resource providers have been assumed to be a fixed value 10. Simulation of resource failures are according to the Condor desktop grid configuration stated in the earlier simulation experiment. We have Further, we have assumed that all the 1200 tasks are submitted at the same time by the application to the resource consumer. The results are tabulated in the Table 7.5. This table presents the two experiments that were conducted for evaluating the impact of failures when multiple resource consumers are present. In the first experiment, a single application submits all the tasks to a single resource consumer and the results are presented in the first row of the Table 7.5. In the second experiment, two applications submit 512 tasks each to the two resource consumers and the results are presented in the second row of the Table 7.5. We could observe that the percentage of completion of tasks in both the experiments are almost same. Thus resource failures have an impact that is independent of the number of resource consumers present in the Mini-Grid environment.

During the above two experiments, we have also measured the average *task completion time*, the time elapsing between the submission of a task and its termination, if successful and the average *task scheduling time*, the time elapsing between the submission of a task and its start of execution, if successful. The results are tabulate in the Table 7.6. When the workload is shared by the increasing the resource consumers, the average task scheduling time decreases. However, the average task completion time does not change much even though the average task scheduling time decreases by 50%. We have observed that during the simulation, when multiple resource consumers conduct auctions at the same time, the resource providers submit bids that are proportional to the TaskExecutor queue length. Since, there is a time gap between the submission of bids and the award of a task for execution, they submit bids with same value. It may be possible that some resource providers win both auctions and hence, a waiting time is introduced for the second task. The bids do not represent the fact that the resource provider is participating in multiple auction process. This

Sl.No.	Avg. Task Completion Time	Avg. Task Scheduling Time
1	612.22	312.43
2	616.83	161.38

Table 7.6: Multiple Resource Consumers - Impact on Task Completion Time.

issue needs further investigation.

7.6 Summary

In this chapter, we have presented the performance of auction-based scheduling strategy in the Mini-Grid Framework. In order to evaluate the performance, we compare the auction-based strategy with centralized scheduling approach. In both of these cases we observe that task granularity has an impact on task execution failure rate. However, the Mini-Grid Framework performs better compare to the centralized approach. Presence of multiple resource consumer does not affect the performance. However, freshness of context information has an impact on the average task completion time. Context gets updated after submission of bids and before declaration of auction result. This problem can be solved by using iterative auctions in which the resource providers can bid simultaneously in a single round. In iterative auctions, the auction protocol repeatedly interacts with the different bidders, aiming to adaptively elicit enough information about the bidders. Iterative auction has been designed to solve resource allocation problems, for example for auctions in multiple identical items M. [2004].

Chapter 8

Real-World Usage and Deployment at a Biology Lab

In this chapter, we describe the deployment of a parallelized sequence search tool (Basic Alignment Search Tool) Altschul et al. [1990] and RNA secondary structure prediction algorithm (PPfold) Sükösd et al. [2011] in Mini-Grid at iNano (interdisciplinary Nanoscience center) at Aarhus University, Aarhus[1]. BLAST is designed to search all available sequence databases for similarities between a protein or DNA query and known sequences. The PPfold is a parallel version of RNA secondary structure prediction using stochastic context-free grammars.

Department of Molecular Biology, University of Aarhus, Aarhus is organized as "labs". Each lab has a professor as its head and a range of associated professors, postdoctoral scholars, PhD students, and technical and non-technical staff members. The research focus of a lab is centered around the head's area of specialization. The researchers within a lab collaborate to a larger extent than researchers between labs. Researchers can work either in the lab or in their office. Each researcher has a personal laptop or desktop computer and each lab has a desktop computer. The department has a library equipped with eight desktop computers. The desktop computer in the lab and the desktop computers in the library are shared between researchers. All these computers are connected both by wired LAN and by wireless LAN. Normally, researchers use their personal computers for literature review, documenting research outcomes, preparation of lecture material, personal use (e-mail, browsing, etc.), and so on. The researchers perform experiments in the lab and use the shared computer in the lab for personal use and for accessing biological data. Most of the researchers have CLC Bio Workbench installed in their personal computers and also in the lab computers. Most of the researchers use different versions of

[1] www.inano.au.dk

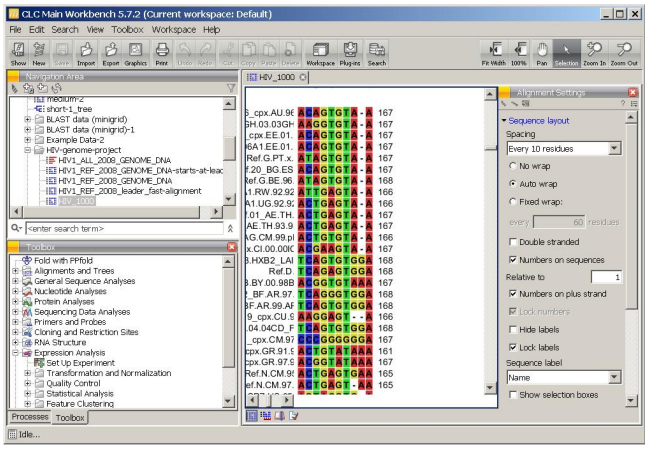


Figure 8.1: The CLC Bio Workbench - Screenshoot.

Windows operating system. However, few researchers also use Mac Os X and different versions of Unix or Unix-like operating system.

8.1 CLC Bio Workbench and Framework Integration

The CLC Bio Workbench creates a software environment enabling users to make a large number of bioinformatics analysis combined with data management and graphical visualization of biological data as shown in Figure 8.1. The software is available for Windows, Mac Os X, and different Unix platforms.

Further, the CLC Bio developer kit API permits integration of various algorithms and workflows within bioinformatics into the workbench as plugins. The CLC Bio Workbench has a complete infrastructure for data management (both locally and through network) that can be utilized by individual plugins as well as an execution model for bioinformatics algorithms.

The Mini-Grid Framework has been integrated with the workbench as one such plugin. Clients participating in the Mini-Grid environment can use this plugin to access the functionality provided by the framework. The plugin abstracts task distribution and context modeling from the algorithm developer.

8.2 Real-World Applications

In this section, we present two bioinformatics algorithms that have been implemented using the integration plugin.

8.2.1 The BLAST Algorithm

Recent advancements in molecular biology techniques permit scientists to gather huge amount of DNA sequence data. Further, these data have been collected and annotated in large sequence databases publicly available at institutions such as NCBI GenBank. By identifying sequences of similar genes or proteins, scientists can find clues to biological function. Thus, extracting information from large databases is a very common and important task for a computational biologist Zomaya [2006].

One of the popular and widely used tools by the bioinformatic community for performing sequence searches is the Basic Alignment Search Tool (BLAST) Altschul et al. [1990], a program to perform pairwise sequence alignments. It uses scoring matrices to compare short subsequences in the query sequence against the entire target or protein sequence database to find statistically significant matches Zomaya [2006]. The algorithm, based on divide-and-concur approaches, recognize the embarrassingly parallel nature of the BLAST searches. The query can be segmented and distributed to each computer node and processed in parallel with the database duplicated on each node. However, for databases too large to fit in the physical memory of each compute node, each query still runs slowly due to virtual memory swapping to hard disks. There is no dependency between tasks in this implementation of the BLAST algorithm. However, in the next section we would look into another algorithm compromising multiple interdependent tasks.

8.2.2 PPfold Algorithm

Pfold implements a "Stochastic Context Free Grammar (SCFG)" designed to produce a "prior probability distribution of RNA structures" for an RNA alignment input. The algorithm consists of three computational intensive parts: phylogenetic calculations, an inside-outside algorithm, and an expectation value calculation algorithm. A parallel version of this algorithm, called PPfold, has been recently parallelized by Zsuzsanna Sukosd et. al. Sükösd et al. [2011].

For the purposes of this study, the task dependencies are very important. In the phylogenetic part of the algorithm, there is no task dependency. Figure 8.2 illustrates the geometry of dependencies among tasks in the inside-outside part. Each parallelogram represents a task, and includes a number of points to be calculated. For example, the computation of the task represented by blue-colored parallelogram depends on the completion of computations of the tasks represented by blue and green parallelograms in Figure 8.2. Dependencies in the expectation value calculation part are analogous to the inside-outside part. The number of divisions in the first row is a parameter of the algorithm. The size of each task depends on the number of divisions.

The algorithm operate over a two dimensional matrix, where the computation of each cell in the matrix depends on the completion of computation in the neighboring three cells that are north, northwest and west cells as shown in Figure 8.3b. This dependency structure would only allow only a single set of

diagonal elements of the matrix to be computed at any time. Thus it creates a pattern of computation that advances diagonally as shown in Figure 8.3a. Thus generation of tasks happens in an asynchronous wavefront form; a new task is generated and queued for immediate execution as soon as its dependencies are completed, which precludes extra waiting time. Further, the number of tasks in each wavefront step varies and hence the number of resource required for computation varies in each wavefront step. Some of the resource may be idle and has an impact on the overhead. Adoption of non-shared memory design makes it possible to run in Mini-Grid. However, this design decision necessitates large amount of data transfer between the resource consumer and the remote resource providers. Hence, data transfer overhead increases significantly.

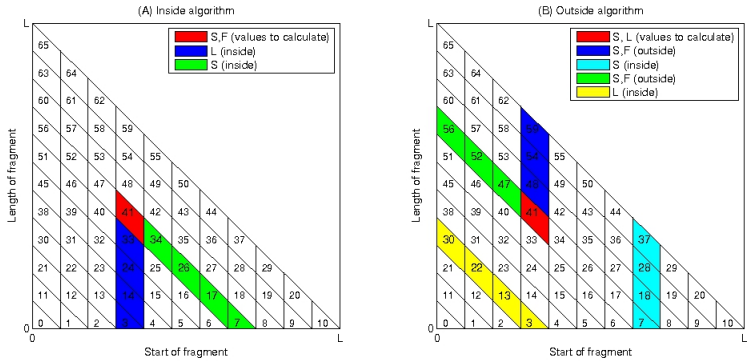
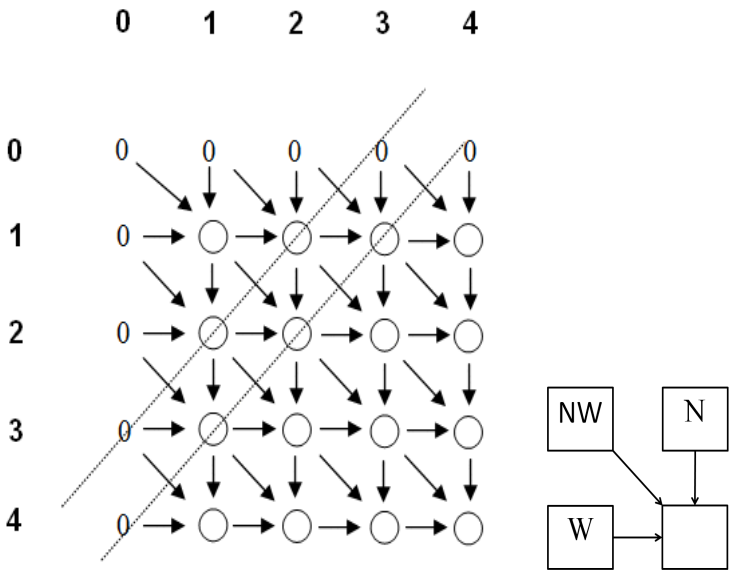


Figure 8.2: Task dependencies in PPfold Sükösd et al. [2011]

8.2.3 CLC Bio Workbench and Application Integration

The PPfold and the BLAST algorithms are developed as separate plugins. These plugins are completely orthogonal to the Mini-Grid framework plugin. These algorithms are not aware of any functionality provided by the framework. They use an abstract mechanism for executing tasks in parallel, and the framework just concentrates on distribution of tasks to the participating clients.

A third integration plugin provides concrete implementation of an execution model that can use the Mini-Grid Framework to distribute the tasks in Mini-Grid. We can say that this integration plugin binds the algorithm with the framework. The integration plugin permits the replacement of PPfold or BLAST with any other parallel version of a bioinformatics algorithm. Similarly, the Mini-Grid framework can be replaced with any other framework that can distribute tasks.



(a) Matrix Wavefront Computation (b) Single Cell

Figure 8.3: Wavefront Application.

8.3 Plugin Installation

The Mini-Grid Framework along with a default configuration file and the implementation of the algorithm gets distributed as an integration plugin. For the first time, the user needs to download the integration plugin from a specific location and install it using the plugin management functionality provided by the workbench. The screenshot in Figure 8.4 depicts the plugin management functionality provided by the workbench. The CLC Workbench then guides the user to navigate through the installation dialogs. Later on, the CLC Workbench detects the availability of a new version of the plugin and automatically upgrades with the user approval.

8.4 Deployment Challenges

Even with the availability of advanced grid programming toolkits, development and deployment of grid applications have been identified as a critical issue Gannon et al. [2003]. Through the extension of Grid to non-dedicated resources, the complexity of the deployment greatly increases. According to Daniel Minoli Minoli [2005], the tasks involved in the deployment of a Grid infrastructure include, but are not limited, to:

- Installation of grid middleware on participating resources.
- Configuration of installed grid middleware for customized use.

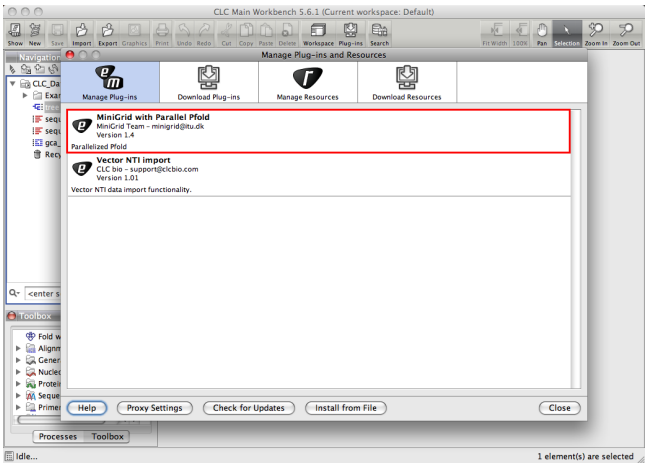


Figure 8.4: Plugin management screen.

- Administrative access on the donor machines to install and configure grid middleware.
- Managing enrollment of donors and users.

Traditionally, a handful of administrators with specialist knowledge manage their Grid infrastructure, configure each resource and pre-install all Grid services which are required von Laszewski et al. [2002]; Friese et al. [2006], and infrastructures are centrally managed Mambretti [2006]. But when a large number of resources are added to the Grid dynamically, central administration is no longer feasible Friese et al. [2006]. With increased adoption of grid application, requiring the end user administrative account on the system for deployment is not feasible Smith et al. [2004].

In the next sections, we discuss our approach to deployment of a parallelized Basic Alignment Search Tool (BLAST) and RNA secondary structure prediction algorithm (PPfold) on Mini-Grid at iNano while addressing the above challenges and solutions.

8.4.1 Deployment Aim

We deploy the Mini-Grid enabled CLC Bio Workbench at iNano by meeting the challenges detailed in Section 8.4 to validate the framework against:

- Support for dynamic resource participation.
- Symmetrical resource sharing.
- Ad hoc grid formation for short duration.
- Support for heterogeneous resource participation.

Future	Deployment#1	Deployment#2
Application	BLAST	PPfold
Type of application	BoT	Wavefront
Load balancing	No	Yes
Context-aware scheduling	No	Yes
Coexistence of multiple versions	No	Yes
Time-to-bid	Constant	Dynamic
Data transfer	No	Yes

Table 8.1: Deployment characteristics.

The parts of the platform exercised during deployment are mapped in Table 8.1.

8.4.2 Deployment Process

The deployment process contained two steps: first deployment and second deployment. Each step had four processes: planning the deployment; deployment execution; observe, patch, package, and update; and feedback for further development.

In the planning process, the stakeholders are consulted to decide on the deployment dates and the responsibilities of individuals. The deployment execution process has been detailed in Section 8.5. During the deployment we observed the system behavior and prepared patches for minor issues and upgraded the system. Major issues served as input to the next iteration of the development process.

In a dynamically changing environment, deployment becomes part of an ad hoc grid application instead of being handled by a system administrator Smith et al. [2004]. The CLC Workbench can detect the availability of a new version of the plugin and can perform automatic updates with the user approval.

8.5 Deployment Execution

The activities involved in the pilot deployment execution process includes: infrastructure setup, resource acquisition, application specific setup and application deployment activity.

The infrastructure set up activity involves installation and configuration of the Mini-Grid Framework. One of the main goals of the framework is to involve minimal configuration. The framework requires four parameters related to communication, for example, the multicast IP address and the port numbers for communication. However, the framework gets distributed with a configuration file containing default values to these parameters. As detailed in Section 8.1, the framework gets distributed as a plugin.

The resource acquisition activity involves recognizing and recruiting the idle resources of users. At the architecture level, Mini-Grid has been designed as a symmetric system, that is, users can not only contribute but also consume. At the practical level, information about deployment and invitation to participation was done through workshops and portal. Like traditional volunteer computing platforms, Mini-Grid does not have the concept of incentive. However, GridOrbit public displays and GridNotify personal notification systems have been used to create awareness Hincapié-Ramos et al. [2011] of the Mini-Grid resource sharing infrastructure in the biology laboratory.

The application specific setup activity involves installing the Mini-Grid application and associated data required for its execution. The framework has been integrated with the CLC Workbench that enables the Mini-Grid application developer to develop his algorithm as a plugin to the CLC Workbench. Further, the workbench provides automated import actions to download the required data from any location. Thus a resource consumer can download the necessary data for computation using this future. However, the actual data distribution among participating resource providers happen through the Mini-Grid task bus.

The application deployment activity involves algorithm description, resource constraints, resource discovery, resource ranking and task execution as detailed in Section 8.5.2.

8.5.1 Application Specific Setup

The application specific setup activity involves the following steps: setting up the CLC Workbench, installing the Mini-Grid plugin, and some basic configuration for the ambient display system. A detailed explanation about installing the CLC Workbench and configuration for the ambient display system is beyond the scope of this thesis. However, we have explained the plugin installation in Section 8.3.

8.5.2 Application Deployment

The application deployment activity involves the following steps: application description, resource constraints, resource discovery, resource ranking, and task execution. The application description involves the end user, that is, the biologist to select an algorithm with associated data and necessary parameters. In the CLC Workbench, first the user needs to select the algorithm as shown in Figure 8.5. Then the user has to a nucleotide sequence, sequence list, or an alignment tree (i.e., the data and input parameters to the algorithm) as shown in Figure 8.6 and in in Figure 8.7. Then has to make a decision on algorithm execution (either locally or in Mini-Grid) as shown in Figure 8.8. And finally the user has to select the mechanism for handling the results as shown in Figure 8.9. These steps explained so far depict the application description step. The application description is the only step which the end user must be involved in.

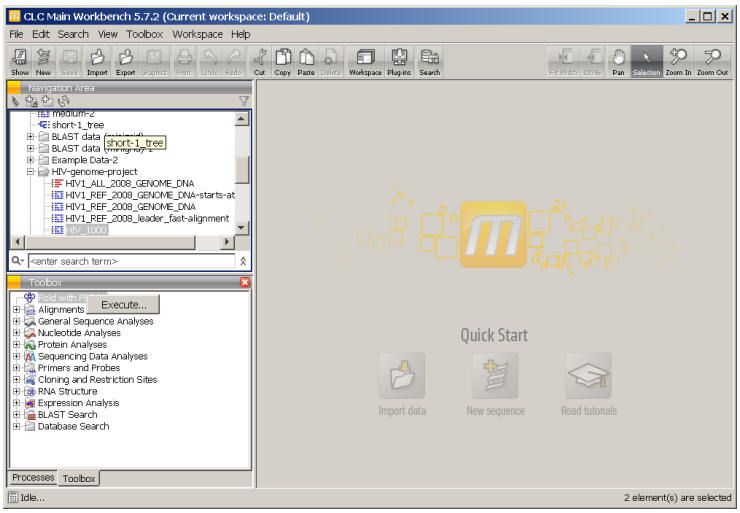


Figure 8.5: Algorithm selection.

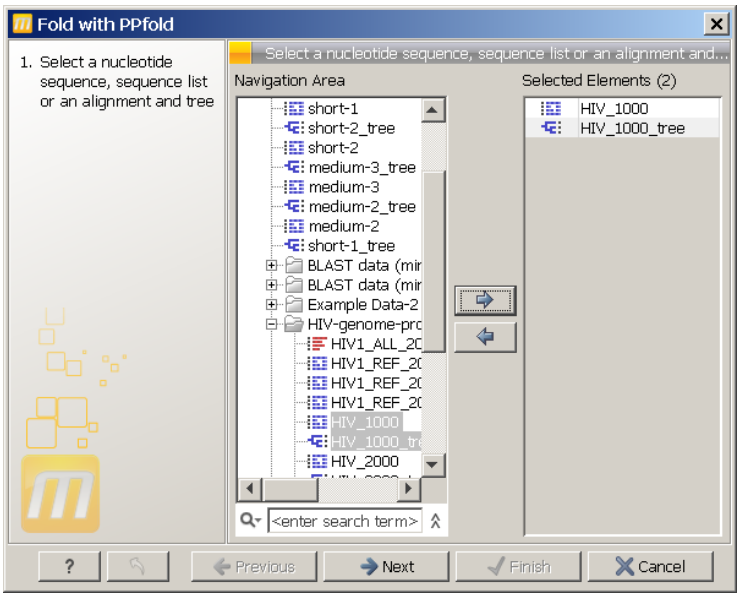


Figure 8.6: Input Data selection.

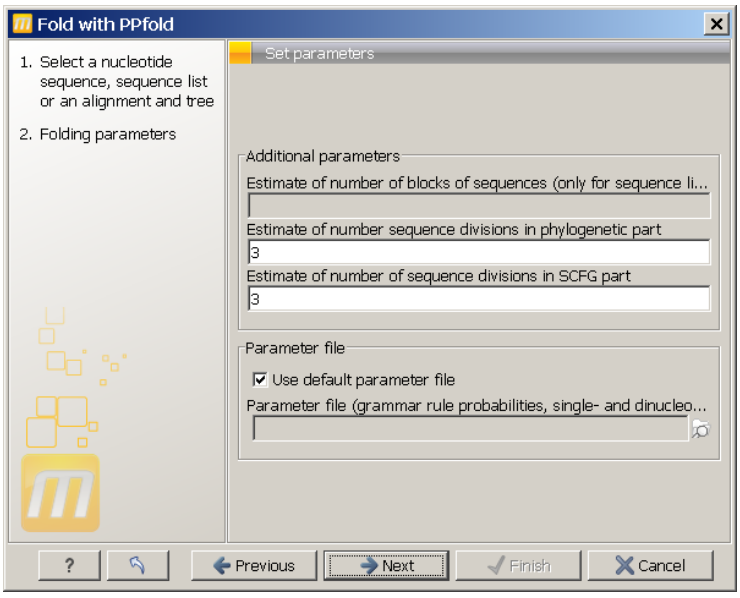


Figure 8.7: Parameter selection.

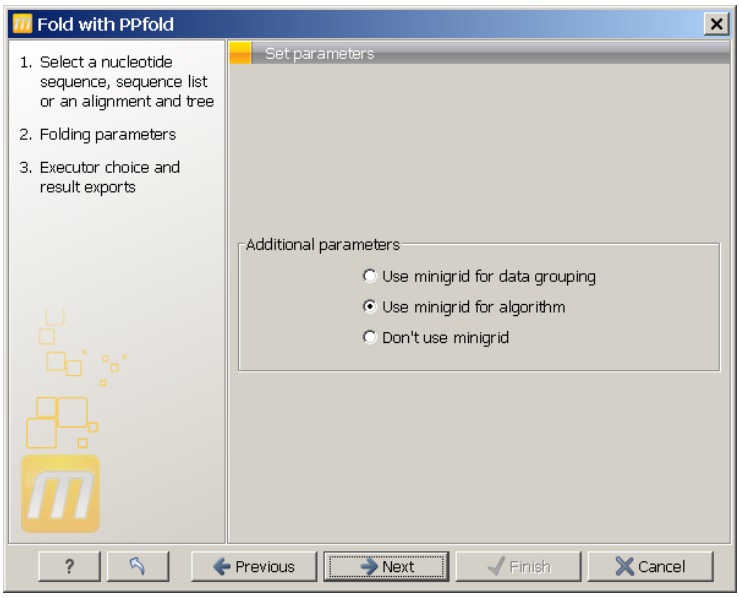


Figure 8.8: Mini-Grid execution.

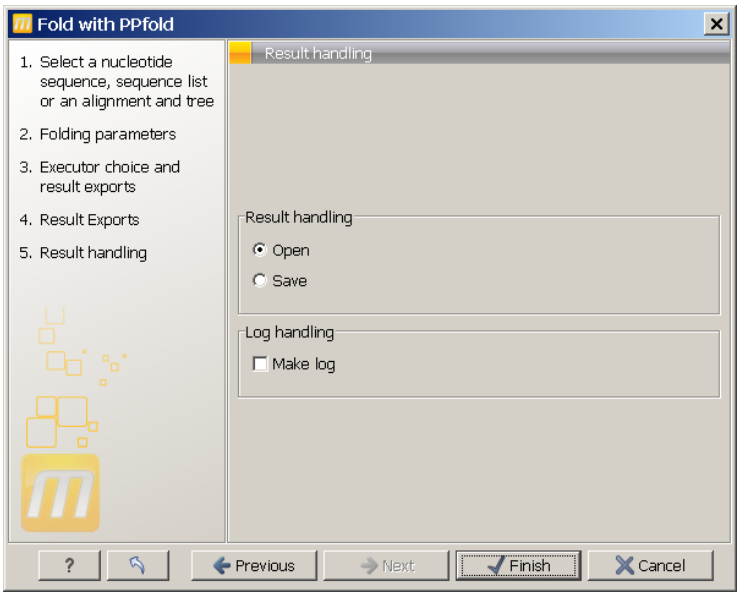


Figure 8.9: Result handling.

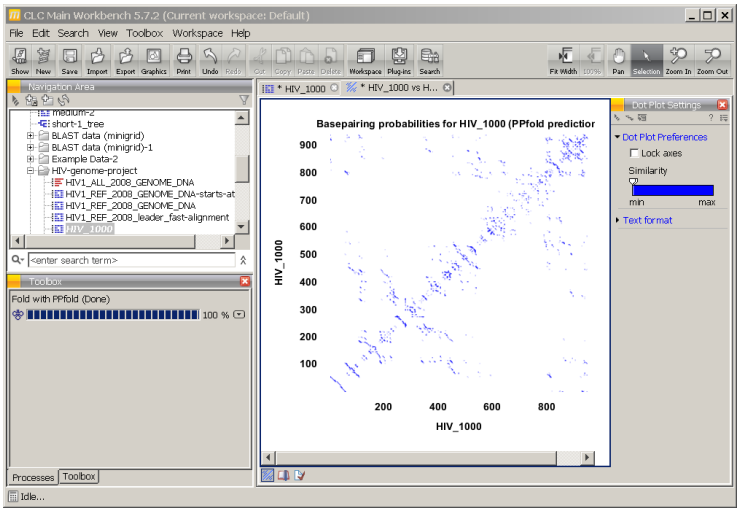


Figure 8.10: Result visualization.

The resource constraints are specified by the Mini-Grid application developer. These are more determined by the nature of the algorithm. For example, PPfold – a wavefront parallel algorithm – generates computational tasks that has certain memory requirements. In the first deployment, we used Basic Alignment Search Tool (BLAST) that finds the region of similarity between sequences and in the second deployment, we used PPfold algorithm explained in Section 8.2.

8.6 Deployment

In this section, we present the pilot deployment of the real-world applications discussed in Section 8.2 by adopting the deployment process detailed in Section 8.4.2.

8.6.1 Deployment Period

The deployment happened at the interdisciplinary Nanoscience center (iNano) at Aarhus University, Aarhus in two phase. We conducted the first phase of the deployment for a period between 31st of May and 4th of July, 2010; and the second phase of the deployment for a period between 3rd of January and 16th of January, 2011. Figure 8.11, shows a user using Mini-Grid enabled CLC Bio Workbench during deployment.

Figure 8.11: User using Mini-Grid enabled CLC Bio Workbench during deployment.

8.6.2 Deployment Environment

The characteristics of the deployment environment are: dynamic and volatile resource participation; and heterogeneous nature of resources.

The resources that participated during the deployment could be classified as shared and personal resources. The shared resource included desktop computers available in library and labs; and the personal resources included desktops and laptops of the biology researchers. The number of non-dedicated resources is much higher than in traditional Grid systems. The Mini-Grid environment encompasses transient individual nodes (e.g., non-dedicated workstations).

The users participating in the deployment typically will try to provide resources to the extent of their possibilities, as they wish to help the community. Users want their resources to be used by the community. However, they still want to retain control of their resources. This will result in intermittent resource availability, as users can decide whether to contribute or not at any moment. Another aspect that must be considered is that participating resources can be heterogeneous in terms of processing capabilities, and configuration. For example, a resource can be single processor or multi-processor with shared or distributed memory.

The run-time of individual instance of an algorithm can range from a few minutes to one or two hours depending on the size of input data and parameters of the algorithm.

8.7 Learning from the Field

In this section, we summarize the lessons learned during the deployment. We have presented the lessons learned during the first deployment detailed in Section 8.7.1 and during the second deployment detailed in Section 8.7.2.

8.7.1 The First Deployment

We were experiencing some form of failure in the first deployment. For example, some software bugs occur only during the deployment. However, in the first deployment both the system developers and the end-user gained better understanding of the real requirements. The important lessons learned during the first deployment are:

- **Deal with virtual IP address:** At iNano, few people used softwares that create virtual IP address, and hence, we needed a solution in the framework for dealing with these virtual IP addresses.
- **Coexistence of multiple versions:** We identified the need of support for coexistence of multiple versions of the framework and algorithm.
- **Need of pre-deployment field test:** The framework exhibits different behavior during the deployment at iNano compared to the testing at ITU.

The difference in network environment at ITU and iNano contributed to the difference in the system behavior.

- **Support for data transfer by the framework:** Initially, we assumed that the data transfer would be done by CLC Bio API. However, we realized the need for having the support for data transfer by the framework independently.
- **Need of dynamic value for time-to-bid:** We assumed that the application would decide on the value for time-to-bid. However, the framework provides default value. But during the deployment we realized that it should be determined dynamically by the framework based on the network and resource load.

8.7.2 The Second Deployment

The second deployment aimed to resolve uncovered and not solved issues during the first deployment. The experience from the first deployment was used to address the weakness of the framework.

To deal with issues with virtual IP addresses, the messenger component now binds to all available interfaces and performs a reachability test to identify correct interfaces for communication. We used the context-awareness framework to deal with coexistence of multiple versions of the framework and plugin. The plugin and the Mini-Grid Framework implementation have version numbers. This metadata has been made part of the context model. Hence, a resource provider participates in the task distribution process only if it had the same framework and plugin versions as requested in the task context. The matching process at the resource provider side ensures this. Pre-deployment test was performed at iNano before actual deployment. The framework has support for data transfer independent of the CLC Developer Kit API. A simple algorithm has been devised to deal with determining the time-to-bid dynamically as explained in Bid Submission subsection of Section 3.5.2.

The following lessons were learned during the second deployment.

- **Need for system stability testing:** The system still needs to be fine tuned for its stability. Currently, we have deployed the system for a short time and we have to ensure that the system can operate on a large scale for several months using stability testing.
- **Remote logging issue:** We had implemented a remote logging facility but it proved to be a potential bottleneck. We had configured a server machine at ITU for pushing the log files periodically and this requires Internet connectivity. During deployment, if network connection from a machine to the ITU server is lost, then the system generates exception messages and the system fails.

8.8 Results and Discussion

In this section, we discuss the results of the first and the second deployments at iNano. We have collected the data manually and through log files. The framework generated log files during the deployment at each resource. The log file data consists of precise traces of each event happening at the client. The log file along with other information provides: number of bids received by a resource consumer during an auction, time taken for conducting the auction, time taken for execution of a task at remote resource providers, time taken for transferring files from resource consumers to resource providers, the usage time of the workbench,and so on. These data have been analyzed and presented here. A wiki page was also maintained during deployment to log in data such as operating system, ip address, processor clock frequency, physical memory available, the owner of the resource and his contact details. The resources were identified by means of a unique identifier generated by the framework automatically the first time when the workbench is used. Regular expressions have been used for extraction of relevant data from log files and Microsoft Excel for data analysis.

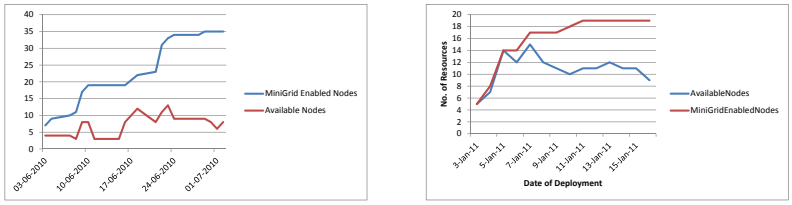
8.8.1 Dynamic Resource Participation

During the deployment, the Mini-Grid Framework supported dynamic resource participation. For example, in the first deployment 35 machines participated and in the second deployment 19 machines participated. The availability of these resources during the period of deployment was also dynamic.

We have presented the resource availability during the first deployment in Figure 8.12a and the resource availability during the second deployment in Figure 8.12b. In Figure 8.12, the X-axis shows the date of deployment and the Y-axis shows the number of resources that were available on a particular date. From the figure, one can observe that even for a fixed number of Mini-Grid enabled resources, their availability varies widely. The participants make volunteer contribution ranging from a few hours a day to entire 24 hours. Thus the availability of the resource is highly dynamic.

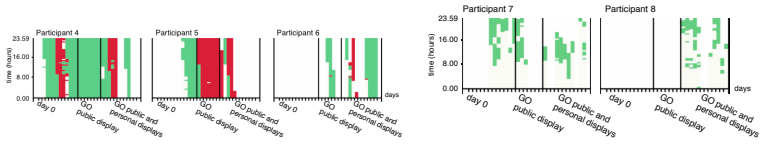
While aggregate resource availability reflects the overall availability of Mini-Grid, it is possible that some hosts are less available than others. We investigate per host availability to reveal any potential imbalance. Figure 8.13 illustrates contribution pattern of personal desktop computers and laptop computers by two example users during the first deployment. In Figure 8.13, the X-axis represents the day of deployment; the Y-axis represents time of a day; and white indicates resource inactive, red indicates available but not contributing resources and green indicates contributing nature of the participating resource. We can observe that resource participation varies highly. However, availability of desktop computers was higher than laptop computers. For more detailed discussion refer to Hincapié-Ramos et al. [2011].

During the deployment, by a rough estimate, we can say that 40-50% of committed volunteers participated. Another observation was that the capacity



(a) First deployment. (b) Second deployment.

Figure 8.12: Host availability.



(a) Desktop Availability. (b) Laptop Availability.

Figure 8.13: Availability of Desktop vs Laptop (adopted from Hincapié-Ramos et al. [2011]).

of Mini-Grid increased by 7% during working hours compared to non-working hours. This in contradiction to resource availability reported in Vladoiu et al. [2009]. They have observed 50-60% resource availabilty during normal and extended working hours and 95-100% resource availability during non-working hours. The other study has considered only laboratory desktop computers of a University compared. Thus the personal resources participating in Mini-Grid contributed to variation in observation. The increase in capacity of Mini-Grid can be attributed to the work habit of participants. Normally, the participants shutdown their computers after working hours or configuring their resources to go into full hibernation mode. In coherence with observations reported in Bhagwan et al. [2003], we also observe steady decrease in total number of resources that are available over subsequent days towards the end of the deployment.

8.8.2 Symmetric Resource Participation

During the first deployment, we observed that out of 35 clients participating in Mini-Grid, 8 clients had played the role of resource consumer by executing at least one instance of the algorithm in the Mini-Grid environment. The architectural design of Mini-Grid as symmetric system led to foster participation through ambient technologies (i.e., GridOrbit and GridNotify) rather than incentives Hincapié-Ramos et al. [2011]. The main motivations for biologists to

join the Mini-Grid are: getting analysis results faster, exploring the potential benefit of the infrastructure, and helping their colleagues.

During the second deployment, we observed that out of 19 clients participating in Mini-Gird environment, only 2 clients played the role of resource consumer. During the second deployment, we could see that 10% of the clients played both roles. On the other hand, during the first deployment we could see that 23% of the clients had played both roles. The BLAST application being the dominant database search tool in molecular biology contributed the increased symmetric resource participation nature during the first deployment. Conversly, only two users were interested in PPfold application and hence others were just contributing. Thus, Mini-Grid enabling a set of useful and possible bioinformatics algorithm could motivate higher user participation. Further, during discussion with Biologists at iNano, it was revealed that other tools having similar functionality to CLC Bio are also being used by the participants. Such tools need to be Mini-Grid enabled for those users to get the benefit. Further, other motivation and awareness techniques can be explored.

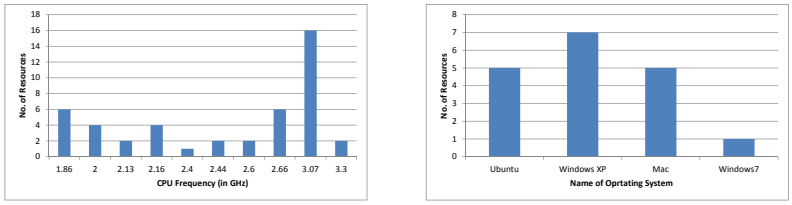
8.8.3 Support for Heterogeneous Resource Participation

The ability of the Mini-Grid Framework to support heterogeneous resource participation can be seen from Figure 8.14. Figure 8.14, illustrates participating resources having different capabilities such as: varied CPU clock frequency; varied physical memory; varied operating systems, and different versions of operating system. Apart from that, the resources had different versions of the plugin and different versions of the framework. Further we observed different types of resource participation during both first and second deployment such as: dedicated desktop computers; non-dedicated office desktop computers; shared desktops in labs and library; and non-dedicated laptop computers. Thus Mini-Grid exhibited heterogeneous resource participation.

The resources participating were heterogeneous in terms of their capability. Figure 8.14a illustrates the distribution of computing power of the resources participating in the second deployment and Figure 8.14b illustrates the distribution of the operating system of the resources participating in the second iteration of the deployment.

8.8.4 Ad hoc Mini-Grid Formation

The Mini-Grid Framework supports a research team to collaborate on a temporary basis to form a small ad hoc grid for a short duration. During the deployment, we observed that one of the participants was able to form an ad hoc grid using resources available in his office. Five desktop computers available in his office were used in this process. They were having different operating systems and different capabilities. He could do that by changing the multicast IP address used to form Mini-Grid. It did not affect the function of the other Mini-Grid deployed at iNano.



(a) CPU Frequency Distribution. (b) OS Distribution.

Figure 8.14: Distribution of CPU Frequency and OS of the Resources.

8.8.5 Deployment Effort

Using documentation available on the BOINC website, we made a qualitative comparison of Mini-Grid to BOINC. We compared the two frameworks using three different measures: the number of steps involved in creating a project, framework requirements and major framework futures. The purpose of the comparison was to determine whether the process of deploying a desktop grid infrastructure was easier using Mini-Grid or using BOINC.

The process involved to create and start a project on a medium to high-power Unix or Linux server has been tabulated in Table 8.2. The BOINC framework consists of two layers which operates under client-server technology, a central scheduling server and a number of clients installed on the volunteer's computers. Due to dependency on other software packages such as MySQL, Apache and PHP, there are several steps involved in the installation of BOINC server. By contrast Mini-Grid has no dependencies on other software packages, and hence there is no need for installation of dependent software packages. The Mini-Grid framework is made available to an application developer as jar files that contains all the class files and related data files that are referenced by the classes. Jar files simplify distributing the framework as a single, compressed file. The application developer has to ensure that they have the necessary jar file in classpath.

To create a functional project, BOINC requires that an *assimilator* be developed. The assimilator component receives canonical result data from BOINC and sends it to a database that is separate from the project database. BOINC requires a project to configure fixed static set of data servers that are maintained for a particular project and makes available for data distribution. An administrator must dedicate time to configure and maintain these data serving machines, which are generally independent for each BOINC project. The Mini-Grid Framework does not require any such component as the results are handled by the application. The framework provides an asynchronous notification method that can be used by the application for collecting the results. Aside

from assimilator component, it is necessary to develop workgenerator component to generate work units and validator component to validates the results. The Mini-Grid Framework does not support result validation, as it aggregates resources from within the organization connected by a local area network. The Mini-Grid Framework provides conceptual abstraction `task` for representing a unit of work that can be distributed over the network to another node for execution. The application developer has to define the tasks that compose the application, as well as the logic ruling their integration. Here, we have considered only a simple task for the purpose of comparison. However, the Mini-Grid Framework provides `context` abstraction for defining the application specific quality of service requirements. The BOINC framework operates on best effort basis and hence does not support any application specific quality of service requirements.

Table 8.2 and table 8.3 show that there are more steps involved in creating a project/application with BOINC than with Mini-Grid. In addition, the steps in the BOINC project creation process tend to be more difficult. One reason is that several of the steps require specific skills like experience with Linux system administration. Another reason is that BOINC does not have a single, unified configuration utility. Instead, users are required to edit XML configuration files in several different locations to create a functional project. Developing components for the BOINC framework requires not only a knowledge of the C++ or FORTRAN programming languages, but also an understanding of the BOINC specific API. All of these factors contribute to the complexity of the BOINC framework. The Mini-Grid Framework requires to edit just one configuration file for modifying the default behavior.

As a client, the BOINC user needs only to download the client software from the BOINC website and register to one of the existing projects. Once the BOINC user runs the client, it allows the user's computer to be included as one of the slaves. BOINC servers use a Apache web server to expose a simple web page offering basic functionalities: user registration, statistics, BOINC client download, etc. The BOINC server must operate user forums related to the project where users can ask questions and their problems.

The Mini-Grid framework, based on peer-to-peer architecture does not have separate server and client software. Using the framework a node can play either resource provider or resource consumer or both roles. The deployment of the Mini-Grid framework needs to download the framework as Java jar file from a website or a central server or can be bundled with the application. It is as simple as deployment of BOINC client software. The project team can maintain a website, for example The Mini-Grid Project for Department of Molecular Biology at Aarhus University[2]. The project website, an important dissemination tools, informs the community about the project and how the users can contribute to the project.

Table reftab:comframe shows a comparison of the requirements and restric-

[2] `http://mbg.medarbejdere.au.dk/en/service-and-facilities/core-facilities/the-Mini-Grid-project-for-department-of-molecular-biology/`

Step	Description
1	Install Python
2	Install PHP
3	Install and configure Apache with the PHP module
4	Install and configure MySQL
5	Download the BOINC source code
6	Compile and install BOINC
7	Develop the science application component in C++ or FORTRAN using the BOINC API
8	Develop the work unit generator component in C++ using the BOINC API
9	Compile the provided ‘ sample trivial validator ’ component (accepts all results as valid)
10	Develop the assimilator component in C++ using the BOINC API
11	Run the `make_project` script
12	Append the BOINC Apache configuration to the system Apache configuration
13	Configure the project by editing the XML configuration file
14	Add the project to the database using the `xadd` scripts
15	Move the MySQL socket to `/var/lib/mysql/mysql.sock`
16	Define a sample application, build the application and copy the executables to the appropriate subdirectory
17	Create an application version for the sample application
18	Sign the sample application with the `sign_executable` program
19	Run the `update_versions` script
20	Add work units to the project work folder at `\$PROJECTROOT/download`
21	Create XML configuration files for each work unit
22	Create XML configuration files for each result
23	Run the `create_work` program to add the work units to the project database
24	Add entries for the work unit generator, feeder, transistor, file deleted, trivial validator, and assimilator to the project configuration file
25	Start the project server
26	Edit the project configuration file to allow users to create accounts

Table 8.2: BOINC Project Creation Process

Step	**Description**
1	Obtain minigrid-framework.jar file and ensure that it is in your class path
2	Create or modify existing ' log4j ' configuration file (if required, otherwise default configuration file is available)
3	Configuration communication end point to be used by the framework (however, default configuration file is available)
4	A resource provider client has to define `ExecutorContext` abstractions (however, default implementations of this abstractions is available)
5	A resource provider client can join the Mini-Grid environment by creating `Executor` object and invoking start method
6	A resource consumer client has to define `Task`,and `TaskContext` (however, default implementation of `TaskContext` is available)
7	A resource consumer has to define `TaskListener` to receive asynchronous notification about current task status
8	A resource consumer client can join the Mini-Grid environment by creating `Submitter` object and invoking start method
9	A resource consumer client can ask Mini-Grid to distribute computation task to the infrastructure by calling `submit()` method of `Submitter`
10	On receiving notification, a resource consumer client can get the results from the `TaskBus`
11	The application integrates the obtained results and has to present them in a user understandable format

Table 8.3: Mini-Grid Application Development and Deployment Process

Requirement	**BOINC**	**Mini-Grid**
Platform	Can function on any platform supported by Java	Server limited to Linux platform; clients are available for Windows, Linux and Mac OS X
Programming language for framework specific components	Java	C++ and optionally FORTRAN wrappers
Database	MySQL	Currently in-memory data structures are used. However, can be extended to use file or database systems
Prerequisites	Apache, Python and PHP	JVM version 1.5 or later

Table 8.4: Comparision of Framework Requirement.

tions of each framework. Platform-independence is an important property of the Mini-Grid Framework as it allows users to select the platform they are most comfortable using. Since our entire framework has been built using Java, both the resource provider and resource consumer can be used on any platform supported by Java. The BOINC server is restricted to the Linux platform. In the BOINC framework, the choice of language in which the various server-side components can be developed are C++ or FORTRAN. The Mini-Grid framework used Java programming language for the component developments. BIONC has database support compared to the Mini-Grid framework. In addition, the Mini-Grid Framework does not have any external programming language or softwares dependencies other than the Java Virtual Machine. However, the Mini-Grid Framework uses Log4j and Jena frameworks for its functioning. Everything required to use the framework is provided along with the framework.

Table 8.5 shows a comparison of the major features of each framework. The features of the framework are quite different, reflecting possible differences in design goals. The features of the Mini-Grid Framework suggests that its goal is making deployment process simple and provides support for application specific quality of service. However, BOINC has the core functionality of a public resource computing. Compared to BOINC deployment, Mini-Grid deployment is simple.

Feature	**BOINC**	**Mini-Grid**
Architecture	Client-server	Peer-to-peer
Result validation	Yes	No
Cross-platform server	No	yes
Cross-platform client	Yes	Yes
Framework components to be implemented	work unit generator, assimilator and validator	Task and Context
Suitable for legacy applications	Yes (but with difficulties)	No (but Java applications can be modified)
Suitable for non-legacy applications	Yes	Yes
Logging support	No	Yes
Configuration management	Using multiple scripts at different location	One single configuration file
QoS support	Best-effort	Application specific

Table 8.5: Comparison of Framework Features.

8.9 Summary

In this chapter, we have described the deployment of a parallel version of BLAST and Pfold algorithm in Mini-Grid at iNano center, Aarhus University, Aarhus. We have discussed the application deployed, the challenges of deployment, deployment process, how the actual deployment happened and learning from the deployment. Finally, we present the results of the deployment and discuss the results.

Part IV

Conclusion

Chapter 9

Conclusion

Desktop grids are attractive platforms for running compute-intensive applications. However, building such systems is complicated as the participating resources are heterogeneous, volatile, do not have adequate security measures and subject to failures. In this thesis, we focused on the development and the deployment of the Mini-Grid Framework for resource management in ad hoc grids using market-based scheduling and context-based resource and application modeling. The Mini-Grid Framework enables the construction and the deployment of ad hoc grids that supports: ease of deployment, decentralized task distribution, smaller scale grid formation, and symmetric resource usage by participants. Furthermore, users can specify non-performance based parameters that influence resource allocation.

The main contributions of this thesis are:

1. Symmetric resource usage: The users can contribute their resources to the Mini-Grid environment as well as use the available resource in the environment.

2. Smaller scale grid formation: The Mini-Grid environment can be created by combining the power of the computers at an institutional level or at an organizational/institutional level.

3. Decentralized task distribution: The Mini-Grid Framework adopts auction strategies for dynamic resource discovery and selection.

4. Resource and task modeling: We have used ontology based context model to describe resource capabilities and to model resource requirements of an application.

5. Application specific quality parameters: The Mini-Grid Framework supports specification of application specific quality parameters for scheduling.

6. Ease of deployment: The Mini-Grid environment can be setup with minimal configuration and installation effort.

In general, we summarize the following main contributions of this thesis:

- **Mapping study of existing desktop grid systems**

  In this thesis, we have presented a taxonomy for desktop grid systems focusing on resource provision, scalability, organization, resource utilization, supported quality-of-service, and deployment effort. We surveyed some of the desktop grid system and mapped them using the taxonomy. On the basis of this taxonomy and mapping, we presented the current state of research in desktop grid systems and we were able to identify the possible research gaps.

- **Context-based auction strategy for dynamic task distribution**

  We used peer-to-peer techniques to implement a desktop grid system that eliminates the need for a centralized scheduling component. We have used a marked-based auction mechanism for dynamic task distribution. Traditionally in price-based mechanism, the price represents supply/demand condition of resources. However, we have used utility function based on application specific quality parameters for representing supply/demand condition. This approach enabled the use of context-based application specific quality parameters for dynamic task distribution. We have shown that a market-based auction mechanism can be used to compensate for absence the central server.

- **Modeling entities such as resources and applications using their context**

  For the resource description model the thesis proposes an alternative to the classic attribute-based symmetric resource description model with an extensible ontology-based context model. This model provides foundation to a flexible and extensible resource discovery and resource selection mechanism. The context information of resources and computational tasks have been modelled as OWL concepts and properties. The context information is stored in a knowledge base and resource requests are expressed using SPARQL queries. Furthermore, we have proposed a template-based approach to model user-application quality of service description or resource capability description for mapping to the corresponding SPARQL query statement. Finally, this approach simplified resource description compared to traditional approach.

- **The Mini-Grid Framework**

  We have shown the feasibility of our approach by implementing these concepts in the Mini-Grid Framework. The framework supports context modeling of resources and computational tasks, and dynamic task distribution based on application specific quality of service parameters. The design choice as framework provides ways to customize the framework for specific needs. The framework supports higher levels of abstraction, hiding complexity of the underlying resource and execution environment

while providing primitives for task scheduling. Efficiency of the framework has been evaluated technically using controlled lab experiments using real-world applications. The performance of the framework has been compared to the Globus and the Entropia systems.

The experimental evaluation shows that scheduling computational-intensive tasks in Mini-Grid scales linearly with increase in number of resources participating in Mini-Grid, provided equal workload partitioning among the participating resources. However, the Mini-Grid Framework introduces overhead due to auctioning process and data transfer between the resource consumer and the resource provider. The overhead due to auction process could be reduced by implementing combinatorial auctions but with increased complexity and communication overhead. The data transfer overhead applies to any desktop grid computing system. We observed that the overhead in Mini-Grid is high compared to the Globus and the Entropia system. However, in the Globus system, there would be additional overhead related to resource selection. Further, the Globus system provides recent but not guaranteed to be absolute latest resource information. Hence, such system may not be suitable for desktop grid system where the resources join and leave the system frequently. The Entropia system performs better compared to Mini-Gird. However, another study reports that in a real deployment there could be additional overhead.

We have evaluated the Mini-Grid Framework for new kind of application, called Wavefront application. These application generate computational tasks in a non-discrete stochastic nature. The Mini-Grid Framework performs better when the individual tasks of the application have higher computational complexity index, a ratio between the execution time and the data transfer time.

- **Deployment at iNano** The framework has been integrated with the CLC Bio Workbench and real-world bioinformatics applications BLAST and PPfold have been integrated. The Mini-Grid enabled CLC Bio Workbench has been deployed at iNano research center to validate the framework against i) support for dynamic resource participation, ii) symmetrical resource sharing, iii) ad hoc grid formation for short usage, and iv) support for heterogeneous resource participation.

  During deployment, we observed that the Mini-Grid Framework supports dynamic resource participation, symmetric resource participation, heterogeneous resource participation, and ad-hoc grid formation. Further, we observed that deployment required minimal effort compared to other popular systems such as BOINC and Globus. BOINC and Globus require non-trival setup efforts from system administrators as shown in Section 8.8.5 and Appendix C.

9.1 Future Work

There are a number of issues related to this work that can become the basis for further research. Some of the most significant future research work are presented here.

- **Fault tolerant scheduling:** Fault tolerant scheduling tolerates failure and volatility. It involves selecting more reliable resources according to availability, volatility, or credibility in order to avoid failures as much as possible, and performing reassignment or replication in the presence of failures or volatility Abbes et al. [2010]; Anglano et al. [2006]; Bardram [2009]. Currently checkpoint, restart, and replication approach are widely adopted for handling fault tolerance. However, proactive fault tolerance mechanism such as relocation or migration would be more useful. Relocation and migration involves relocating the execution from faulty or suspicious resources upon detection of imminent failures. Here suspicious resources mean resource that are expected to fail based on performance analysis. Degradation in performance of a resource could be a sign of resource failure.

- **Trust-based Security:** When we scale the Mini-Grid Framework to Internet scale, then the Mini-Grid Framework need to address security issues. The volunteers to Internet scale desktop systems are at the edge of the Internet. Hence, a trust-based scheduling Sonnek et al. [2006] can be implemented for resource selection. Trust can be parametrized using factors such as volunteering time, execution behavior and so on.

- **Network connectivity:** Another important issue that need to be addressed is the connectivity of the computing resources. The resources are located in different administrative domains and are protected by NATs and Firewalls. In such environments, direct communication between participating resources may not be possible unless relevant configurations are made at gateway softwares such as NATs and networking devices such as routers. Hence, the Mini-Grid Framework need to support mechanism to deal with NATs and Firewalls to establish direct communication between participating resources.

- **Combinatorial Auctions:** Current implementation support Fixed-Price Sealed Bid auction. In this implementation, tasks are auctioned one by one (i.e., sequential). However, multi-unit combinatorial auctions that involves auctioning a set of tasks would be useful. Such multi-unitcombinatorial auctions would reduce overhead.

- **Consistency Management:** Current implementation ensures that the context information supplied by default context monitors are consistent with the defined context model. However, when the application developer extends the framework and define his own context model and context monitors then the developer has to ensure that the model is validated

and the context information provided by the monitor is consistent with the context model. The framework needs to be modified for ensuring consistency checks and conflict detection.

Toward these ends, we believe that the work in this thesis will be a helpful stepping stone for industrialization efforts to make it a production desktop grid.

Part V

Appendices

Appendix A

Mini-Grid Messaging System

As described earlier, messages are the basic unit of data exchanged between clients participating in Mini-Grid environment to provide information about task announcements, bid submissions, etc. This section describes the format of the messages exchanged and the set of rules governing how these messages are exchanged.

A.1 Layers of Messaging System

The Mini-Grid messaging system defines a framework for communication between clients over the network. It defines communication process into three layers, dividing the task involved in moving information between networked clients. Information that need to be transferred from one client to another client proceeds through these layers, as shown in Figure A.1.

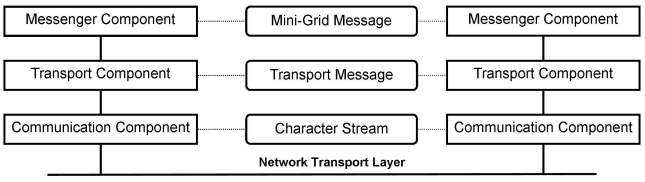


Figure A.1: Layers of messaging system.

From the highest to lowest, these are the *messenger component*, the *transport component* and the *communication component*. These three layers use various forms of control information to communicate with their peer layers in other client. This control information consists of specific requests and instructions

that are exchanged between peer layers. Headers are the basic forms to carry control information and are perpended to data that has been passed down from upper layers.

A.2 Messenger Component

The executor or submitter of a client prepares its data and calls on its local messenger component to send that data as *Mini-Grid message* with required parameters as arguments of the call. The messenger component prepares a Mini-Grid message header and attaches the data to it. It sends the Mini-Grid message to the local transport component to transport the Mini-Grid message to the destination messenger component. The receiving messenger component unwraps the Mini-Grid message and passes the data to the destination executor or submitter. The structure of Mini-Grid message and the protocol are described in more detail in section B.

A.3 Transport Component

The transport component interfaces on one side to messenger component and on the other side to a lower level communication component. The messenger component transmits Mini-Grid message by calling on the transport component with required arguments. The transport component packages the Mini-Grid messages as *transport messages* and calls the local communication component to transport the transport message to the destination transport component. The receiving transport component places the Mini-Grid messages to its local messenger component for processing. To make sure that each client does not process the same transport message more than once, each transport message is identified by a Unique IDentifier (UID). The transport component memorize these UIDs and when a transport layer receives a transport message with a UID it has encountered previously, it simply drops that transport message.

A.4 Communication Component

Communication component has the ability to transfer a stream of characters in each direction between clients. It communicates directly with various types of actual network transport protocols such as User Datagram Protocol (UDP), Transmission Control Protocol (TCP), Secured Socket Layer (SSL), etc.

Communication component is implemented as a wrapper around a socket and serves as an endpoint in a bidirectional communication. If two transport layer components want to communicate, they can obtain an instance of communication component and character streams sent from one can be received at other. Among other things, it has the job of keeping up the underlying socket descriptor, and it automatically closes its descriptor when it is destroyed. It

provides `send()` and `receive()` methods that are wrappers around the `send()` and `recv()` call for the underlying socket descriptor.

Appendix B

The Mini-Grid Protocol

This section describes the Mini-Grid protocol. It consists of a set of Mini-Grid messages for communicating command and control information between messenger component of clients, and a set of rules governing how messages are exchanged. An overview of the various messages is shown in Figure B.1. This figure shows how the various messages relate to each other and how they are packaged to create a Mini-Grid message that is passed around between clients in the Mini-Grid environment.

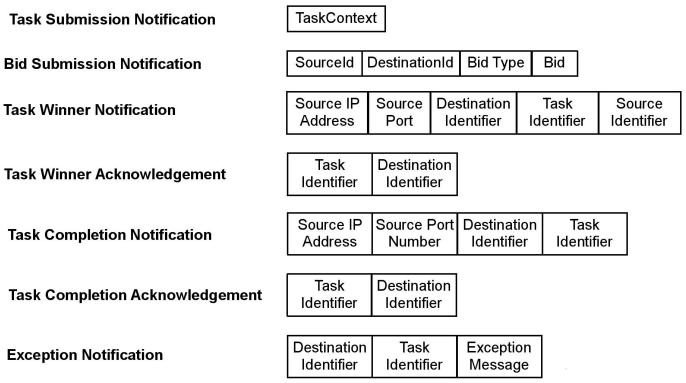


Figure B.1: An overview of messages used to define Mini-Grid protocol

Briefly, a Mini-Grid message consists of a header and a payload. The header includes the common attributes to any payload (i.e., Mini-Grid message type). Then there are seven payloads (i.e., notifications and acknowledgments) that can be exchanged using this protocol. Table B.1 shows the list of messages used in the Mini-Grid protocol and their purpose. Each Mini-Grid message has a message header, which is described in the next section.

Message	Purpose
TaskSubmissionNotification	Announces submission of a task
BidSubmissionNotification	Announces submission of a bid
TaskWinnerNotification	Announces winner of an auction
TaskWinnerAcknowledgement	Acknowledges receipt of task winner notification
TaskCompletionNotification	Announces completion of execution of a task
TaskCompletionAcknowledgement	Acknowledges receipt of task completion notification
ExceptionNotification	Announces occurrence of exception at clients

Table B.1: Mini-Grid Messages and their Usage.

B.1 Mini-Grid Message Header

The Mini-Grid message header consists of:

1. ***Message Type:*** determines the type of client to whom the payload is intended to. The payloads TaskSubmissionNotification, TaskWinnerNotification, and TaskCompletionAcknowledgement are for client acting as resource provider. On the other hand the payloads BidSubmissionNotification, TaskWinnerAcknowledgement, and TaskCompletionNotification are for clients acting as resource consumer. The payload ExceptionNotification can be used both by the resource provider and resource consumer to inform about any exceptions that occur during communication process or during execution of a task at remote resource provider.

2. ***Payload Descriptor:*** contains the type of payload, i.e., TaskSubmissionNotification, BidSubmissionNotification, e.t.c.

Each payload of Mini-Grid message is described in detail in the next sections.

B.1.1 Mini-Grid Message Payloads

TaskSubmissionNotification

TaskSubmissionNotification contains task context information as they are used to announce task submission for execution in the Mini-Grid environment. The header informs that it is a task submission notification and intended for clients acting as resource provider. The task context contains

1. **Resource identifier:** the identifier of the client that has announced task submission.

2. **Bidding strategy:** the strategy used for conducting the bidding process. Normally it contains information about the type of bid expected by the client acting as resource consumer.

3. **Execution environment:** the information about the environment at resource provider necessary for execution of the submitted task.

TaskContext provides generic context information about the task and hence it could contain any context information about the task. However, Mini-Grid only uses the above information for scheduling the task in the Mini-Grid environment.

B.1.2 BidSubmissionNotification

BidSubmissionNotification are send by clients that are ready to execute the announced task. A resource provider should have the requested execution environment and should submit the bid of requested type. BidSubmissionNotification consists of:

1. **Source identifier:** the identifier of the client interested in executing the submitted task.

2. **Destination identifier:** the identifier of the client that has announced the submission of task.

3. **Type of bid:** the type of submitted bid.

4. **Submitted bid:** the bid as determined by the announced bidding strategy and current resource context.

This is a response message to TaskSubmissionNotification. A client can send a response only when it has the requested environment available for execution of the announced task. The notification contains the resource consumer's identifier (*Destination identifier*) that was provided in the TaskSubmissionNotification to ensure that only the resource consumer that announced the task submission would process it.

B.1.3 TaskWinnerNotification

TaskWinnerNotification informs the result of the auctioning process. It is send by resource consumer to the winning resource provider. It notifies the winning resource provider about the award of task execution. This message needs to be acknowledged by the winning resource provider. This message contains:

1. **IP Address:** is the resource consumer's network address.

2. **Port:** is the resource consumer's port for incoming connections to transfer the submitted task.

3. **Destination identifier:** is the resource provider identifier that has won the auction.
4. **Task identifier:** is the identifier of the announced task.
5. **Source identifier:** is the identifier of the resource consumer that announce the task submission.

The notification contains the resource provider's identifier (*Destination identifier*) that was provided in the BidSubmissionNotification to ensure that only the resource provider that won the auction process would process it. *IP Address* and *Port* are optional parameters for specifying communication endpoint of resource consumer. These parameters are required only when the resource provider establishes a direct connection with the resource consumer to download the submitted task.

B.1.4 TaskWinnerAcknowledgement

TaskWinnerAcknowledgement acknowledges the receipt of TaskWinnerNotification by a resource provider. It notifies the resource consumer that it has received the winner notification and is ready to execute the task.

1. **Task identifier:** is the identifier of the task whose execution has been own by resource provider.
2. **Destination identifier:** is the identifier of the resource consumer that has send the TaskWinnerNotification.

The notification signals the end of scheduling operation. The process of making scheduling decisions involves allocating task to suitable resource provider.

B.1.5 TaskCompletionNotification

TaskCompletionNotification informs the resource provider that its task execution at a remote resource provider has been completed. This notification requires an acknowledgment by the receiving task consumer.

1. **IP Address:** is the resource provider's network address.
2. **Port:** is the resource provider's port for incoming connection to transfer the completed task.
3. **Destination identifier:** is the resource consumer identifier that has submitted the task for execution.
4. **Task identifier:** is the identifier of the completed task.

IP Address and *Port* are optional parameters for specifying communication endpoint of resource consumer. These parameters are required only when the resource consumer established a direct connection with the resource provider to download the completed task.

B.1.6 TaskCompletionAcknowledgement

TaskCompletionAcknowledgement acknowledges the receipt of TaskCompletion-Notification by a resource consumer.

1. **Destination identifier:** is the resource provider identifier that has send the TaskCompletionNotification.
2. **Task identifier:** is the identifier of the completed task.

The acknowledgment signals the end of task execution operation at remote resource provider.

B.1.7 ExceptionNotification

ExceptionNotification reports the problems in the Mini-Grid environment to the clients. Notifications can be send both by resource provider and resource consumer to other parties under the following conditions:

- Communication problems such as socket timeout, I/O error when creating the socket, etc.
- Serialization problems when reading from or writing to a socket.
- Exception in task execution at a remote resource provider.

The notification contains:

1. **Destination identifier:** is the identifier of the destination resource provider or consumer to whom this notification is send.
2. **Task identifier:** is the identifier of the task for which exception occurred under the conditions explained earlier.
3. **Exception Message:** contains detailed message that describes the condition under which the exception occurred.

The protocol provides room to inform exceptions that occur under various conditions. However, exception handling has to be dealt by appropriate components in resource provider or resource consumer.

B.1.8 Message Sequence in Operations

We have discussed how the Mini-Grid protocol organizes the information in different payloads. In this section, we are going to discuss how the Mini-Grid operations are performed using the above Mini-Grid messages.

In Mini-Grid environment these messages are used to carry out two operations:

- **Task Scheduling Operation:** is the process of making scheduling decisions.

- **Task Execution Operation:** is the process of informing completion of task execution in remote resource provider.

Upcoming sections explain in more detail how different messages are used in the above two operations using a message sequence diagram.

B.1.9 Task Scheduling Operation

A sequence diagram illustrating the sequence of messages transmitted during task scheduling operation is shown in figure B.2 explaining normal flow. The example assumes that there exists one resource consumer and two resource providers currently participating in the Mini-Grid environment. Further it assumes that the clients acting as resource provider have the capabilities required for execution.

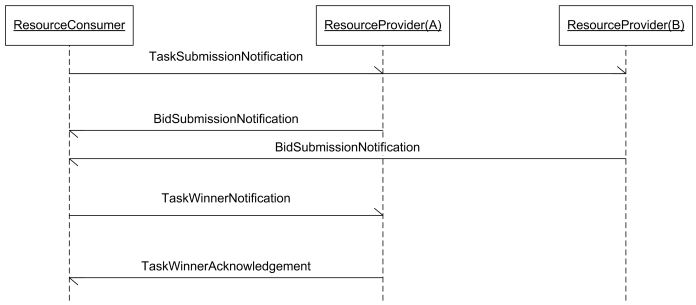


Figure B.2: Message sequence diagram for task scheduling.

As we can see, a client ResourceConsumer sends a TaskSubmissionNotification announcing the availability of a task for execution. Two clients ResourceProvider(A) and ResourceProvider(B) on receiving the announcement, since they have the requested capabilities, express their interest in executing the task by replying with a BidSubmissionNotification. The resource consumer selects ResourceProvider(A) based on its task distribution algorithm and informs it by sending a TaskWinnerNotification. The ResourceProvider(A) acknowledges the receipt of TaskWinnerNotification by sending a TaskWinnerAcknowledgement.

B.1.10 Task Execution Operation

A sequence diagram illustrating the sequence of messages transmitted during task execution operation is shown in figure B.3 explaining normal flow. The example assumes that the ResourceProvider(A) has completed the execution of task without any problem. On completion of execution of the task, the ResourceProvider(A) sends a TaskCompletionNotification and once the ResourceConsumer that submitted the task receives the notification sends an acknowledgment.

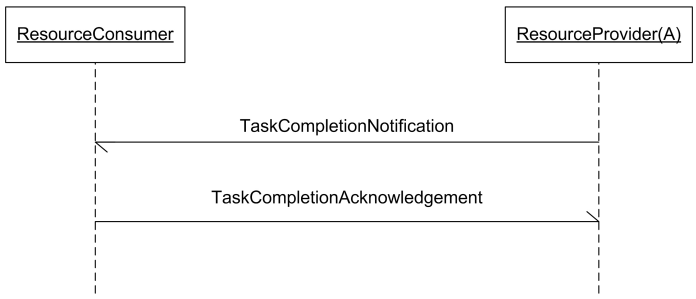


Figure B.3: Message sequence diagram for task execution.

Appendix C

Installation of Globus - An Experience

Here, we provide the experience of John P. Morrison et. al., in installing Globus Toolkit Version 4 (GT4) during deployment of WebCom-G Information System over GT4 Morrison et al. [2004].

> Version 4 of Globus Toolkit requires a UNIX-based operating system for installation. Hence, it cannot be installed on Windows system without emulation of UNIX specific function. This is not exactly in line with the requirement that a grid should support as many environment as possible. Installation of Globus Toolkit is more complex than the installation of the Web-service environment Apache Tomcat: First, for the installation, the toolkit requires the configuration of the desired location. Afterwards, GT4 runs without security options. Second, to enable authentication and authorization, SSL certificates need to be created. Third, description files, containing the name of the grid and information about the security policies, need to be created. These files are required on each node participating in the grid. Finally, the standard services need to be configured, e.g., one needs to specify the address of a directory service for publishing grid services. When following these steps, we observed several problems: Using all standard services proposed, Globus Toolkit throws a lot of errors on start up. The reason is that the services are not configured correctly. Further, errors concerning security arise, even when the platform is started following the second step, i.e., security is deactivated. We also had problems of different behavior of the middleware on different systems. We encountered different, uncommented errors on different UNIX distributions. The complete installation requires a lot editing of configuration files with the correct parameters.

Bibliography

Rdf vocabulary description language 1.0: Rdf schema. Technical report, W3C, 2004. URL `http://www.w3.org/TR/2004/REC-rdf-schema-20040210/`.

Heithem Abbes, Christophe Cerin, and Mohamed Jemni. A decentralized and fault-tolerant desktop grid system for distributed applications. *Concurr. Comput. : Pract. Exper.*, 22:261–277, March 2010. ISSN 1532-0626.

Khalid Abdelkader, Jan Broeckhove, and Kurt Vanmechelen. Economic-based resource management for dynamic computational grids: Extension to substitutable cpu resources. In El Mostapha Aboulhamid and José Luis Sevillano, editors, *AICCSA*, pages 1–6. IEEE, 2009. ISBN 978-1-4244-3807-5.

Nabil Abdennadher and Régis Boesch. Towards a peer-to-peer platform for high performance computing. In *Proceedings of the 2nd international conference on Advances in grid and pervasive computing*, GPC'07, pages 412–423, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-72359-2.

David Abramson, Rajkumar Buyya, and Jonathan Giddy. A computational economy for grid computing and its implementation in the nimrod-g resource brok. *CoRR*, CS.DC/0111048, 2001.

Devlic Alisa. Sip-based context distribution: does aggregation pay off? *SIGCOMM Comput. Commun. Rev.*, 40(5):35–46, October 2010. ISSN 0146-4833.

M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 2581 (Draft Standard), September 2009. URL `http://www.ietf.org/rfc/rfc5681.txt`.

S.F. Altschul, W. Gish, W.Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

Balachandar R. Amarnath, Thamarai Selvi Somasundaram, Mahendran Ellappan, and Rajkumar Buyya. Ontology-based grid resource management. *Softw., Pract. Exper.*, 39(17):1419–1438, 2009.

Kaizar Amin, Gregor von Laszewski, and Armin R. Mikler. Toward an architecture for ad hoc grids. *IEEE 12th Int. Conf. on Advanced Computing and Communications, ADCOM 2004*, December 2004.

Yair Amir, Baruch Awerbuch, and Ryan S. Borgstrom. The Java market: Transforming the Internet into a metacomputer. Technical Report CNDS-98-2, The Johns Hopkins University, 1998.

David P. Anderson. BOINC: a system for public-resource computing and storage. In *GC '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 365–372, New York, NY, USA, 2004. ACM Press.

David P. Anderson, Eric Korpela, and Rom Walton. High-performance task distribution for volunteer computing. In *Proceedings of the First International Conference on e-Science and Grid Computing*, E-SCIENCE '05, pages 196–203, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2448-6.

David P. Anderson, Carl Christensen, and Bruce Allen. Grid resource management—designing a runtime system for volunteer computing. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 126, New York, NY, USA, 2006. ACM Press. ISBN 0-7695-2700-0. doi: http://doi.acm.org/10.1145/1188455.1188586.

Nazareno Andrade, Lauro Costa, Guilherme Germoglio, and Walfredo Cirne. Peer-to-peer grid computing with the ourgrid community. In *Proceedings of the 23rd Brazilian Symposium on Computer Networks*, May 2005.

Sergio Andreozzi, Stephen Burke, Felix Ehm, Laurence Field, Gerson Galang, Balazs Konya, Maarten Litmaath, Paul Millar, and J. P. Navarro. GLUE Specification v. 2.0. Technical report, Open Grid Forum, March 2009. URL `http://www.ogf.org/documents/GFD.147.pdf`.

Cosimo Anglano, John Brevik, Massimo Canonico, Dan Nurmi, and Rich Wolski. Fault-aware scheduling for bag-of-tasks applications on desktop grids. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, GRID '06, pages 56–63, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 1-4244-0343-X.

Cosimo Anglano, Massimo Canonico, and Marco Guazzone. The sharegrid peer-to-peer desktop grid: Infrastructure, applications, and performance evaluation. *Journal of Grid Computing*, 8:543–570, 2010. ISSN 1570-7873.

Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, and Darren Pulsipher. Job submission description language (jsdl) specification, version 1.0. Technical Report GFD-R.056, Open Grid Forum, 2007.

Gabriel Antoniu, Mathieu Jan, and David A. Noblet. Enabling the p2p jxta platform for high-performance networking grid infrastructures. In Laurence T. Yang, Omer F. Rana, Beniamino Di Martino, and Jack Dongarra, editors, *High Performance Computing and Communcations*, volume 3726 of *Lecture Notes in Computer Science*, pages 429–439. Springer Berlin / Heidelberg, 2005.

Knudsen B and Hein J. Rna secondary structure prediction using stochastic context-free grammars and evolutionary history. *Bioinformatics*, 15:446–454, 1999.

Knudsen B and Hein J. Pfold: Rna secondary structure prediction using stochastic context-free grammars. *Nuc Acids Res*, 31:3423–3428, 2003.

Mark Baker and Rajkumar Buyya. Cluster computing at a glance. In Rajkumar Buyya, editor, *High Performance Cluster Computing: Architectures and Systems, Volume 1*, pages 3–47. Prentice Hall PTR, 1999.

Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Softw. Pract. Exper.*, 32:1437–1466, December 2002. ISSN 0038-0644.

Z. Balaton, G. Gombas, P. Kacsuk, A. Kornafeld, J. Kovacs, A.C. Marosi, G. Vida, N. Podhorszki, and T. Kiss. Sztaki desktop grid: a modular and scalable way of building large computing grids. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1 –8, march 2007.

James D. Baldassari. Design and evaluation of a public resource computing framework. Master's thesis, Worcester Polytechnic Institue, Worcester, MA, USA, May 2006.

A. Baratloo, M. Karaul., H. Karl, and Z. M. Kedem. Knittingfactory: An infrastructure for distributed web applications. Technical Report TR 1997-748, New York, NY, USA, November 1997.

A. Baratloo, M. Karaul, Z. M. Kedem, and P. Wijckoff. Charlotte: metacomputing on the web. *Future Gener. Comput. Syst.*, 15:559–570, October 1999. ISSN 0167-739X.

Jakob E. Bardram. The Java Context Awareness Framework (JCAF) - A Service Infrastructure and Programming Framework for Context-Aware Applications. In Hans Gellersen, Roy Want, and Albrecht Schmidt, editors, *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005)*, volume 3468 of *Lecture Notes in Computer Science*, pages 98–115, Munich, Germany, May 2005a. Springer Verlag.

Jakob E. Bardram. Tutorial for the Java Context Awareness Framework (JCAF), version 1.5. Technical report, Centre for Pervasive Healthcare, Aarhus, Denmark, 2005b. Available from http://www.daimi.au.dk/ bardram/jcaf/.

Jakob E. Bardram. Design, Implementation, and Evaluation of the Java Context Awareness Framework (JCAF). Technical report, Centre for Pervasive Healthcare, Aarhus, Denmark, 2005c. Available from http://www.daimi.au.dk/ bardram/jcaf/.

Jakob E. Bardram. The contingency management framework version 1.0. Technical Report IT University Technical Report Series TR-2009-121, IT University of Copenhagen, Denmark, December 2009.

Blaise Barney. Introduction to parallel computing homepage. URL `https://computing.llnl.gov/tutorials/`.

Jyoti Batheja and Manish Parashar. A framework for opportunistic cluster computing using javaspaces. In *Proceedings of the 9th International Conference on High-Performance Computing and Networking*, HPCN Europe 2001, pages 647–656, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42293-5.

Joseph Bauer. *Identification and Modeling of Contexts for Different Information Scenarios in Air Traffic*. PhD thesis, Technische Universitat Berlin, Berlin, Germany, 2003.

Pourebrahimi Behnaz and Bertels Koen. Auction protocols for resource allocations in ad-hoc grids. In *Proceedings of the 14th international Euro-Par conference on Parallel Processing*, Euro-Par '08, pages 520–533, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85450-0.

Rémi Bertin, Sascha Hunold, Arnaud Legrand, and Corinne Touati. From Flow Control in Multi-path Networks to Multiple Bag-of-tasks Application Scheduling on Grids. Rapport de recherche RR-7745, INRIA, Sep 2011.

Ranjita Bhagwan, Stefan Savage, and Geoffrey M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on*, volume 2735 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin / Heidelberg, February 2003.

Guy E. Blelloch. Programming parallel algorithms. *Commun. ACM*, 39(3): 85–97, 1996.

John Brooke, Donal Fellows, Kevin Garwood, and Carole Goble. Semantic matching of grid resource descriptions. In *In Proceedings of the European Across Grids Conference, 2004*, pages 240–249. Springer, 2004.

Thomas Buchholz and Michael Schiffers. Quality of context: What it is and why we need it. In *In Proceedings of the 10th Workshop of the OpenView University Association: OVUA03*, 2003.

Paul D. Buck. Unofficial boinc wiki: Overview of daemons, 2005. URL `www.boinc-wiki.info/BOINC_Server-Side_Daemon_Program`.

David Butler. Gridft server simple performance measurements. Technical Report BBC Research White Paper WHP 178, British Broadcasting Corporation, January 2010.

R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. In M. Parashar and C. Lee, editors, *Proceedings of the IEEE*, volume 93 of *Special Issue on Grid Computing*, pages 698–714. IEEE Press, New Jersey, USA, Mar 2005.

Rajkumar Buyya and Sudharshan Vazhkudai. Compute power market: Towards a market-oriented grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, pages 574–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1010-8.

Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15): 1507–1542, 2002.

Neil Cafferkey, Philip D. Healy, David A. Power, and John P. Morrison. Job management in webcom. In *ISPDC*, pages 25–30. IEEE Computer Society, 2007.

Charlie Catlett, Pete Beckman, Dane Skow, and Ian Foster. Creating and operating national-scale cyberinfrastructure services, May 2006. URL `http://www.ctwatch.org/quarterly/print.php%3Fp=35.html`.

A.J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(3):373 – 384, May 2005. ISSN 1083-4427. doi: 10.1109/TSMCA.2005.846396.

Matthew Chalmers. A historical view of context. *Comput. Supported Coop. Work*, 13(3-4):223–247, August 2004. ISSN 0925-9724.

Steve J. Chapin, Dimitrios Katramatos, John F. Karpovich, and Andrew S. Grimshaw. The legion resource management system. In *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS/SPDP '99/JSSPP '99, pages 162–178, London, UK, 1999. Springer-Verlag. ISBN 3-540-66676-1. URL `http://portal.acm.org/citation.cfm?id=646380.689541`.

Kyle Chard, Kris Bubendorfer, and Peter Komisarczuk. High occupancy resource allocation for grid and cloud systems, a study with drive. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 73–84, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-942-8. doi: http://doi.acm.org/10.1145/1851476.1851486. URL `http://doi.acm.org/10.1145/1851476.1851486`.

Harry Chen, Filip Perich, Timothy W. Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *MobiQuitous*, pages 258–267. IEEE Computer Society, 2004. ISBN 0-7695-2208-4.

Keith Cheverst, Keith Mitchell, and Nigel Davies. Design of an object model for a context sensitive tourist guide. In *Computers and Graphics*, pages 883–891, 1999.

Andrew A. Chien. Architecture of a commercial enterprise desktop grid: The entropia system. In F. Berman, G. Fox, and T. Hey, editors, *Making the*

Global Infrastructure a Reality, pages 337 – 350. John Wiley and Sons Ltd.,, Chichester, UK, 2003.

Andrew A. Chien, Shawn Marlin, and Stephen T. Elbert. Resource management in the entropia system. In Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, editors, *Grid resource management*, pages 431–450. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1-4020-7575-8.

Chia-Hung Chien, Paul Hsueh-Mtn Chang, and Von-Wun Soo. Market-oriented multiple resource scheduling in grid computing environments. In *Proceedings of the 19th International Conference on Advanced Information Networking and Applications - Volume 1*, AINA '05, pages 867–872, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2249-1. doi: http://dx.doi.org/10.1109/AINA.2005.239. URL `http://dx.doi.org/10.1109/AINA.2005.239`.

SungJin Choi and Rajkumar Buyya. Group-based adaptive result certification mechanism in desktop grids. *Future Generation Comp. Syst.*, 26(5):776–786, 2010.

SungJin Choi, HongSoo Kim, EunJung Byun, and ChongSun Hwang. A taxonomy of desktop grid systems focusing on scheduling. Technical Report KU-CSE-2006-1120-02, Department of Computer Science and Engineering, Korea University, Seong gbuk-gu, Seoul, 2006.

SungJin Choi, HongSoo Kim, EunJoung Byun, MaengSoon Baik, SungSuk Kim, ChanYeol Park, and ChongSun Hwang. Characterizing and classifying desktop grid. In *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on*, pages 743 –748, May 2007. doi: 10.1109/CCGRID.2007.31.

Bernd O. Christiansen, Peter R. Cappello, Mihai F. Ionescu, Michael O. Neary, Klaus E. Schauser, and Daniel Wu. Javelin: Internet-based parallel computing using java. *Concurrency - Practice and Experience*, 9(11):1139–1160, 1997.

Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, and Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and e-business applications. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 151–159, Washington, DC, USA, 2007. IEEE Computer Society. ISBN 0-7695-3064-8.

Walfredo Cirne, Francisco Brasileiro, Nazareno Andrade, Lauro Costa, Alisson Andrade, Reynaldo Novaes, and Miranda Mowbray. Labs of the World, Unite!!! *Journal of Grid Computing*, 4(3):225–246, 2006. doi: http://doi.ieeecomputersociety.org/10.1007/s10723-006-9040-x.

F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the multics system. In *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*, AFIPS '65 (Fall, part I), pages 185–196, New York, NY, USA, 1965. ACM. doi: 10.1145/1463891.1463912.

George F Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321263545.

Celso Maciel da Costa, Marcelo da Silva Strzykalski, and Guy Bemard. A reflective middleware architecture to support adaptive mobile applications. In *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, pages 1151–1154, New York, NY, USA, 2005. ACM. ISBN 1-58113-964-0.

DAG. Directed Acyclic Graph Manager Homepage. http://www.cs.wisc.edu/condor/dagman/.

Nurmi Daniel, Brevik John, and Wolski Rich. Modeling machine availability in enterprise and wide-area distributed computing environments. In Jos Cunha and Pedro Medeiros, editors, *Euro-Par 2005 Parallel Processing*, volume 3648 of *Lecture Notes in Computer Science*, pages 612–612. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-28700-1.

de Almeida Damiao Ribeiro, Baptista Claudio de Souza, and de Andrade Fabio Gomes. Using ontologies in context-aware applications. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 349–353, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2641-1.

D. de Roure, M. Baker, N. R. Jennings, and N. Shadbolt. The evolution of the grid, 2003.

Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, 2001. ISSN 1617-4909. doi: http://dx.doi.org/10.1007/s007790170019.

Distributed.net. distributed.net homepage. http://www.distributed.net/.

Samir Djilali, Thomas Herault, Oleg Lodygensky, Tangui Morlier, Gilles Fedak, and Franck Cappello. Rpc-v: Toward fault-tolerant rpc for internet connected desktop grids with volatile nodes. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 39–, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2153-3.

Feitelson Dror and Nitzberg Bill. Job characteristics of a production parallel scientific workload on the nasa ames ipsc/860. In Dror Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 337–360. Springer Berlin / Heidelberg, 1995. ISBN 978-3-540-60153-1.

Derek L Eager, Edward D Lazowska, and John Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *Performance Evaluation*, 6(1):53 – 68, 1986. ISSN 0166-5316.

Howell F and McNab R. Sim java: A discrete event simulation package for java with applications in computer systems modelling. *1st International Conference on Webbased Modelling and Simulation*, 1998.

G. Fedak, C. Germain, V. Neri, and F.Cappello. XtremWeb: a generic global computing system. In *Proceedings of First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 582–587. IEEE Press, 2001.

Gilles Fedak. Recent advances and research challenges in desktop grid and volunteer computing. In Frdric Desprez, Vladimir Getov, Thierry Priol, and Ramin Yahyapour, editors, *Grids, P2P and Services Computing*, pages 171–185. Springer US, 2010. ISBN 978-1-4419-6794-7.

Gilles Fedak, Haiwu He, Oleg Lodygensky, Zoltan Balaton, Zoltan Farkas, Gabor Gombas, Peter Kacsuk, Robert Lovas, Attila Csaba Marosi, Ian Kelley, Ian Taylor, Gabor Terstyanszky, Tamas Kiss, Miguel Cardenas-Montes, Ad Emmen, and Filipe Araujo. Edges: A bridge between desktop grids and service grids. In *Proceedings of the The Third ChinaGrid Annual Conference (chinagrid 2008)*, CHINAGRID '08, pages 3–9, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3306-3.

Marco Ferrante. The jxta way to grid: a dead end? JXTA/Grid survey slide (2008), available at `http://www.disi.unige.it/person/FerranteM/papers/JXTA-survey.pdf`, 2008.

Ian Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, 2006.

Ian Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *In 2nd International Workshop on Peer-to-Peer Systems (IPTPS03*, pages 118–128, 2003.

Ian Foster and Carl Kesselman. Computational grids. pages 15–51, 1999.

Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001. ISSN 1094-3420. doi: 10.1177/109434200101500302.

James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, July 2002. ISSN 1386-7857. doi: 10.1023/A:1015617019423. URL `http://portal.acm.org/citation.cfm?id=592899.593005`.

T. Friese, M. Smith, and B. Freisleben. Service-oriented ad hoc grids. In Thomas Barth and Anke Schll, editors, *Grid Computing*, pages 143–190. Vieweg, 2006. ISBN 978-3-8348-0033-6. doi: 10.1007/978-3-8348-9101-3_8. URL `http://dx.doi.org/10.1007/978-3-8348-9101-3_8`.

Mohammad Muztaba Fuad. *An autonomic software architecture for distributed applications*. PhD thesis, Bozeman, MT, USA, June 2007.

Nathalie Furmento, Jeffrey Hau, William Lee, Steven Newhouse, and John Darlington. Implementations of a service-oriented architecture on top of jini, jxta and ogsi. In Marios D. Dikaiakos, editor, *Grid Computing*, volume 3165 of *Lecture Notes in Computer Science*, pages 249–261. Springer Berlin / Heidelberg, 2004.

Fabrizio Gagliardi, Bob Jones, Mario Reale, and Stephen Burke. European datagrid project: Experiences of deploying a large scale testbed for e-science applications. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 480–500, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44252-9.

Dennis Gannon, Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan, and Aleksander Slominski. *Grid Web Services and Application Factories*, pages 251–264. John Wiley and Sons, Ltd, 2003. ISBN 9780470867167. doi: 10.1002/0470867167.ch9. URL `http://dx.doi.org/10.1002/0470867167.ch9`.

gLite. glite - lightweight middleware for grid computing homepage. http://glite.cern.ch/.

Jacek Gomoluch and Michael Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Proceedings of the First International Workshop on Middleware for Grid Computing. Rio de*, pages 211–218, 2003.

GPT. Grid Packaging Tools Homepage. http://grid.ncsa.illinois.edu/gpt/.

Brusa Graciela, Caliusco Maria Laura, and Chiotti Omar. A process for building a domain ontology: an experience in developing a government budgetary ontology. In Orgun Mehmet A. and Meyer Thomas, editors, *Second Australasian Ontology Workshop (AOW 2006)*, volume 72 of *CRPIT*, pages 7–15, Hobart, Australia, 2006. ACS.

H. Gümüskaya, A. V. Gürel, and M. V. Nural. Architectures for small mobile communication devices and performance analyses. In *International Conference on the Applications of Digital Information and Web Technologies*, pages 342–347, Ostrava, Czech Republic, August 2008.

Alastair Hampshire and Gordon Blair. Jgrid: Exploiting jini for the development of grid applications. In Nicolas Guelfi, Egidio Astesiano, and Gianna Reggio, editors, *Scientific Engineering for Distributed Java Applications*, volume 2604 of *Lecture Notes in Computer Science*, pages 132–142. Springer Berlin / Heidelberg, 2003.

K. A. Hawick and H. A. James. A java-based parallel programming support environment. In *Proceedings of the 8th International Conference on High-Performance Computing and Networking*, HPCN Europe 2000, pages 363–372, London, UK, 2000. Springer-Verlag. ISBN 3-540-67553-1.

Juan David Hincapié-Ramos, Aurlien Tabard, and Jakob Bardram. Gridorbit – an awareness system supporting the adoption of a volunteer computing infrastructure. In *CHI '11: Proceedings of the 29th of the international conference on Human factors in computing systems*, New York, NY, USA, 2011. ACM.

Yan Huang. Jisga: A jini-based service-oriented grid architecture. *Int. J. High Perform. Comput. Appl.*, 17:317–327, August 2003. ISSN 1094-3420.

Soonwook Hwang and Carl Kesselman. A flexible framework for fault tolerance in the grid. *J. Grid Comput.*, 1(3):251–272, 2003.

Adriana Iamnitchi and Ian T. Foster. A problem-specific fault-tolerance mechanism for asynchronous, distributed systems. In *ICPP*, pages 4–14, 2000.

Brevik J., Nurmi D., and Wolski R. Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. In *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, CCGRID '04, pages 190–199, Washington, DC, USA, 2004. IEEE Computer Society.

Bart Jacob. Grid computing: What are the key components? IBM developer works, June 2003. URL `http://www-106.ibm.com/developerworks/grid/library/gr-overview/`.

Bart Jacob, Luis Ferreira, Norbert Bieberstein, Candice Gilzean, Jean-Yves Girard, Roman Strachowski, and Seong (Steve) Yu. *Enabling Applications for Grid Computing with Globus*. IBM Red Books, June 2003. ISBN 073845333.

Bart Jacob, Michael Brown, Kentaro Fukui, and Nihar Trivedi. *Introduction to Grid Computing*. IBM Red Books, December 2005. ISBN 0738494003.

Case J.D., Fedor M., Schoffstall M.L., and Davin J. Simple Network Management Protocol (SNMP). RFC 1157 (Historic), May 1990. URL `http://www.ietf.org/rfc/rfc1157.txt`.

Kerry Jean, Alex Galis, and Alvin Tan. Context-aware grid services: Issues and approaches. In *International Conference on Computational Science*, pages 166–173, 2004.

Hai Jin. Challenges of grid computing. In Wenfei Fan, Zhaohui Wu, and Jun Yang, editors, *Advances in Web-Age Information Management*, volume 3739 of *Lecture Notes in Computer Science*, pages 25–31. Springer Berlin / Heidelberg, 2005.

Wingstrom Joshua and Casanova Henri. Statistical Modeling of Resource Availability in Desktop Grids. Technical report, University of Hawaii atManoa, Honolulu, U.S.A., Nov 2007.

Z. Juhasz, A. Andics, and S. Pota. Jm: A jini framework for global computing. In *Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on*, page 395, may 2002a.

Zoltan Juhasz, Arpad Andics, Krisztian Kuntner, and Szabolcs Pota. Towards a robust and fault-tolerant multicast discovery architecture for global computing grids. In *In Proceedings of the 4th DAPSYS workshop*, pages 74–81. Desprez, 2002b.

Henricksen Karen, Indulska Jadwiga, and Rakotonirainy Andry. Generating context management infrastructure from high-level context models. In *4th International Conference on Mobile Data Management, Melbourne, Australia*, 2003.

S. Kent, P. Broadbent, N. Warren, and S.R. Gulliver. On-demand hd video using jini based grid. In *Multimedia and Expo, 2008 IEEE International Conference on*, pages 1045 –1048, April 2008.

Jik-Soo Kim, Beomseok Nam, Michael A. Marsh, Peter J. Keleher, Bobby Bhattacharjee, Derek Richardson, Dennis Wellnitz, and Alan Sussman. Creating a robust desktop grid using peer-to-peer services. In *IPDPS*, pages 1–7. IEEE, 2007.

Paul Klemperer. How (not) to run auctions: The european 3g telecom auctions. CEPR Discussion Papers 3215, C.E.P.R. Discussion Papers, 2002.

Paul Klemperer. *Auctions: Theory and Practice*, chapter 1. Princeton University Press, 2004.

Melvin Koh, Jie Song, Liang Peng, and Simon See. Service Registry Discovery using GridSearch P2P Framework. *Proceeding of CCGrid*, 2:11, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2006.166.

Derrick Kondo. *Scheduling Task Parallel Applications for Rapid Turnaround on Desktop Grids*. PhD thesis, University of California, San Diego, 2005.

Derrick Kondo, Andrew A. Chien, and Henri Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *SC*, page 17. IEEE Computer Society, 2004. ISBN 0-7695-2153-3.

Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Softw. Pract. Exper.*, 32:135–164, February 2002. ISSN 0038-0644.

H. Kurdi, M. Li, and H. Al-Raweshidy. A classification of emerging and traditional grid systems. *Distributed Systems Online, IEEE*, 9(3):1–1, March 2008.

Mcguinness Deborah L. and van Harmelen Frank. OWL web ontology language overview. W3C recommendation, W3C - World Wide Web Consortium, February 2004. URL `http://www.w3.org/TR/owl-features/`.

Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.*, 1:169–182, August 2005. ISSN 1574-1702.

Spyros Lalis and Alexandros Karipidis. Jaws: An open market-based framework for distributed computing over the internet. In Rajkumar Buyya and Mark Baker, editors, *Grid Computing GRID 2000*, volume 1971 of *Lecture Notes in Computer Science*, pages 87–106. Springer Berlin / Heidelberg, 2000.

Laszewski, Warren Smith, and Steven Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, HPDC '97, pages 365–, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-8117-9. URL `http://portal.acm.org/citation.cfm?id=822082.823138`.

Gregor von Laszewski, Eric Blau, Michael Bletzinger, Jarek Gawor, Peter Lane, Stuart Martin, and Michael Russell. Software, component, and service deployment in computational grids. In *Proceedings of the IFIP/ACM Working Conference on Component Deployment*, pages 244–256, London, UK, 2002. Springer-Verlag. ISBN 3-540-43847-5.

Young Choon Lee and Albert Y. Zomaya. On effective slack reclamation in task scheduling for energy reduction. *JIPS*, 5(4):175–186, 2009.

Capra Licia, Emmerich Wolfgang, and Mascolo Cecilia. Reflective middleware solutions for context-aware applications. In *Proceedings of the Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, REFLECTION '01, pages 126–133, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42618-3.

M.J. Litzkow, M. Livny, and M.W. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111. IEEE Press, 1988.

Lu Liu and Nick Antonopoulos. From client-server to p2p networking. In Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon, editors, *Handbook of Peer-to-Peer Networking*, pages 71–89. Springer US, 2010. ISBN 978-0-387-09751-0.

O. Lodygensky, G. Fedak, F. Cappello, V. Neri, M. Livny, and D. Thain. Xtremweb & condor sharing resources between internet connected condor pools. In *Proceedings of the 3st International Symposium on Cluster Computing and the Grid*, CCGRID '03, pages 382–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1919-9.

Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal. Alchemi: A .net-based enterprise grid computing system. In Hamid R. Arabnia and Rose Joshua, editors, *International Conference on Internet Computing*, pages 269–278. CSREA Press, 2005. ISBN 1-932415-69-6.

Amdahl Gene M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), pages 483–485, New York, NY, USA, 1967. ACM.

Ausubel Lawrence M. An Efficient Ascending-Bid Auction for Multiple Objects. *The American Economic Review*, 94(5), 2004. ISSN 00028282.

Grüninger M. and Fox M. Methodology for the Design and Evaluation of Ontologies. In *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing*, April 1995.

P Nixon MA Razzaque, S Dobson. Categorization and modelling of quality in context information. In *Proceedings of the IJCAI 2005 Workshop on AI and Autonomic Communications*, 2005.

T. Malone, R. Fikes, K. Grant, and M Howard. *Enterprise: A market-like task scheduler for distributed computing environments*, pages 177–205. North-Holland, 1988.

Joe Mambretti. The grid and grid network services. In Franco Travostino, Joe Mambretti, and Gigi Karmous-Edwards, editors, *Grid Networks: Enabling Grids with Advanced Communication Technology*, pages 1–16. John Wiley and Sons, Ltd.,, 2006. ISBN 0-470-01748-1.

Csaba Attila Marosi, Gabor Gombás, Zoltán Balaton, Péter Kacsuk, and Tamás Kiss. Sztaki desktop grid: Building a scalable, secure platform for desktop grid computing. In Marco Danelutto, Paraskevi Fragopoulou, and Vladimir Getov, editors, *CoreGRID Workshop - Making Grids Work*, pages 365–376. Springer, 2007. ISBN 978-0-387-78447-2.

Timothy Mattson, Beverly Sanders, and Berna Massingill. *Patterns for parallel programming*. Software Patterns Series. Addison-Wesley Professional, first edition, 2004. ISBN 0321228111.

Jim Maurer. A conversation with david anderson. *Queue*, 3(6):18–25, July 2005. ISSN 1542-7730.

mgr inż. Marcin Cieślak. Boinc on jxta - suggestions for improvements. Technical report, Poland, June 2007.

Daniel Minoli. Grid system deployment issues, approaches, and tools. In *A Networking Approach to Grid Computing*, pages 201–331. John Wiley and Sons, Inc.,, Hoboken, NJ, USA, 2005. ISBN 0-471-68756-1.

Alberto Montresor, Hein Meling, and zalp Babaoglu. Messor: Load-balancing through a swarm of autonomous agents. In Gianluca Moro and Manolis Koubarakis, editors, *Agents and Peer-to-Peer Computing*, volume 2530 of *Lecture Notes in Computer Science*, pages 125–137. Springer Berlin / Heidelberg, 2003.

John P. Morrison, Brian C. Clayton, David A. Power, and Adarsh Patil. Webcom-g: Grid enabled metacomputing. *Neural Parallel & Scientific Comp.*, 12:419–438, 2004.

Michael Neary, Alan Phipps, Steven Richman, and Peter Cappello. Javelin 2.0: Java-based parallel computing on the internet. In Arndt Bode, Thomas Ludwig, Wolfgang Karl, and Roland Wismller, editors, *Euro-Par 2000 Parallel Processing*, volume 1900 of *Lecture Notes in Computer Science*, pages 1231–1238. Springer Berlin / Heidelberg, 2000.

Michael O. Neary, Sean P. Brydon, Paul Kmiec, Sami Rollins, and Peter Cappello. Javelin++: scalability issues in global computing. In *Proceedings of the ACM 1999 conference on Java Grande*, JAVA '99, pages 171–180, New York, NY, USA, 1999a. ACM. ISBN 1-58113-161-5.

Michael O. Neary, Bernd O. Christiansen, Peter Cappello, and Klaus E. Schauser. Javelin: parallel computing on the internet. *Future Gener. Comput. Syst.*, 15:659–674, October 1999b. ISSN 0167-739X.

N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet-the popcorn project. In *Distributed Computing Systems, 1998. Proceedings. 18th International Conference on*, pages 592 –601, May 1998.

R. Olejnik, E. Laskowski, B. Toursel, M. Tudruj, and I. Alshabani. Dg-adaj: a java computing platform for desktop grid. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Cracow Grid Workshop 2005 Proceedings*, Cracow, Poland, April 2006. Published by Academic Computer Centre CYFRONET AGH. ISBN 1-58113-161-5.

Zhelong Pan, Xiaojuan Ren, Rudolf Eigenmann, and Dongyan Xu. Executing mpi programs on virtual machines in an internet sharing system. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 101–101, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 1-4244-0054-6.

R. Mantovaneli Pessoa, C. Z. Calvi, J.G. Pereira Filho, C.R. Guareis de Farias, and R. Neisse. Semantic context reasoning using ontology based models. In A. Pras and M.J. van Sinderen, editors, *Dependable and Adaptable Networks and Services, 13th Open European Summer School and IFIP TC6.6 Workshop (EUNICE)*, volume LNCS 4606 of *Lectures Notes in Computer Science*, pages 44–51, Germany, July 2007. Springer Verlag.

Serge Petiton, Lamine Aouad, and Laurent Choy. Peer to peer large scale linear algebra programming and experimentations. In Ivan Lirkov, Svetozar Margenov, and Jerzy Wasniewski, editors, *Large-Scale Scientific Computing*, volume 3743 of *Lecture Notes in Computer Science*, pages 430–437. Springer Berlin / Heidelberg, 2006.

Tran Vu Pham, Lydia MS Lau, and Peter M Dew. An adaptive approach to p2p resource discovery in distributed scientific research communities. *Proceeding of CCGrid*, 2:12, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2006.119.

Alper Pinar, Corcho Oscar, Kotsiopoulos Ioannis, Missier Paolo, Bechhofer Sean, and Goble Carole. S-OGSA as a Reference Architecture for OntoGrid and for the Semantic Grid. In *GGF16 Semantic Grid Workshop*, 2006.

Line Pouchard, Luca Cinquini, Bob Drach, Don Middleton, David E. Bernholdt, Kasidit Chanchio andIan T. Foster andVeronika Nefedova andDavid Brown, Peter Fox, Jose Garcia, Gary Strand, Dean Williams, Ann L. Chervenak, Carl Kesselman, Arie Shoshani, and Alex Sim. An ontology for scientific information in a grid environment:the earth system grid. In *CCGRID*, pages 626–632, 2003.

Eric Prud́hommeaux and Andy Seaborne. SPARQL query language for RDF, 2008. URL `http://www.w3.org/TR/rdf-sparql-query/`.

R. Raman, M. Livny, and M. Solomon. Matchmaking: distributed resource management for high throughput computing. *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*, pages 140–146, July 1998. ISSN 1082-8907.

Rajesh Raman, Marvin Solomon, Miron Livny, and Alain Roy. The classads language. In Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, editors, *Grid resource management*, pages 255–270. Kluwer Academic Publishers, Norwell, MA, USA, 2004. ISBN 1-4020-7575-8.

Ala Rezmerita, Vincent Neri, and Franck Cappello. Toward Third Generation Internet Desktop Grids. Technical Report RT-0335, INRIA, 2007. URL `http://hal.inria.fr/inria-00148923/PDF/RT-0335.pdf`.

RSL. The Globus Resource Specification Language RSL v1.0 . http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html, 2007.

Robert G Sargent. Verification and validation of simulation models. *Science Education*, 17(2):161–103, 1995.

Luis F. G. Sarmenta and Satoshi Hirano. Bayanihan: building and studying web-based volunteer computing systems using java. *Future Gener. Comput. Syst.*, 15:675–686, October 1999. ISSN 0167-739X.

Hirano Satoshi. Horb: Distributed execution of java programs. In Takashi Masuda, Yoshifumi Masunaga, and Michiharu Tsukamoto, editors, *Worldwide Computing and Its Applications*, volume 1274 of *Lecture Notes in Computer Science*, pages 29–42. Springer Berlin / Heidelberg, 1997.

Bill Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 85–90. IEEE Computer Society, 1994.

B. Schnizler. *Resource Allocation in the Grid : A Market Engineering Approach.* Dissertation, Universität Karlsruhe (TH), 2007. ISBN: 978-3-86644-165-1.

Christian Schönberg and Burkhard Freitag. Evaluating rdf querying frameworks for document metadata. Technical Report MIP-0903, University of Passau, 2009.

Jennifer M. Schopf, Mike ĎArcy Neill Miller, Laura Pearlman, Ian Foster, and Carl Kesselman. Monitoring and discovery in a web services framework: Functionality and performance of the globus toolkit's mds4. Technical Report ANL/MCS-P1248-0405, Argonne National Laboratory, April 2005.

Jahanzeb Sherwani, Nosheen Ali, Nausheen Lotia, Zahra Hayat, and Rajkumar Buyya. Libra: a computational economy-based job scheduling system for clusters. *Softw. Pract. Exper.*, 34:573–590, May 2004. ISSN 0038-0644.

João Nuno Silva, Lu Veiga, and Paulo Ferreira. nuboinc: Boinc extensions for community cycle sharing. *Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 248–253, October 2008.

Matthew Smith, Thomas Friese, and Bernd Freisleben. Towards a service-oriented ad hoc grid. In *Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, ISPDC '04, pages 201–208, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2210-6.

Jason Sonnek, Mukesh Nathan, Abhishek Chandra, and Jon Weissman. Reputation-based scheduling on unreliable distributed infrastructures. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ICDCS '06, pages 30–37, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2540-7.

Florian Stegmaier, Udo Gröbner, Mario Dö ller, Harald Kosch, and Gero Baese. Evaluation of current rdf database solutions. In *Proceedings of* 10^{th} *International Workshop of the Multimedia Metadata Community on Semantic Multimedia Database Technologies*, SeMuDaTe09, 2009.

Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, Nottingham/England*, 2004.

Z Sükösd, B Knudsen, M Vaerum, J Kjems, and E S Andersen. Multithreaded comparative rna secondary structure prediction using stochastic context-free grammars. *BMC Bioinformatics*, 12, 2011.

Domenico Talia and Paolo Trunfio. A p2p grid services-based protocol: Design and evaluation. In M. Danelutto, D. Laforenza, and M. Vanneschi, editors, *Proceedings of Euro-Par 2004*, volume 3149 of *Lecture Notes in Computer Science*, pages 1022–1031. Springer Verlag, 2004.

Andrew Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd ed.)*. Prentice Hall, 2007.

Hongsuda Tangmunarunkit, Stefan Decker, and Carl Kesselman. Ontology-based resource matching in the grid - the grid meets the semantic web. In *In Proceedings of the Second International Semantic Web Conference, Sanibel-Captiva Islands*. Springer, 2003.

Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In Thomas Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

Ian J. Taylor and Andrew Harrison. *From P2P and Grids to Services on the Web: Evolving Distributed Communities*. Computer Communications and Networks. Springer Publishing Company, Incorporated, 2nd edition, 2009. ISBN 9781848001220.

Yong Meng Teo and Xianbing Wang. Alice: A scalable runtime infrastructure for high performance grid computing. In Hai Jin, Guang R. Gao, Zhiwei Xu, and Hao Chen, editors, *NPC*, volume 3222 of *Lecture Notes in Computer Science*, pages 101–109. Springer, 2004. ISBN 3-540-23388-1.

Douglas Thain and Miron Livny. Multiple bypass: Interposition agents for distributed computing. *Cluster Computing*, 4(1):39–47, March 2001. ISSN 1386-7857.

Berners-Lee Tim, Hendler James, and Lassila Ora. The semantic web. *Scientific American*, May 2001.

P. Trunfio, D. Talia, H. Papadakis, P. Fragopoulou, M. Mordacchini, M. Pennanen, K. Popov, V. Vlassov, and S. Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Gener. Comput. Syst.*, 23:864–878, August 2007. ISSN 0167-739X.

George Tsouloupas and Marios D. Dikaiakos. Characterization of computational grid resources using low-level benchmarks. In *Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '06, pages 70–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2734-5. doi: http://dx.doi.org/10.1109/E-SCIENCE.2006.36. URL `http://dx.doi.org/10.1109/E-SCIENCE.2006.36`.

Kurt Vanmechelen. A performance and feature-driven comparison of jini and jxta frameworks. Master's thesis, University of Antwerp, Antwerpen, Belgium, 2003.

Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Comput. Surv.*, 38, June 2006. ISSN 0360-0300.

Dinesh C. Verma. *Legitimate Peer to Peer Network Applications: Beyond File and Music Swapping*. Wiley-Interscience, 2004. ISBN 0471463698.

Monica Vladoiu and Zoran Constantinescu. An extended master worker model for a desktop grid computing platform (qadpz). In *ICSOFT 2008 - Proceedings of the Third International Conference on Software and Data Technologies*, pages 169–174. INSTICC Press, 2008.

Monica Mihaela Vladoiu, Zoran Constantinescu, and Catalina Negoita. Availability of computational resources for desktop grid computing. Bulletin of PG University, MIF Series 1, Petroleum-Gas University of Ploiesti (UPG), Romania, 2009.

Gregor von Laszewski, Warren Smith, Steven Tuecke, Steven Fitzgerald, Ian Foster, and Carl Kesselman. A directory service for configuring high-performance distributed computations. *High-Performance Distributed Computing, International Symposium on*, 0:365, 1997. ISSN 1082-8907. doi: http://doi.ieeecomputersociety.org/10.1109/HPDC.1997.626445.

Gregor von Laszewski, Eric Blau, Michael Bletzinger, Jarek Gawor, Peter Lane, Stuart Martin, and Michael Russell. Software, component, and service deployment in computational grids. In Judith Bishop, editor, *Component Deployment*, volume 2370 of *Lecture Notes in Computer Science*, pages 79–103. Springer Berlin / Heidelberg, 2002.

Jim Waldo. *The Jini Specifications*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000. ISBN 0201726173.

Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Trans. Softw. Eng.*, 18:103–117, February 1992. ISSN 0098-5589.

Xiao Hang Wang, Tao Gu, Da Qing Zhang, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *IEEE International*

Conference on Pervasive Computing and Communication (PerCom04), pages 18–22, 2004.

Zhuozhi Wang and Kaizhong Zhang. Rna secondary structure prediction. In Tao Jiang, Ying Xu, and Michael Q. Zhang, editors, *Current Topics in Computational Molecular Biology*, pages 345–363. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-10092-2.

Marek Wieczorek, Stefan Podlipnig, Radu Prodan, and Thomas Fahringer. Applying double auctions for scheduling of workflows on the grid. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 27:1–27:11, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. URL `http://portal.acm.org/citation.cfm?id=1413370.1413398`.

Allcock William, Bresnahan John, Kettimuthu Rajkumar, Link Michael, Dumitrescu Catalin, Raicu Ioan, and Foster Ian. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 54–64, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 1-59593-061-2.

R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.

Peter R. Wurman, William E. Walsh, and Michael P. Wellman. Flexible double auctions for electionic commerce: theory and implementation. *Decis. Support Syst.*, 24(1):17–27, November 1998. ISSN 0167-9236.

Lijuan Xiao, Yanmin Zhu, Lionel M. Ni, and Zhiwei Xu. Gridis: An incentive-based grid scheduling. In *IPDPS*. IEEE Computer Society, 2005. ISBN 0-7695-2312-9.

Wei Xing, Marios D. Dikaiakos, and Rizos Sakellariou. A core grid ontology for the semantic grid. *Cluster Computing and the Grid, IEEE International Symposium on*, 0:178–184, 2006. doi: http://doi.ieeecomputersociety.org/10.1109/CCGRID.2006.3.

Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *J. Grid Comput.*, 3(3-4):171–200, 2005.

Yuanyuan Zhang, Wei Sun, and Yasushi Inoguchi. Predict task running time in grid environments based on cpu load predictions. *Future Generation Comp. Syst.*, 24(6):489–497, 2008.

Liu Zhong, Dou Wen, Zhang Wei Ming, and Zou Peng. Paradropper: A general-purpose global computing environment built on peer-to-peer overlay network. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCSW '03, pages 954–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1921-0. URL `http://portal.acm.org/citation.cfm?id=839280.840612`.

Albert Y. Zomaya. *Parallel computing for bioinformatics and computational biology : models, enabling technologies, and case studies / edited by Albert Y. Zomaya.* John Wiley & Sons, Hoboken, N.J. :, 2006. ISBN 9780471718482 0471718483.