A Configurable Job Submission and Scheduling System for the Grid

Jeevak Kasarkod

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

July 28, 2003
Blacksburg, Virginia

Keywords: Grid Resource Brokers, Resource Management Systems, Matchmaking, User Level Middleware, Design Patterns, Advance Reservation

A Configurable Job Submission and Scheduling System for the Grid

Jeevak Kasarkod

Abstract

Grid computing provides the necessary infrastructure to pool together diverse and distributed resources interconnected by networks to provide a unified virtual computing resource view to the user. One of the important responsibilities of the grid software is resource management and techniques to allow the user to make optimal use of the resources for executing applications. In addition to the goals of minimizing job completion time and achieving good throughput there are other minimum requirements such as minimum memory and cpu requirements, choice of operating system, fine grained file access permissions etc. Currently such requirements are being fulfilled by resource brokers, which act as mediating agents between users and resource owners.

In this thesis we approach the resource brokering architectural issue in a different manner. Instead of a monolithic broker, which performs all the superscheduling functions we propose a Modular Framework based Architecture for Task Initiation and Scheduling (MFATIC) based on the three main stages in the superscheduling process.

There are three major goals of this research. The first aim is to develop a decoupled architectural model that not only provides a clear distinction in the responsibilities of each of the components but also provides the user the flexibility to replace one component with another functionally equivalent component. Secondly each of these components should be configurable and extensible to be able to accommodate user requirements. Finally, the design should enable the user to plug in modules within components of different deployments of the resource broker and thus promoting software reuse.

Table of Contents

List Of Figures

List of Tables

Chapter 1. Introduction

Grid computing [1] infrastructures are unified collections of computing resources interconnected by high speed networks. This infrastructure is heterogeneous, dynamic and highly scalable. Computing resources differ in hardware components as well as software infrastructure such as operating systems and data and resource management systems. Resources may join and leave the infrastructure without prior notification. The number of resources constituting the grid is scalable to thousands of computing nodes.

A grid computing system, or grid system, runs programs on the grid to form a unified system of interacting resources. A set of programs facilitate interaction with other resources in the system. This collection of programs managing the resource interaction is called grid system middleware because it forms a software layer above the native operating system. The grid system middleware manages the underlying resource heterogeneity, and provides the abstraction of a unified system. A grid infrastructure has multiple grid system users. The combination of the grid fabric and the middleware enables seamless computing across the Internet. A grid system user can run applications on various grid resources by running the application on top of the middleware layer. Different local scheduling and security policies for different domains, in addition to multiple users and varied application requirements, create a complex web of issues to be handled by the grid resource management system. The grid resource management system needs to provide schedulers which determine schedules based upon application requirements, local scheduling and security policies.

The idea of grid computing is very appealing. The grand vision for grid computing is to provide computing power as a utility to every household. It is compared to the electric grid where electricity is distributed in similar distribution grid infrastructures. *Utility computing* poses a number of challenges ranging from commoditization of resources to security and resource management issues. Commoditization of resources is dependent on the quality of service provided by the system. The ability to specify and guarantee quality of service levels need to be achieved before utility computing can be a reality. In order to achieve fine grain quality of service levels, it is necessary to improve middleware services for the grid. Currently the majority of the effort in the grid computing world is being invested in grid middleware. This thesis intends to contribute to this ongoing effort in the area of resource management for grid systems.

1.1 Thesis

The future of grid computing systems is a world wide grid with decentralized services for tapping into computing power. The grid resource management system needs to be ubiquitously available and should be flexible to accommodate multiple users, computing environments and varying application requirements. Secondly the user should be able to control every phase of the resource selection process if he desires to without violating security and local scheduling policies. In this thesis we describe a **Modular,**

Framework-based Architeture for Task Initialization and sCheduling (MFATIC) on the grid which is designed to meet these requirements. The goal for MFATIC is the following :

- Provide a decoupled task initiation and scheduling architecture

  The process of executing a job on the grid involves resource discovery, resource selection, job startup and monitoring. This is followed by post processing steps of transferring data back for result analysis. In this thesis we describe MFATIC, a decoupled system for task initiation and scheduling. As a result of the system being decoupled the user can control all the phases of the task initiation process and enable the required components. The interface developer is provided a choice of minimal requirements matching policies, scheduling policies and job submission mechanisms. This idea of pluggable modules fits into the idea of a world wide grid with decentralized scheduling mechanisms where every user with different application requirements can build a customized task scheduling and initiation system.

- Provide configurability and extensibility for each phase of the scheduling and initiation process

  The results for each of the phases of the scheduling and initiation process can be achieved in multiple ways and the user should have a choice. In current scenarios the grid system user utilizes the services of a grid broker for resource selection. In order to support scheduling policy extensibility to meet varying application needs and different local scheduling policies at the various sites, grid middleware (e.g, Globus) does not implement its own scheduler but depends on higher level schedulers. Metaschedulers or resource brokers which are currently available provide scheduling solutions for a particular class of applications. They also lack the flexibility of accommodating multiple scheduling policies or providing a subset of the policies implemented. In this thesis we provide a solution for providing frameworks for scheduling policy extensibility, minimum requirement extensibility and choices of job submission mechanisms.

- Encourage software reuse

  We use the term "framework-based" to describe MFATIC because of its reliance on framework libraries for every phase of the scheduling and initiation process. The frameworks are based on well known design patterns which promote software reuse. Currently, scheduling functionality for every application and for different computing environments is replicated. The use of frameworks and libraries ensures that the code can be reused in different environments.

The rest of the thesis is organized as follows. Chapter 2 provides a literature survey of the current implementations of resource management systems for grids. Chapter 3 emphasizes the motivation and approach for MFATIC. The design and implementation of the system is covered in Chapters 4 and 5 and use cases are presented in Chapter 6. Chapter 7 presents the conclusion and related future work.

Chapter 2. Related Work

2.1 Introduction to Grid Resource Brokers

The grid computing environment contains heteregenous resources and local schedulers. Applications have varied requirements which not only include system resources but also encompass other requirements like data locality and economic and deadline constraints. A number of resource management architectures have been proposed and implemented. In this chapter we describe the most influential projects. In the case of local management systems, centralized policies exist since complete state information is available. The local management systems attempt to optimize performance by resorting to optimal scheduling strategies. This method of scheduling cannot be extended to the grid environment since performance metrics cannot easily be defined for a complex environment and since neither complete state information nor scheduling policy control is available.

The typical approach to scheduling across a grid uses decentralized or hierarchical scheduling policies implemented by a grid resource broker. The grid resource broker mediates between application requirements and the resource owner's policies. The broker performs resource discovery, selection, computational environment setup, initiation of execution on the remote resource, monitors progress and gathers results.

2.2 Grid Components and Characteristics

The components of the grid are shown in Figure 2.1.

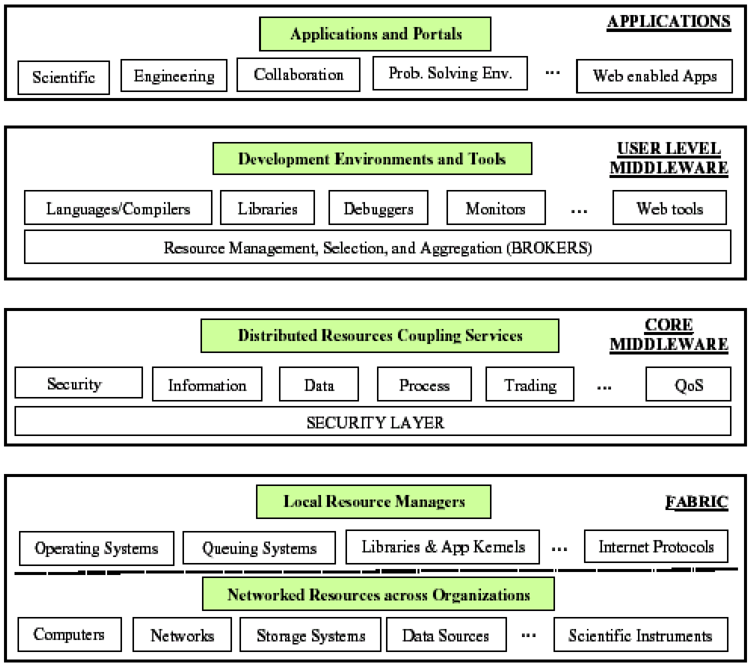

Figure 2.1 : Grid Components [2]

The key components are as follows

- Grid Fabric: This component represents the grid resources which can be accessed through the Internet. This also includes the various resource management systems deployed at the sites.
- Core Grid Middleware: The various services of remote job submission, file transfer, information services, storage access, QoS, advance reservation are part of the core grid middleware.
- User Level Middleware [3]: This includes the tools necessary for the user to build grid enabled software. It includes libraries, programming tools and resource brokers.
- Grid Applications and Services: Grid applications are built using user level middleware, and grid services are deployed in a web server to enable access to grid services from browsers. The two scenarios are applicable to computational and service grids, respectively.

The characteristics of the system influence the design of the system wide scheduling policy. Grids have the following four characteristics

a. Multiple administrative domains: Grid infrastructures span multiple organizations and hence need to take into account different resource management policies.

b. Heterogeneity: The resources which form the grid are heterogeneous and hence scheduling metrics should not be restricted to computational power.

c. Scalable: Grids are highly scalable and can encompass millions of resources geographically dispersed across the world.

d. Dynamic: Dynamicity is the result of such a highly scalable system. The number and variety of resources in the grid is highly volatile.

2.3 Taxonomy of Grid Resource Management Systems

Grids are built for various purposes. Grid resource management systems can be broadly classified on the basis of their application into computational, data and service grids. As the name suggests, computational grids are used for high throughput computing and distributed computing applications. The combination of large datasets, geographically dispersed users and datasets and computationally intensive result analysis, demand optimal performance that are not satisfied by general data management architectures. Hence data grids [4, 5] are used for data management purposes. The most sought after use of grid computing for the industry is service grids. Service grids [6] are a result of the union of grid computing and web services [7] technologies.

According to Buyya [2], resource management systems (RMS) can be classified on the basis of the attributes defined in Table 2.1. The organization of machines in the Grid determines the communication pattern between the resource management systems and hence the scalability of the resultant architecture. The resource model is used to describe and manage the grid resources. The schema based approach describes resources in a descriptive language with certain integrity constraints while the object model describes the resources in terms of generic objects whose definitions are extensible [8]. The QoS allows the user to define the level of service expected and this influences the design of the RMS. Resource information discovery, dissemination and storage mechanisms are characteristic of the RMS and are used as classifying attributes. Just like other distributed systems the scheduling system can be classified as centralized, hierarchical or decentralized.

Attributes	**Taxonomy**
Grid type	Computational grid, data grid, service grid
Machine organization	Flat, cell(flat cells and hierarchical cells), hierarchical
Resource model	Schema, object model
Namespace organization	Relational, hierarchical, graph
QoS	Soft, hard and none
Resource information store	Network directory and distributed objects
Resource discovery	Query and agents
Resource info dissemination	Batch/period, online/on-demand
Scheduler organization	Centralized, hierarchical, decentralized
Scheduling policy	System-centric, user centric
State estimation	Predictive and non-predictive
Rescheduling	Periodic, event driven

Table 2.1 : Attributes and taxonomy for grid resource management systems

[adapted from [2]]

A listing of some of the currently available grid resource management systems and their attributes (Table 2.1) is presented in Table 2.2.

System	Grid Type	Organization	Resource: model, discovery, storage and dissemination mechanisms	Scheduling: organizations, state-estimation and policies
AppLeS	Computational Grid	Hierarchical	Uses resource model by underlying middleware services.	Decentralized scheduler, predictive heuristic state estimation and fixed application oriented policy

DataGrid	Data and Computational	Hierarchical	Extensible schema model, network directory store, distributed query-based discovery, periodic push dissemination.	Hierarchical schedulers, predictive heuristic state estimation and extensible scheduling policy.
Condor	Computational Grid	Flat	Extensible schema model, network directory store, centralized query based discovery, periodic push dissemination	Cooperative/centralized scheduler
Globus	Grid toolkit	Hierarchical cells	Extensible schema model, LDAP network directory store, distributed query based discovery, periodic push dissemination	Hierarchical scheduler, ad-hoc extensible policy
Legion	Computational Grid	Flat Hierarchical	Extensible object model, object model store, distributed query-based discovery, periodic pull dissemination	Hierarchical scheduler, ad-hoc extensible scheduling policies
NetSolve	Computational and Service grid	Hierarchical	Extensible schema model, centralized query based resource discovery, periodic push dissemination	Decentralized scheduler, fixed application oriented policy
Nimrod-G	Computational and Service grid	Hierarchical cells	Uses the model by the underlying middleware and extends it for economic based approach	Decentralized scheduler, predictive pricing models, fixed application oriented scheduling policy
MFATIC	Computational and Service grid	Hierarchical	Uses the model by the underlying middleware.	Decentralized scheduler, extensible and configurable scheduling policy

Table 2.2 : Characteristics of Grid RMS [adapted from [2]]

The above resource management systems and brokers can be organized based upon the layered architecture of grid components (Figure2.1) as shown in Table 2.3.

User Level MiddleWare	AppLeS Nimrod-G Condor-G MFATIC
Core MiddleWare	Globus Legion

Programming environments	NetSolve(Integrated Grid system)
	EU-DataGrid(Applications driven effort)

Table 2.3 : Hierarchical Organization of Grid RMS

AppLeS, Nimrod-G and Condor-G are resource brokers and provide an interface to the user that hides the complexities of the core middleware. The Globus toolkit is a bag of service which provides the basic middleware services used by resource brokers in scheduling. NetSolve is a complete programming environment and runtime system for users of the Grid and provides transparent access to high-performance libraries and resources. The European DataGrid is more application oriented and provides the necessary middleware for a class of applications.

2.4 AppLeS

The AppLeS (Application Level Scheduling) project at the University of California, San Diego focuses on developing scheduling agents for applications on a computational grid. It uses the Network Weather Service (NWS) [9] which dynamically forecasts the usability of system resources. It interacts with resource management systems such as Globus, Legion and NetSolve to implement its tasks. The applications have embedded agents and are thus self-schedulable on the grid. NWS uses predictive heuristic state estimation and this feature is used by AppLeS for scheduling and online rescheduling strategies.

The AppLeS scheduler has application oriented scheduling policies and the framework provides templates for different classes of applications like parameter sweep and master-slave applications. It uses the resource models provided by the underlying resource management system. The mapping of jobs to resources is provided by the scheduler but the actual execution of application units is performed by the local resource schedulers.

Hence the key idea for AppLeS is to schedule applications for a dynamic grid, but it only reacts to changes in system resources. Other factors like data locality and grid economy are ignored.

2.5 Condor

Condor [10] is a high throughput computing environment developed at the University of Wisconsin at Madison, USA. It can manage a collection of computers owned by different individuals and yet be able to satisfy both the user demands and the resource constraints placed by the resource owner. Condor achieves this by the matchmaking process. Both resource constraints and user requirements are represented as ClassAds [11]. The Condor system has five basic mechanisms for its operation

- Detecting an idle machine: CPU load and time since last keyboard or mouse activity are two conditions checked to determine if a workstation is idle.
- Fair allocation mechanisms: Resources are allocated to users based upon their priority and this priority is assigned in accordance to the up-down algorithm. This assures that heavy users get a large amount of work done and still prevents infrequent users from starvation.
- Provides a remote execution mechanism: File I/O is redirected from the execution machine to the submitting machine to eliminate the post processing overhead of staging in data from the remote machine.
- Mechanism to return the workstation to the owner on the first user activity: The use of preemptive processes to run Condor jobs helps in preempting the resource held by the executing process and returning it back to the owner.
- Checkpointing and migration mechanism: Condor provides a transparent checkpointing mechanism which allows it to checkpoint a running job and migrate it to another machine when the machine it is running on becomes busy with other non-Condor activity.

A Condor resource agent runs on each machine periodically advertising the resource constraints to the central manager. The request agents advertise user requests for resources to the central manager. The Condor matchmaker then looks for compatible agents and contacts each of them indirectly to check if both the parties are satisfied. Once this step is done the client is allowed to instantiate computation on the target resource.

Condor is a computational grid with a flat organization. It uses an extensible schema with a hybrid namespace. Resource discovery is achieved by centralized queries with periodic push dissemination. The scheduling is centralized.

Condor-G [12] is a resource broker for computational grid environments. Used as a front-end to a computational grid, Condor-G can manage job execution across distributed sites across multiple administrative domains. It provides job monitoring, logging, notification, policy enforcement, fault tolerance, credential management, and it can handle complex job-interdependencies.

There are two major implementations of resource brokers which use the classad matchmaking mechanism.

a) EU DataGrid(WP1 Workload Management System[WMS]) [13]: This architecture has a generalized approach towards workload management and could be used for any kind of application. The resource broker is one of the important components of the management architecture. It finds a resource which best matches the requirements and preferences of a submitted job. The factors include :

1. Location of input and output data
2. Authorization to use a resource.
3. Allocation of resources to the user or to the groups the user belongs to.
4. Requirements and preferences of the user.
5. Status of the available resources.

b) D0 SAMGrid [14]: The D0 project approaches the problem from two directions: logical job management and physical job management. Logical job management involves understanding a D0 job and how it is viewed by the user and the scheduler. Physical job management involves resource brokering, planning and scheduling. D0 jobs are fundamentally data intensive and so the decision of matching a job to a resource depends heavily on the data management system.
Factors affecting scheduling :

1. The utilization, current and projected, of the disk caches managed by SAM stations [14]. The rate of data consumption by the existing tasks.
2. The throughput of the data transfers to/from stations.
3. The near term predictions, from the Data Handling Global Resource Manager[14], on the availability of the remotely cached datasets.

In addition there is a mechanism by which an external ranking function can be provided.

2.6 Data Grid

The international DataGrid Project [13, 15] was established by CERN, the European Organization for Nuclear Research, and the high Energy Physics (HEP) community with the intent to apply the work to other scientific communities like the Earth Observation and Bioinformatics. This project focuses on developing middleware services in order to enable distributed analysis of physics data. The core middleware system is Globus with extensions for supporting data grids. Data in the order of petabytes will be distributed in a hierarchical fashion amongst multiple sites worldwide. Global namespaces for data items, special workload distribution facilities for load balancing across multiple sites and user and application monitoring are some of the special features of the DataGrid middleware.

The DataGrid Project is organized into 12 Work Packages. The first five Work Packages are developing the “middleware” software, based on existing grid toolkits. WorkPackage 1 deals with workload scheduling and aims at defining and implementing an architecture for distributed scheduling and resource management. Work Package 2 deals with data management and aims at implementing and comparing different distributed data management approaches including caching, file replication and file migration. Work Package 3 is specifying, developing, integrating and testing tools and infrastructures to enable access to status and error information in a grid environment. The objective of Work Package 4 is to develop new automated system management techniques that will enable the deployment of very large computing fabrics constructed from mass market components with reduced system administration and operation costs. Work Package 5 has two objectives: defining a common user API and data export/import interfaces to various local mass storage management systems used by project partners and publishing details of available storage systems via grid information systems. Work Package 6 (Testbed) and Work Package 7 (Network) deal with planning, organizing and operating the testbeds used to demonstrate and test the data and computing intensive grid in production quality operation over high performance networks. The objective of Work Package 8 is to demonstrate the feasibility of the DataGrid technology to implement and operate effectively an integrated computing and data access service in an internationally distributed environment.

The DataGrid has a hierarchical machine organization with less data stored at lower levels of the hierarchy. It has an extensible schema based resource model with a hierarchical namespace organization. Resource dissemination is batched and periodically pushed to other parts of the grid.

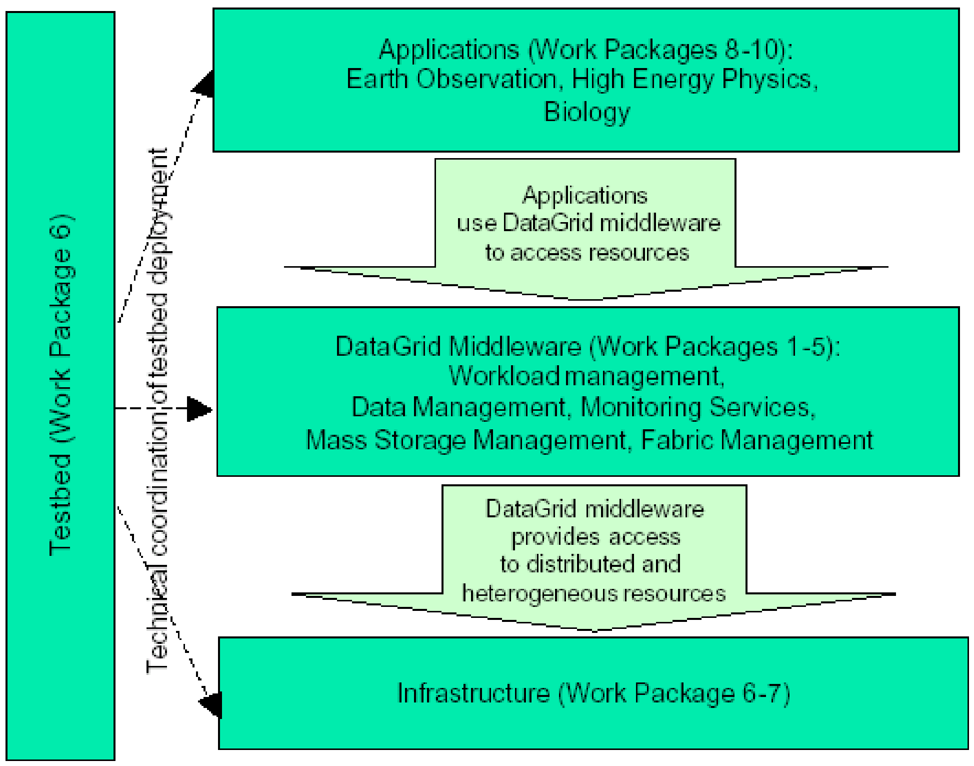
Testbed (Work Package 6)
Technical coordination of testbed deployment
Applications (Work Packages 8-10): Earth Observation, High Energy Physics, Biology
Applications use DataGrid middleware to access resources
DataGrid Middleware (Work Packages 1-5): Workload management, Data Management, Monitoring Services, Mass Storage Management, Fabric Management
DataGrid middleware provides access to distributed and heterogeneous resources
Infrastructure (Work Package 6-7)

Figure 2.2 : DataGrid Middleware organization [15]

2.7 Globus

Globus [16] provides a software infrastructure that enables applications to view distributed heterogeneous computing resources under different administrative domains as a single virtual machine. The toolkit is oriented towards providing middleware services for building computational grids. It implements four main services which are security, resource management, information services and data management. Data management functionality is minimal and contains only basic services such as gridftp to transfer data across the grid. The toolkit provides a bag of services which developers of specific tools or applications can select to meet their needs. Globus is constructed as a layered architecture in which higher level services can be developed using the lower level core services.

Information about grid resources is maintained by an LDAP-based network directory called the Metacomputing Directory Service (MDS) [17, 18]. The MDS has a hierarchical structure that consists of three main components. The Grid Index Information Service (GIIS) provides an aggregate directory of lower-level data. The Grid Resource Information Service (GRIS) runs on a resource and acts as a modular content gateway for a resource. Information Providers (IPs) interface from any data collection service and then talk to the GRIS. A GRIS registers with a GIIS, and one GIIS may register with another using a soft-state protocol. Caching at each level reduces the network traffic related to periodic updates. The MDS follows both push and pull protocols for information dissemination.

Higher level tools such as the resource brokers and user level middleware schedulers can query the LDAP service to obtain resource specific information. MDS is capable of providing both static and dynamic information regarding resources and their usage.

2.8 NetSolve

NetSolve [19] is a client-server system that enables users to solve complex scientific problems remotely. The system allows users to access both hardware and software computational resources distributed across a network. The intent of Netsolve is to hide parallel processing complexity from the users and yet provide the necessary parallel processing power to their applications. Clients can be written in C, Fortran, Matlab or using web pages to interact with the server.

NetSolve provides resource discovery, fault tolerance and load balancing. The agent represents the gateway to the NetSolve system. It maintains a database of NetSolve servers along with their capabilities and dynamic usage statistics for use in scheduling decisions. The NetSolve agent attempts to find the server that will service the request, balance the load amongst its servers, and keep track of failed servers. The agents use a push resource dissemination model. The agent also acts as a resource broker and performs resource discovery and scheduling. Agents work co-operatively on the scheduling problem.

This system follows the service grid model with hierarchical cell based machine organization. Netsolve has a decentralized scheduler organization.

2.9 Nimrod/G

Nimrod/G [20] supports soft deadline and budget constraining scheduling of task farming applications on a P2P grid distributed across the globe. This approach provides

economic incentive for resource owners to share their resources on the grid and encourages the emergence of a new service-oriented computing industry. More importantly, it provides mechanisms to trade-off quality of service (QoS) parameters, deadlines, and computational costs and offers incentive for relaxing requirements. Depending on users' QoS requirements, Nimrod/G dynamically leases grid services at runtime depending on their cost, quality, and availability. In addition to deadline and budget constraints in scheduling, it supports optimization of both or either.
The scheduling model for computational economy based constraints is as follows

1. non-uniform cost model which differs from user to user and is based on the duration for which that particular user utilizes the computational resource.
2. encourages use of local resources first
3. the user has the option of using a remote resource but at a higher cost

Nimrod-G follows the hierarchical and computational market model in resource management. It uses the services of grid middleware systems such as Globus and Legion for resource discovery. It supports resource reservation and QoS through the computational economy services of the GRACE infrastructure.

Chapter 3. Motivation and Approach

3.1 Need for User Level Middleware

In the beginning application programmers for the grid built applications directly based on the grid fabric layer. Middleware was rare and early grid applications were huge and cumbersome to maintain. Programmers, in their experience with developing such applications, began to develop a basic set of services and data that is common for any distributed application. These services could act as glue between the application level and the network level. Core middleware (see Figure 2.1) performs this exact function of providing a core set of services to the application layer and thus reduces the complexity of developing grid applications. Grid middleware provides infrastructure and services for better usability of the grid. It also provides a single system view of distributed grid resources. The emergence of core middleware was a step forward, but overall application performance can still suffer if application developers are forced to handle user level middleware functions such as resource management, selection and brokering.

User-level middleware promised to improve the efficiency of applications. It was developed to hide more of the complexity of the grid and provide hooks into core middleware for improving performance. Better application performance, portability and retargetability are some of the advantages of using user level middleware software. User level middleware software provides tools to the user of a programming environment to build grid-enabled applications. Resource broker services and their corresponding libraries belong to the category of user level middleware.

3.2 Traditional Grid Resource Brokers and Distributed Resource Management Systems (DRMS)

Current grid resource brokers fit into the category of user-level middleware. These brokers simplify the problem of scheduling jobs over grid resources. Most brokers have been developed considering a limited set of scheduling policies and most of them were developed for a particular class of applications. AppLeS performs scheduling of a job based upon the class of applications it belongs to or to the application profiling used by the embedded scheduling agents. AppLeS uses both static and dynamic information of both the application as well as the resources for deciding the schedule. Nimrod-G deals with scheduling in a completely different manner, where the schedule is dependent upon budget and deadline constraints.

The Globus team has identified five challenges [21] to the scheduling problem in metacomputing environments. These are listed below.

1. The site autonomy problem refers to the fact that resources are typically owned and operated by different organizations, in different administrative domains. A direct result of this fact is diversity in scheduling policies and security policies

2. The heterogeneous substrate problem refers to the fact that different sites may use different local resource management systems, such as Condor, NQE, CODINE, EASY, LSF, PBS and LoadLeveler. The same system at two different sites may have different configurations leading to significant changes in functionality.

3. Metacomputing applications arise from different domains with completely different resource requirements and hence scheduling policy extensibility is one of the problems for a grid resource management system. A resource management solution must support the frequent development of new domain-specific management structures, without requiring changes to code installed at participating sites. The resource brokers make scheduling decisions which are application oriented and the resource management system needs to support these decisions.

4. Many applications have resource requirements that can be satisfied only by using resources simultaneously at several sites. Co-allocation mechanisms solve this particular need for allocating multiple resources, initiating computation on those resources, and monitoring and managing those computations.

5. The online control problem arises because substantial negotiation can be required to adapt application requirements to resource availability, particularly when requirements and resource characteristics change during execution.

The resource management system architecture for Globus deals with each of these five problems. It addresses problems of site autonomy and heterogeneous substrate by introducing entities called resource managers to provide a well-defined interface to different local resource management tools, policies, and security mechanisms. For online control an extensible resource specification language is defined that supports negotiation between different components of a resource management architecture. Resource brokers handle the mapping of high-level application requests into requests to individual managers and hence the problem of policy extensibility is solved or pushed to a higher level of abstraction. Resource co-allocators with co-allocation strategies help solve the co-allocation problem.

The problem of scheduling policy extensibility is not addressed by the Globus core middleware level. Instead responsibility for resource selection is pushed to the level of brokers which are part of the user level middleware. The resource brokers at the user level middleware are written for particular applications and hence implement a fixed set of scheduling policies for a particular class of applications. However, imagine a case where an application needs to be scheduled based on deadline constraints and a basic set of system resource requirements. Suppose the software design team feels that the most efficient schedule could be developed by a joint effort of the Nimrod-G broker and the AppLeS agents. This is not achievable with existing resource brokers because such a co-operative resource selection effort has not been taken into account during the design of these systems. Furthermore, a subset of the resource brokers functionality cannot be used. A final problem with resource brokers is that they are tightly coupled to their own job submission mechanisms and the user has no choice. Our solution to these problems is to decouple each phase or aspect of superscheduling. The term “superscheduling” refers to the process of making scheduling decisions and starting up jobs across resources in multiple administrative domains. The goal is a pluggable environment which allows the user to have a choice of both scheduling policies and job submission mechanisms.

3.3 Solution and Approach

Applications have a choice among several scheduling policies. Application developers should be able to make a decision as to which set of policies are applicable to their application. In a way, this is similar to the pluggable authentication modules mechanism (PAM) which is a flexible mechanism for authenticating users. The idea for PAM is to help users develop programs which are independent of the authentication scheme.

The solution we propose in this thesis provides similar flexibility of choice of scheduling policy, but the decision is made by the application developer. It may be argued that scheduling decisions need to be made by the middleware and that the developer should be unaware of the fact that the application is being scheduled over a grid. In the solution presented here, the developer needs to specify the policies and not the implementation; this achieves some level of abstraction but does not completely hide the scheduling process. This may go against the principles of abstracting lower level processes, but it provides flexibility needed for grid systems.

Our solution relies on a Matchmaker framework and a Resource Allocation framework. The scheduling issues for a metacomputing environment include more issues than just application performance. Issues like minimum system requirements, data locality, fine [22] and coarse grained access to resources cannot be ignored. These miscellaneous issues pertaining to running a job on the grid can be termed “minimal requirements” for the application. Hence the scheduling process can be divided into a two phase process of requirement fulfillment and scheduling decisions. The first phase, requirement fulfillment, results in a subset of resources which can run the job but will not necessarily ensure optimal performance. The scheduling decision phase decides the final resource or set of resources for job execution. This decision is based on both application requirements and system centric policies established by the resource owner.

Our Matchmaker framework provides an extensible framework for specifying the minimum requirements and the Resource Allocator provides an extensible framework for scheduling policies.

The Matchmaker framework is based upon the Factory design pattern [23]. Each application requirement is implemented as a subclass of a parent abstract class and hence exposes a common interface. Each of these subclasses is instantiated as a factory object at runtime. The Matchmaking framework performs resource selection but the other steps to scheduling will be performed by other components of the architecture. From the implementation point of view, by “components” we do not mean software components but libraries which are used by the application. Each library is a framework which offers a choice of different implementations. The MFATIC architecture provides the application developer a choice of a distributed resource management system (DRMS) job submission mechanism and scheduling policies. By providing a framework as a library just a few method calls need to be made and the rest is handled by the framework, thus reducing the code complexity.

Our approach fits the view of a global service for different applications in the case of a global grid. In this scenario different developers with different application requirements will need to schedule their tasks on the grid. A global resource selection service with a configurable set of policies for different users is necessary. Every user can configure the set of services he needs from each of these components and plug them together for a particular application.

Chapter 4. Design

4.1 Decoupled Job Scheduling and Initiation System

Job scheduling and initiation service architectures in use today have a rigid format with respect to the steps taken for a job to be executed. The resource broker is an essential component of any job scheduling and initiation architecture. In practice, resource brokers are responsible for a wide range of functions, including resource selection, resource allocation, job startup, monitoring, and steering. In current practice, users of a particular job scheduling and initiation service cannot replace or reconfigure the functionality of the resource broker. For example, a user who requires special features from Nimrod-G and AppLeS cannot combine the strengths of these two approaches. Or, a user may need just one function from the resource broker (e.g., the matchmaking service to find the right resource for the job) but that user may want to use a local scheduler for scheduling the job instead of the resource broker's scheduling service. Current resource brokers select the resource and continue to schedule the job on it without considering user preferences. The primary goal of this research is to design a job scheduling and initiation architecture which is much more open and reconfigurable than current approaches. Users should be able to pick and choose components to fit into the architecture. The results of every step of the resource selection process should be accessible so that job scheduling and initiation services with a variety of characteristics can easily be implemented.

Figure 4.1 gives a high-level view of the overall system architecture for MFATIC. We first give a brief overview of the components of this architecture; more details are given in the remainder of this chapter. The user interacts with the system through the user interface. The Matchmaker returns a set of resources that satisfy the minimum requirements defined by the user. These requirements include job requirements (e.g., CPU speed, memory, budget constraint, etc.) and scheduling policies. Choosing this set of minimum application requirements can be thought of as selecting a set of "filtering" criteria. The set of resources is filtered down from the set of all available resources on a grid to a much smaller list of candidate resources based on the requirements defined by the user. The job requirements along with the filtering criteria are passed to the Matchmaker (step 1 in Figure 4.1) and the subset of matched resources is returned (2). Next, the set of matched resources is given to the Resource Allocator (3). It chooses the most suitable resource, maintains a dialogue of negotiation, determines a service level agreement (SLA) between the client and the resource owner, and returns this SLA to the client (4). This SLA along with a runtime specification of the DRMS is used for job initiation. Depending on the choice of the DRMS, the information from the SLA and the application requirements are provided to the corresponding job submission mechanism with the help of Distributed Resource Management Application API (DRMAA) bindings.

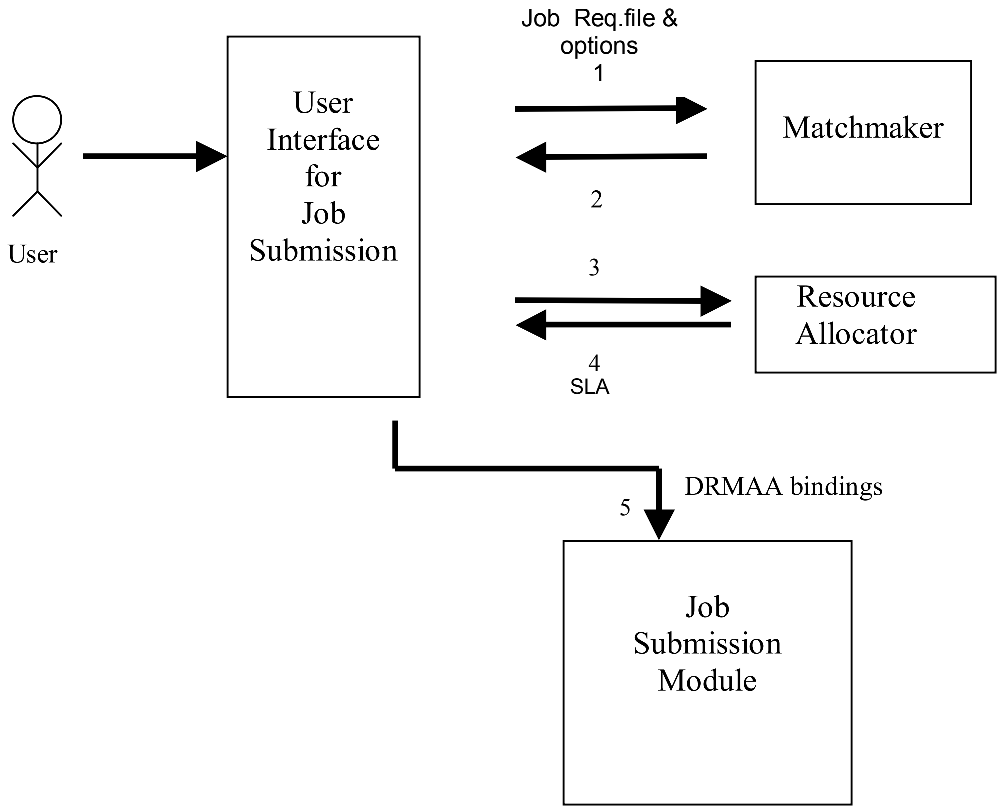

Figure 4.1: Decoupled Job Scheduling and Initiation Service Architecture

Figure 4.2 presents the relationship between MFATIC and the grid environment. MFATIC runs on the client machine. Hence MFATIC is not a centralized scheduling mechanism. Every component of the system is a library and each of these services are spawned when the related library calls are executed.

The Matchmaker (MM), Resource Allocator (RA) and Job Submission Module execute on the client machine. The Matchmaker interacts with grid services like the Metacomputing Directory Service (MDS) to discover resources. The Resource Allocator uses the Grid Resource Allocation Agreement Protocol (GRAAP) to negotiate service level agreements (SLA) with the local scheduler on the selected resource. Based on the SLA the job submission module generates the required DRMAA bindings for the chosen job submission mechanism and submits the task to the selected resource. As seen in Figure 4.2 the final job submission mechanism could be to any local resource management system supported by DRMAA.

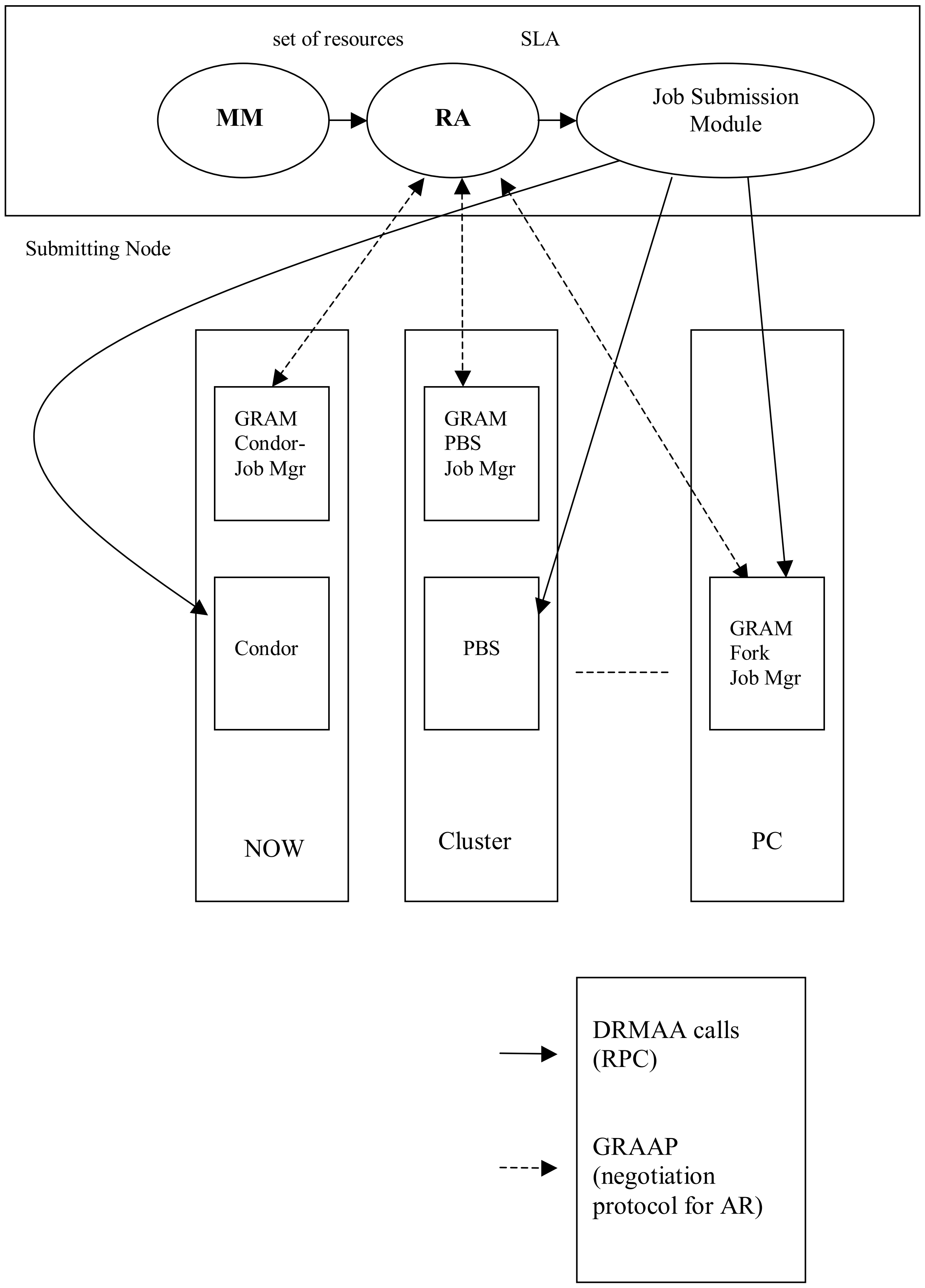

Figure 4.2 : Relationship between MFATIC and the grid environment

The remainder of this section is organized as follows. We first set the MFATIC architecture in the context of emerging community practices for scheduling across the grid ("superscheduling"). Then in Sections 4.2 through 4.5 we give further details regarding each of the modules shown in Figure 4.1.

Relation to established steps in superscheduling

According to the superscheduling working draft [24] produced by the GGF scheduling working group (Sched-WG) there are ten steps to superscheduling. The scheduling process includes three phases. Phase one is resource discovery, in which the user makes a list of potential resources to use. Phase two involves gathering information about those resources and choosing a best set to use. In phase three the user then runs the job. In the remainder of this section we relate the phases of the scheduling process with the functionalities of each of the modules in the MFATIC architecture.

Phase 1: Resource Discovery
The steps to be performed as part of this phase are authorization filtering, application requirement definition and minimal requirement filtering. This phase is a preparation phase for phase 2 which looks deeper into the information provided by phase 1 and then performs the resource selection. Authorization filtering involves deriving a set of resources from the available grid resource infrastructure depending on whether the user has access to the resource or not. Minimal requirement filtering is done on the basis of the minimum requirements defined by the user.

Phase 2: System Selection
The steps involved in this phase and the next may be iterated over a number of times if the scheduler supports rescheduling of jobs. The first step in this phase is to filter the resources based on dynamic information such as current load on the system. The dynamic factors depend on the application requirements and could include data locality and network traffic as important scheduling criteria. A single resource is then selected from the resultant set in the next step.

Phase3: Running the job
Before the actual submission of the job to the selected resource some form of advance reservation is possible. This is an optional step since it depends on the local resource management system deployed at the site. The feature of advance reservation [25] is characteristic of a scheduler and not all local schedulers support it. The advance reservation feature helps reserve a part or all of the resources selected by the previous step. Some of the jobs may need a prior setup or staging phase before the job can be submitted. Depending on the execution time required by the application, users may need to monitor the progress and be notified when the job is completed. Monitoring sensors need to be activated in this case which monitor the progress of the job. Once the job is completed post processing tasks need to be performed which may involve moving files across the network to the client machine for data analysis.

The idea of decoupling [26] is to define tasks for the various pluggable modules in the system and still provide maximum flexibility to the user to replace any module by another. In the MFATIC architecture, the user dependent step of application requirements generation is provided by the user interface. The minimum filtering and authorization filtering is performed by the Matchmaker framework. In this thesis all the minimum requirements for running a job - which involves authorization, minimum system requirements, data requirements, etc. - are collectively termed “minimum requirements” and are handled by the Matchmaker. The Resource Allocator is responsible for the steps in phase 3 of the superscheduling process. The Resource Allocator provides features of advance reservation if the local scheduler supports it. Job submission mechanisms provide job startup and notification. The clean separation of concerns into different modules is the attractive feature of the MFATIC job scheduling and initiation architecture.

4.2 User Interface to the Job Scheduling and Initiation Service

The interface to the system varies for different applications. This service can be initialized and used either by an automated script or by a human. For this to be possible the backend service should expose an interface which can be used by an interface developer. The user interface falls in the category of application software which makes use of the user level middleware. The other components of the architecture are user level middleware.

For a user friendly interface a Graphical User Interface (GUI) can be developed which accepts the required inputs for calling the services of the other components. Each of the modules in the job scheduling and initiation service is independent of each other. The user interface uses the API exposed by each of these services and ties them all up to work as one unit. The interface makes calls to the Matchmaker to return a set of matched resources for the job based on criteria specified by the user. This set of resources can be fed as an input parameter to the Resource Allocator, which in turn performs the function of reserving the most suitable resource and defining a schedule for the job. The service level agreement (SLA) is returned to the calling program. Using DRMAA bindings the necessary parameters for job submission are set up and the job is initiated on the selected node.

4.3 Resource Allocator

The Resource Allocator selects the most suitable resource from the selected set of resources. The selection is based upon the selected set of scheduling policies for the application. This functionality can be overridden by the users choice. In other words the user has a choice of using the Resource Allocator functionality. The final decision is made in accordance with negotiations with the local scheduler for the site. The Resource Allocator is used to resolve race conditions when multiple Resource Allocator modules decide on using the same resource. Not all grid users will use the Resource Allocator since they may prefer to use another scheduling mechanism for their computational jobs. The Resource Allocator considers these situations to generate a deterministic schedule.

The distinguishing factor between the Matchmaker and the Resource Allocator is functionality. The Matchmaker can only provide a potential set of resources; it does not guarantee that the minimum requirements of the job will be available when the job is scheduled to run. This guarantee is provided by the Resource Allocator.

The Resource Allocator is designed as a framework. This provides flexibility to the user in choosing and combining appropriate scheduling policies. The scheduling policies select a resource or a set of resources depending upon the number of resources requested. The Resource Alllocator provides a mechanism to add new scheduling policies and to combine the results of two or more scheduling policies.

Before the Resource Allocator goes ahead with the submission it makes an advance reservation with the DRMS to ensure that the resource is available for the job and that the minimum system requirements for the job are satisfied. Each of these resource guarantees are incorporated in the service level agreement (SLA). Advance reservation capabilities are characteristic of the DRMS and hence a local scheduler which supports advance reservation through SLAs should be used. Future versions of the GRAM protocol in Globus intends to provide grid scheduling through service level agreements [27].

The resource reservation grants limited or restricted delegation of resource capabilities for a defined time interval obtained from a negotiation process. Based on the needs defined by the job, the Matchmaker looks out for resources which fulfill these requirements and returns a set of resources. The function of the Resource Allocator is to reserve these resource capabilities for a defined time interval.

The communication protocol for such reservations at present is dependent on the DRMS and each of them has different access modes. However, the GGF Scheduling Group (Sched-WG) has defined the requirements for a standard Grid Resource Management Protocol [28] which will provide for interoperability between the clients and the DRMS independent of the type of resource or the native resource access modes. These requirements are being extended for advance reservation systems. The Grid Resource Allocation Agreement Protocol (GRAAP)[29] has been devised for this purpose by the GRAAP working group[30]. The Resource Allocator should communicate with the local resource managers (LRMS) using GRAAP. The GRAAP protocol has been devised to reconcile the competing demands of the resource owner and the client by negotiating service-level agreements (SLA). The SLA is a contract between the client and the resource owner containing measurable capabilities which the resource owner promises to the client.

In an advance reservation the requests are permitted access only at a determinate future time. The commitment protocol in the case of advance reservation is different from other systems which generally follow a two phase commit. An advance reservation system follows delayed semantics and the policies change for different schedulers. In some cases a message needs to be sent at the time of job startup to confirm the reservation or the request is executed automatically at the determined job startup time. In the case of the Resource Allocator it should negotiate this commitment procedure with the manager during the advance reservation negotiation process. The agreed upon commitment protocol is then used for the start of the job submission request at the determined time. The Resource Allocator returns the SLA and other agreed upon protocols.

4.4 Job Submission Module

The job submission phase is tightly coupled to the DRMS. The specification of job submission parameters and the resource name needs to be defined for submitting the request to the DRMS. The resource name and the configuration parameters are available from the SLA. This binding of configuration parameters to the request and the submission of the job is defined by APIs provided by different vendors for their DRMS. The GGF Distributed Resource Management Application API working group (DRMAA-WG) is defining a generalized API to a DRMS. This API should be used by the job submission framework to submit requests. The choice of DRMS can be delayed until runtime by use of the Distributed Resource Management Application API (DRMAA) [31]. The DRMAA is a work in progress but aims at providing an uniform interface to various DRMS.

The job submission system should be able to provide asynchronous status requests and notification. To query for job status information asynchronously, persistent job IDs are important. The DRMS needs to be able to support persistent job IDs. On dispatching the request the job submission mechanism needs to retrieve and archive the job ID for future usage. These IDs will not only help in obtaining job status but will also help in case of dynamic binding of configuration parameters to running jobs. Users may be spread across a domain or across different domains and yet would need access to this data. In such a scenario the job submission module should be able to convey job ID information to a restricted set of users. This set is defined at submission time and an Access Control List (ACL) is generated for the corresponding job ID. This ensures reliable access to job related information. The job monitoring and control capabilities are outside the scope of this paper. Monitoring can be performed by frameworks like the Distributed Monitoring Framework (DMF) [32].

4.5 Matchmaker Framework Architecture

The motivation for modular frameworks is stated nicely in a technical report from CC Pace Systems [33]:

"*One of the key problems in the development of modern software systems is planning for change: open systems must be flexible in that they must be easy to adapt to new and changing requirements. Increasingly systems developers have come to the consensus that the best way of dealing with open requirements is to build systems out of reusable components conforming to a "plug-in architecture". The functionality of an open system* *can then be changed or extended by substituting components or plugging in new components.*

A software system that may be reused for creating complete applications is called a framework. The idea is that it should be relatively easy to produce a range of specific systems within a certain domain starting from the framework software."

As mentioned above, systems need to be able to adapt to new and changing requirements. Current matchmaking systems have a rigid criterion for matchmaking. However, the needs of users change for different applications. Flexibility to choose matchmaking criteria is an important factor. Hence the MFATIC Matchmaker has been implemented as a framework. In this case the framework has a special purpose of creating an extensible system for specifying multiple criteria for matchmaking. The matchmaking process is then based upon the choice and ranking of criteria specified by the user. The set of resources go through this series of filtering criterion which filters out inappropriate resources and returns a subset of the resources which match the job's minimum requirements.

Frameworks are designed on the basis of well established design patterns. Two immediate problems are anticipated while designing such a system.

a. Determining at run time which matchmaking option to activate for the matchmaking process.
b. Ability to load any class implementing the matchmaking option at run time. In other words the program should not contain any hard coded class names which implement the matchmaking option to be loaded at runtime.

The Factory Pattern is a creational pattern. This pattern helps to model an interface for creating an object which at creation time can let its subclasses decide which class to instantiate. We call this a Factory Pattern since it is responsible for "Manufacturing" an Object. The Factory Pattern promotes loose coupling by eliminating the need to bind application-specific classes into the code. The Factory Pattern is used when

- A class cannot anticipate which kind of class of objects it must create.
- A class uses its subclasses to specify which objects it creates.
- The knowledge of which class gets created should be localized

The Matchmaker framework exposes an interface to the interface developer to use its services. The controller class is called the Initiator. This class is responsible for initiating the matchmaking process and returning the final set of matched resources to the calling program. The calling program is the user interface to the job submission service. The initiator class executes the required criteria by instantiating the required subclasses and these subclasses are returned by the ConcreteFactory. The UML for the design of the framework is depicted in the Figure 4.3.

The controller (Initiator) class does not make the decisions of which class needs to be instantiated. This decision is made by the ConcreteFactory class. In this case the FactoryMethod within the ConcreteFactory class decides which class to instantiate. It uses dynamic class loading to load the appropriate ConcreteModule instance.

ConcreteModule is an implementation of a matchmaking criterion. The input to the ConcreteFactory class is a criteria the user has specified for matchmaking. Based on this input the factory makes a decision about the instantiation of the right ConcreteModule class. All of these matchmaking options are implemented as subclasses of the AbstractModule class which defines an interface for the subclasses to implement. The ConcreteModule returned is typecast to the AbstractModule class type and is used by the Initiator class.

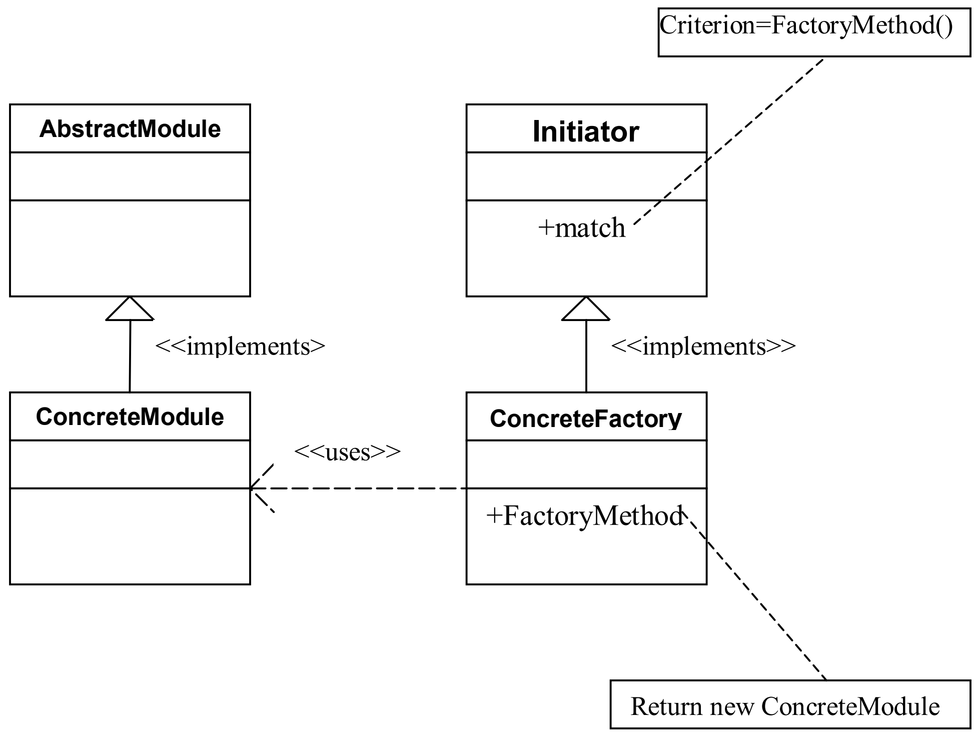

Figure 4.3 : UML for Factory design pattern for the Matchmaker framework

The number of subclasses of the AbstractModule present in the framework keeps changing and hence the decision making process of the ConcreteFactory should be able to accommodate these changes in its process. The Java dynamic class loading mechanism[34] helps delay class loading until the very last minute thus reducing memory overhead of loading these classes and also improving system response time. Dynamic class loading allows the ConcreteFactory class to find classes at run time which are subclasses of the AbstractModule class.

Chapter 5. Implementation

MFATIC is still in its developing stage and the matchmaking framework is the only component that has been implemented. The other components depend on other services which at the time of this implementation did not exist. The Matchmaker framework is implemented as a library so that the user can use the API exposed to extract the required matchmaking functionality.

5.1 Grid Infrastructure Setup

The Globus toolkit is used as the grid middleware and the matchmaking library makes use of the grid middleware for matchmaking; but the final job submission can be done using any job submission mechanism. The testbed includes machines in the Virginia Bioinformatics Institute (VBI). Globus is deployed across all 40 Linux boxes at VBI. This deployment is done automatically using the SystemImager software. It is used to automate installs, updates and software distribution across Linux machines. Once the image is distributed to all machines a script for Globus installation is executed. This script mounts the Globus installation directory from the golden client. The common binary directories are copied and the /etc directory files are modified for the particular host. The host certificates and keys are copied in the required locations. The gatekeeper daemon, the gridftp server and the GRIS is started on each of the machines. The peculiarity of the grid infrastructure setup at VBI is that the Globus services running are for the Linux platform. Most of the workstations are dual boot systems with both Windows and Linux operating systems and hence the grid is extremely dynamic depending on whether the individual has decided to boot his system in Linux or in Windows. This dynamism in terms of the number of machines on the grid is accompanied by other dynamics like changes in system memory usage and CPU utilization. Some of the machines on this grid are Symmetric Multiprocessor systems (SMPs) thus providing a variety in the type of resources available for computing. The dynamic nature of the grid is well represented by the VBI grid and hence acts as a good testbed for the matchmaking facility.

A provider to push authentication information for each of the resources to the LDAP server has been developed. The installation is configured as a hierarchical grid Information and Indexing Service (GIIS) [35]. Each of the resources in the grid infrastructure reports to a centralized GIIS. This particular setup helps reduce a lot of network overhead of contacting each of the grid resources for information but it also means it has a central point of failure. The Distinguished Name (DN) is changed for this collection of Lightweight Directory Interchange Format (LDIF) records by setting Mds-Vo-name attribute in the DN to a different name for the site and thus providing the aggregate directory a virtual organization name.

The authentication provider is a perl script which parses the entries in the grid-mapfile for the local machine and prints out the subject DN and its mapping to the local account. This information is pushed to the GIIS as LDIF records. An example of an LDIF record obtained by querying the centralized server about machine txue is shown in Figure 5.1.

```
dn: Mds-user-info=gridId, Mds-Host-hn=txue,Mds-Vo-name=vbicentral,o=Grid
objectClass: MdsJKExtension
Mds-authentic-user-DN: /O=Grid/O=Globus/OU=cs.vt.edu/CN=Jeevak Kasarkod
Mds-authentic-user-DN: /Email=jkasarko@vt.edu/CN=Jeevak Kasarkod/OU=Virginia T
 ech User/OU=Class 2/O=vt/C=US
Mds-authentic-user-DN: /Email=kgarach@vt.edu/CN=Kunal Garach (kgarach)/OU=Virg
 inia Tech User/OU=Class 1/O=vt/C=US
Mds-authentic-user-DN: /Email=adandapa@vt.edu/CN=Anusha Dandapani (adandapa)/O
 U=Virginia Tech User/OU=Class 1/O=vt/C=US
Mds-authentic-user-DN: Anusha
Mds-authentic-user-account: jkasarko
Mds-authentic-user-account: jkasarko
Mds-authentic-user-account: kgarach
Mds-authentic-user-account: adandapa
Mds-validfrom: 20030430205759Z
Mds-validto: 20030501085759Z
Mds-keepto: 20030501085759Z
```

Figure 5.1 : Sample LDIF record containing the authenticated user- DNs and the accounts

5.2 Data Representation: Simulation Definition Language (SDL)

The eXtensible Markup Language (XML) is a de-facto standard for sharing data and information. It is license-free, platform-independent and is well supported. The data representation for specifying the job and its requirements is an XML based language called the simulation definition language (SDL) [36]. This core set of tags and attributes can be extended for additional functionalities. A sample SDL file is shown in Figure 5.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation>
     <no_procs>1</no_procs>
    <working_directory>/home/jkasarko</working_directory>
    <executable>/home/condor/data/jkasarko/remote.bin/BLAST/blastall</executable>
    <stdout>OUT</stdout>
    <input_files nfiles="2">
        <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/vector</file>
        <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/sample.fasta</file>
    </input_files>
    <cpufree average="1min" equality="greaterthan">10</cpufree>
    <fsfree equality="greaterthan">10000</fsfree>
    <ramfree equality="greaterthan">10</ramfree>
    <vmfree equality="greaterthan">40</vmfree>
    <osname>Linux</osname>
    <rank>ramfree</rank>
</simulation>
```

Figure 5.2 : Sample SDL file

The SDL schema allows a user to define the requirements of the job which includes the input and output files for the job to be run at the destination resource. It also includes the system requirements as far as CPU utilization and amount of free memory is concerned. The semantics of the XML tags are given in Tables 5.1 and 5.2.

XML Tag	Semantics of the Tag
no_procs	Number of Processors
working_directory	Working directory for the job.
Executable	Executable for the job
Stdin	Standard input for the job
Stdout	Standard output for the job
input_files	Input files for the executable
output_files	Output files for the executable
Cpufree	Average cpu utilization for a period of 1,5 or 15 minutes
fsfree(in bytes)	Amount of free filesystem space
ramfree(in bytes)	Amount of free memory
vmfree(in bytes)	Amount of free virtual memory
Osname	Name of operating system
Osrelease	Release version of the operating system
Rank	System ranking attribute

Table 5.1 : SDL tags

XML Attribute	Semantics of the attribute
nfiles	Number of nested tags for a multivalued tag
equality	Relational operator to be used in conjunction with element.
average	Specific to cpu utilization. It can hold only 3 possible values of 1, 5 or 15 minute averages.

Table 5.2 : SDL attributes

The elements defined in Table 5.1 are used by the Matchmaker for filtering based on system requirements. This basic set of elements can be extended to support more requirements as needed by the user. However the parsing and handling of data generated from these new elements would need to be added to the matchmaking library.

5.3 Structure of the Matchmaker Library

The Matchmaker library consists of three packages: Matchmaker.environment, Matchmaker.Filters and Matchmaker.MDS. The latter two packages are structured to distinguish the factory methods from the search functions. The search functions are a set of helper classes for the factory methods which perform most of the resource filtering operations. This distinction between factory methods and search functions is helpful in two scenarios - one where the user wants to use the framework for matchmaking and the other where the user wants to use only the search modules. For example, if a user needs to locate machines with a certain user account, he does not need an SDL file. The coarse grain search module can be called directly to filter out the resources. This flexibility makes the matchmaking library more useful.

Each of the packages is described in subsections 5.3.1 to 5.3.3. An overview of each package is given and the names of the constituent classes are described.

5.3.1 Matchmaker.environment package

As the name suggests, the environment package serves as an environment setup package. The environment variables present in this package need to be set up only once at installation time. It contains java and text files for setting up the matchmaking environment. The environment setup files are as follows.

a. Constants.java: It contains static strings with predefined values which can be altered by the site administrator for all users at installation time. The assumption made is that a hierarchical GIIS is setup for the site and all the local GIISs report to a centralized GIIS. The configurable variables for the package are:
   - MDSHOST- The server name for the centralized GIIS
   - MDSPORT- The port number on which the GIIS is running
   - Search_Base- The DN for the site from which the required entries need to be searched.
   - FilterOptionspath- The path to the file that contains mappings from the filter names to the implemented factory classes. The filternames are used by the user to specify the filters and the corresponding class names are used by the dynamic classloader.
   - log4jpropertiespath- The path to the log4jproperties file.
   - pathtoscript- The path to the perl script for testing fine grained access permissions on files. This path is the path to the script on the remote machine for which the fine grain permissions need to be checked.

b. SDLConstants.java: This class contains static strings containing elements and attributes for an SDL file. If the SDL schema needs to be extended the new elements can be added and the corresponding entries should be made in this class. These constants are used in the programs instead of literals.

c. log4jproperties: This configuration file contains settings for using the log4j framework. The log4j software provides a simple logging mechanism for applications. The log4jproperties file also sets up a path for the log file.

5.3.2 Matchmaker.Filters package

The filters package contains factory classes which involve the creation of objects which handle filtering based on a particular criterion. It contains all the main classes which form the framework. The following controller classes and factories are contained in this package.

a. Initiator.java: It is the controller class for the framework. It is the entry point for the matchmaking framework. It contains a single member function, match. This function accepts as input the SDL file of requirements and a hashtable containing the filter options. This SDL file is passed as a parameter to another class that parses and then creates an object containing the values from the SDL file. Depending on the filtering options in the hashtable the Initiator calls the ConcreteFactory to return the appropriate instances of filtering options to perform their function. The match method for each of these factories is initiated for matchmaking.

   The match method returns a hashtable containing the subset of machines that fulfills the criteria. If the set is empty the initiator writes out a log mentioning that none of the resources matched the requirements and exits.

b. JobReq.java and CreateModel.java: The CreateModel class parses the SDL file to generate an object containing the job requirements defined. This object is an instance of the JobReq class. The JobReq class contains a set of member variables to store the values parsed by the CreateModel class. These helper classes for the matchmaking modules allow parsing to take place only once during the matchmaking process. The JobReq object is created once, requiring only one pass over the SDL file. For the rest of the matchmaking process this object is passed around to modules which need the values. The JobReq class has accessor and modifier functions for each member variable and so it is easy to add additional elements to the SDL file and then write a corresponding set of accessor modifier modules to the JobReq class. The number of lines of code to incorporate extensions in SDL is minimal.

c. AbstractModule.java: This is the abstract class that contains the interface to be implemented by its subclasses (see example subclasses in Section 5.3.3), which are the factory classes for the filters. It contains a member function called match which accepts the JobReq object and a hashtable containing the set of filtered resources.

d. ConcreteFactory.java: This class is the subclass of the controller class, Initiator. It is unarguably the most important class in the Matchmaker framework. It makes decisions regarding the instantiation of factory objects. The decision is based on the entries in the file pointed to by the FilterOptionspath variable in the environment package. This file plays a similar role as the web.xml document in the case of servlets where the mapping from a generic servlet name to a class name is used to load the right class in the browser VM. In this case the mapping from the generic factory name to the implemented class is used to load the appropriate factory class.

5.3.3 Factory classes and helper classes

The factory classes are subclasses of the AbstractModule class and implement its defined interface. All the factory classes (part of Matchmaker.Filters) use helper classes (part of Matchmaker.MDS) to perform the search for the appropriate resources which fulfill certain criterion. These helper classes can be used by programmers outside the Matchmaker framework for other purposes.

The following subclasses have been implemented to illustrate the usefulness of the matchmaking framework. These subclasses are explained in context with the relevant helper classes.

a. **CoarsegrainFactory.java**: This factory method is used to check for coarse grain access rights for users of the grid resources. This factory uses a helper class called AuthenticUserSearch. In the section on Globus setup (Section 5.1) an authentication provider for the MDS was mentioned. This provider uses the information in the grid-mapfile of every resource in the infrastructure to generate the required authentication information. This information is reported to the centralized GIIS server for the site. The AuthenticUserSearch object extracts the proxy certificate for the user running the Matchmaker. The location for the proxy certificate is configurable and is stored in a file called cog.properties. The distinguished name for the user is extracted from the certificate and is provided as a query to the MDS to retrieve the records related to this DN. The hosts on which the user is authenticated can be extracted from the retrieved LDIF records. The query and manipulation of the generated search results is done using JNDI. The output from this module is an intersection of the matched resources with the input set of resources.

b. **DataLocalFactory.java**: This particular module filters resources based on data locality. High performance applications generally involve huge amounts of data, e.g., the BLAST application mentioned in Chapter 6 uses huge databases. Some of these databases run into gigabytes of data and it would not make sense to transfer all this data across the network. The more efficient way would be to execute BLAST on the machine that hosts the database. The information regarding the machines and the databases it houses needs to be catalogued and is stored in an ascii file called the replica catalog.

This catalog was modeled after the Globus Replica catalog [37]. The catalog file contains logical collections of files belonging to a single application. For example, all the files and databases related to the BLAST application can be listed under a single collection. Each of the entries in the collection has a generic name which is mapped to machines and the entire path to the particular file or database on that machine. The filter gets the input files and looks up the replica catalog for the corresponding entries. A sample replica catalog file containing a logical collection of files is listed in Figure 5.3.

```
BLASTLogicalFileCollection vector sample.fasta
vector      http://nazgul.bioinformatics.vt.edu/home/condor/data/jkasarko/remote.bin/BLAST/db/vector
http://aagrawal.bioinformatics.vt.edu/home/condor/data/jkasarko/remote.bin/BLAST/db/vector
http://jkasarko.bioinformatics.vt.edu/home/condor/data/jkasarko/remote.bin/BLAST/db/vector
sample.fasta
http://jkasarko.bioinformatics.vt.edu/home/condor/data/jkasarko/remote.bin/BLAST/db/sample.fasta
http://nazgul.bioinformatics.vt.edu/home/condor/data/jkasarko/remote.bin/BLAST/db/sample.fasta
```

Figure 5.3 : Sample Replica Catalog

The BLASTLogicalFileCollection contains entries for the databases and sequence files used by the BLAST users of a particular group. The database and sample sequence form one logical collection of files used by a particular group and so they are grouped together to simplify the search for a particular BLAST run. As seen in the entries the database vector is matched to three machines nazgul, aagrawal and jkasarko. The complete path to the database present on the corresponding node is specified. If a complete match of the input files specified in the SDL is found in the replica catalog then the helper class, Matchmaker.MDS.DataSearch, extracts the machine names and then finds the intersection of this set with the input set of machines. If no match is found the filter fails and exits from the Matchmaker with a message in the logs specifying that none of the resources matched the filter requirements.

c. **SysReqFactory.java**: This factory class is responsible for returning a set of resources which meet the system requirements defined in the SDL description for the job. At present system requirements are restricted to cpu utilization, amount of free RAM, virtual and filesystem memory and the operating system name and release. This information about each resource is retrieved from the MDS. The centralized GIIS server for the site contains the aggregate information about all the grid resources. This information is queried using the JNDI API. This factory, like the other factories, implements the interface defined by the AbstractModule class and hence the method accepts as input a JobReq object. The required elements for the search are extracted from the object and the LDAP query for the search is formed using its method formFilter. This is passed on to its helper class in the MDS package, SystemSearch.

The SystemSearch class implements the search method which accepts as input two string parameters. One of them is the partial LDAP query generated by the formFilter method and the second is the ranking metric mentioned in the SDL. This ranking attribute is used to generate a ranked list of machines. For example, the ranking attribute could be amount of available RAM. The machines which match the other system requirements defined will then be ranked among themselves based on the amount of free memory available. The search method first generates a set of machines which match the set of requirements defined in the partial LDAP query defined by the formFilter option.

For ranking the machines based on a certain attribute a quicksort method of ranking objects based on a certain attribute has been implemented. Most of the LDAP servers now implement server side sorting of LDIF record but this functionality is absent with MDS. The sorting is hence performed on the client side by a helper class, Qsort which exists in the MDS package. This class sorts objects which are instances of the ResObject class. The ResObject instance contains the name of the machine and the corresponding ranking attribute value for the machine. These objects are populated by the information retrieved when the MDS was queried hence the query is performed only once. The generated objects are stored in an array and passed on to the Qsort class which sorts these objects using the quick sort algorithm and returns the sorted array to the calling program which in this case is the SystemSearch class. This ranked set is then returned to the factory class which finds the appropriate intersection of the input set and the matched set.

d. **FineGrainFactory.java**: This factory class is used to check the fine grain[22] permission for file access for the user running the Matchmaker facility. This module assumes that the user knows the complete path to the executables, input files and the output working directory as described in the SDL. The check is made by staging a script to each of the machines in the input set of resources passed to the factory module. This is the only module which starts from the input set of resources and filters the resources directly from the input set. It contacts each of the resources and runs a script on each of these machines which is time consuming and is proportional to the number of machines in the input set. Hence a user should prefer running this module at the very end of the matchmaking process to improve efficiency. The helper class for the factory is the FineGrainSearch class. The permissions needed on each of the files are deduced from the SDL. The user would need read permission on the input files, execute permissions on the executable and write permissions for the working directory. A perl script is used to check on the required permissions on the files and returns a status value. This value is based on the result of the permission check for all the files. If it returns with a value 1 the machine contains files which can be used by the job without throwing permission errors.

This perl script is copied to the remote machine for which the access permissions need to be checked. The destination path is defined in the Constants.java class by the static string pathtoscript. A helper class to transfer the file from the machine running the Matchmaker to the destination machine is present in the MDS package. The class MyGridFTP takes as input the name of the remote machine to which it needs to connect. This information is used by its dispatch method to write a file to the destination machine which acts as the server. The first assumption is that the user has write access to the destination directory and that the set of machines being tested for fine grain permissions have already been filtered on the basis of coarse grain access. In the default installation the destination directory is /tmp.

To be able to run the perl script on each of these machines and be able to retrieve the output from each of these machines a GridJob class has been implemented. To be able to redirect the output from the remote console back to the calling program on the submitting machine a GASS [38] Server is started locally and the output is requested from the machine to which the job was submitted. This helper class is called by the FineGrainSearch which passes it the name of the remote machine and a boolean value to set the submission mode to interactive instead of batch submission. Once the setup is complete the run method of the GridJob class is invoked with an RSL string as input. This string is a description of the job with the path to the perl script executable on the remote machine and the arguments are the complete pathnames to the files and the related permissions that have to be checked. Based on the output redirected from the remote machine to the calling program the FineGrainSearch object decides on either retaining or removing the machine from the input set of machines. This filtered set of machines is returned to the FineGrainFactory object.

Chapter 6. Example Scenarios

We have chosen the standalone Basic Local Alignment Search Tool (BLAST) [39] as our test application. To exemplify the usage of the matchmaking framework in the case of BLAST we present two cases. In the first case an incomplete set of minimum requirements leads to a failure in submitting the task. In the second case a complete set of requirements with two different ranking criteria are presented which leads to different sets of ranked resources.

BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs designed to explore all of the available sequence databases. The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm which seeks local alignments and is therefore able to detect relationships among sequences which share only isolated regions of similarity. The inputs to the BLAST program are the sequence file and the related database to perform its search operations. The sequence databases are huge and transporting these files across the network is an overhead that should be avoided during a computational process. Hence the alternative is to move the sequence file to the resource holding the database, if possible.

The output of the search operation can be written to a file or redirected back to the terminal on the submission machine. A simple interface to the BLAST program was created to test the Matchmaking framework. The Resource Allocator service was replaced by the user's choice of a resource from the set of resources returned by the Matchmaker. The job is submitted to a fork job manager provided by the GRAM service.

In Section 6.1 the creation of the interface for the standalone BLAST program is presented. In Section 6.2 we present two different use cases. In the first case the user ignores the requirement of data locality and fine grain access permissions to the database. The second case has two different scenarios within it where the requirements remain the same but the ranking criteria changes; this results in a change in the choice of resource for the task.

6.1 Simple BLAST interface

For demonstration purposes a simple interface to BLAST was developed. The parameters to be set at installation time are as follows.

- Centralized GIIS server name

```
/*Constant for the centralized GIIS server hostname */
      public final static String MDSHOST="nazgul.bioinformatics.vt.edu";
```

- The port number for the GIIS service

```
/**
```

```
  *Constant for the port number of the centralized GIIS server
 **/
  public final static String MDSPORT="2135";      //Port for the GIIS
```

- Search base for the virtual organization on the GIIS. It acts as the starting point for the searches made by the matchmaking service querying for the collection of resources in that particular virtual organization.

```
/*
 *Constant for the DN of the site
 */
public final static String Search_Base="Mds-Vo-name=vbicentral, o=Grid";
```

- Path to the file containing the mappings of the filtername to the implemented factory method.

```
/*
 *Constant for the path to the file containing the mappings of the filter name to the
        implemented factory method
 */
 public final static String
        FilterOptionspath="/home/jkasarko/Matchmaker/Filters/FilterOptions.txt";
```

- Path to the log4j properties file.

```
  /*
   *Constant for the path to the log4j.properties file
   */
  public final static String
        log4jpropertiespath="/home/jkasarko/Matchmaker/environment/log4j.prop
        erties";
```

- Path to the perl script which checks for the fine grain permissions.

```
  /*
   *Constant for the path to the file permission test perl script
   */
 public final static String pathtoscript="/tmp/filePerms.pl";
```

To access the functionality of the Matchmaker the interface calls the match function of the Initiator class, as seen in Figure 6.2. This function accepts an SDL file and a table of filter names as input parameters. The path to the SDL file is provided as a commandline argument. The hashtable values are set in the interface program, as shown in Figure 6.1. In this figure all the default options available with the Matchmaker service are activated. Each criterion is given a rank which is specified by its corresponding key in the hashtable data structure. The filtering is performed in an ascending order of key numbers.

```
/*setting up the hashtable of options*/
Hashtable options=new Hashtable();
options.put(String.valueOf("1"),"Coarsegrained");
options.put(String.valueOf("2"),"System Requirements");
options.put(String.valueOf("3"),"Data Locality");
options.put(String.valueOf("4"),"Fine Grain");
```

Figure 6.1 : Setting up filters for Matchmaking

Once the filter names are set up the Matchmaker service is activated by a single function call to the Initiator class as shown in Fig 6.2.

```
/*calls*/
Initiator init= new Initiator();
Hashtable resources=init.match(args[0],options);
```

Figure 6.2 : Initiation of the Matchmaker service

Each of the resources in the set of resources returned by the Matchmaker module is suitable for the current task requirements. The most suitable resource would be selected by the Resource Allocator; in its absence the user chooses the appropriate entry from the hashtable. The task is then submitted to the fork job manager for the chosen resource.

6.2 Use scenarios

Case 1: BLAST without data locality and fine grain permission checks

In this first case we present a situation in which the user performs filtering based only on authorization and minimum system requirements. This set of requirements provides a set of resources from which the user selects the first ranked resource for submitting a BLAST task. Unfortunately that particular machine cannot provide it to the user since the user does not have the required permissions to access the database on that machine. The SDL generated for this task is shown in Fig 6.3.

The SDL describes the task input and output files and the minimum system requirements needed. The task needs

- a minimum average of 10 percent free cpu time per minute
- a minimum of 10000MB of free filesystem space
- a minimum of 10MB RAM
- a minimum of 40MB Virtual memory
- Linux operating system

The executable for the task is blastall and its entire path name is provided. The input files are the sequence file called vector and the database called sample.fasta. The output file is OUT and is written to the working directory which is /home/jkasarko as specified in the SDL. The resources will be ranked on the basis of average cpu free time over a period of one minute. The log for this matchmaking process is shown in Figure 6.4.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<simulation>
        <host></host>
        <no_procs>1</no_procs>
        <working_directory>/home/jkasarko</working_directory>
        <executable>/home/condor/data/jkasarko/remote.bin/BLAST/blastall</executable>
        <stdout>OUT</stdout>
        <input_files nfiles="2">
               <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/vector</file>
               <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/sample.fasta</file>
        </input_files>
        <cpufree average="1min" equality="greaterthan">10</cpufree>
        <fsfree equality="greaterthan">10000</fsfree>
        <ramfree equality="greaterthan">10</ramfree>
        <vmfree equality="greaterthan">40</vmfree>
        <osname>Linux</osname>
        <rank>cpufree</rank>
</simulation>
```

Figure 6.3 : SDL for ranking based on average number of free cpu cycles

```
2003-07-03 10:38:39,604 [main] INFO  Matchmaker.Filters.Initiator -
          ************************Entering Matchmaker***************************
2003-07-03 10:38:39,863 [main] INFO  Matchmaker.Filters.CoarsegrainFactory - <--Coarsegrain filtering
          starts-->
2003-07-03 10:38:41,209 [main] INFO  Matchmaker.MDS.AuthenticUserSearch - Now searching
          ldap://nazgul.bioinformatics.vt.edu:2135 for (&(object
class=MDSJKExtension)(Mds-authentic-user-DN=/EMAIL=jkasarko@vt.edu/CN=Jeevak
          Kasarkod/OU=Virginia Tech User/OU=Class 2/O=vt/C=US))
2003-07-03 10:38:48,721 [main] INFO  Matchmaker.Filters.Initiator - Rank 3:bchennup
2003-07-03 10:38:48,722 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :jkasarko
2003-07-03 10:38:48,722 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-03 10:38:48,741 [main] INFO  Matchmaker.Filters.SysReqFactory - <--System Requirements
          filtering starts-->
2003-07-03 10:38:48,751 [main] INFO  Matchmaker.MDS.SystemSearch - Now searching
          ldap://nazgul.bioinformatics.vt.edu:2135 for (&(objectclass=MdsComputer)(Mds-Cpu-Smp-
          size>=1)(Mds-Cpu-Free-1minX100>=10)(Mds-Fs-freeMB>=10000)(Mds-Memory-Vm-
          freeMB>=40)(Mds-Memory-Ram-freeMB>=10)(Mds-Os-name=Linux))
2003-07-03 10:38:51,838 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
          1minX100: nazgul value:106
2003-07-03 10:38:51,839 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
          1minX100: bchennup value:186
2003-07-03 10:38:51,839 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
          1minX100: jkasarko value:100
2003-07-03 10:38:51,840 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :jkasarko
2003-07-03 10:38:51,840 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :bchennup
2003-07-03 10:38:51,840 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :nazgul
2003-07-03 10:38:51,840 [main] INFO  Matchmaker.Filters.Initiator - RESULT: Final Hostnames of
          machines which match all the requirements
2003-07-03 10:38:51,841 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 Resource jkasarko
2003-07-03 10:38:51,841 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 Resource bchennup
2003-07-03 10:38:51,841 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 Resource nazgul
2003-07-03 10:38:51,841 [main] INFO  Matchmaker.Filters.Initiator -
          ***********************Exiting Matchmaker************************
```

Figure 6.4 : Log File for case 1

The log provides the details of the filtering performed by each of the filters. The first filter in this case is the coarse-grained access filter and is followed by the systems requirement filter. The coarse grained filter performs filtering based on authorization based on entries in the grid mapfiles of the resources belonging to the virtual organization. The final set of filtered resources is listed with a random rank attached to each entry. This phase is followed by the system requirement filtering phase which looks for resources with the minimum system requirements mentioned in the SDL. This set of requirements is used to extract a subset of suitable resources from the resultant set of the previous phase. From the logs it is evident that two resources were filtered out from the set of resources returned by the first phase.

The error message at the time of submission to the first ranked resource in the set of returned resources is presented in Figure 6.5. The GRAM server complains about the access permissions of the user to the database.

```
Sending job request to: bchennup
Error submitting job: class org.globus.gram.GramException:Cannot access cache files in
        ~/.globus/.gass_cache, check permissions, quota, and disk space
```

Figure 6.5 : Error output for case 1

Case 2: BLAST with all four filters activated

In contrast to the previous case, in this case the four default options for filtering are activated. This filtering returns a smaller subset of resources compared to case 1. To display the ranking property provided by the matchmaking framework based on system requirements, two sub-cases have been presented: in the first, ranking is based on free CPU cycles per minute, and in the second ranking is based upon the amount of free filesystem space.

Use case 2a: Ranking based on average free CPU time available per minute

The SDL file used for this case is the same as case 1 and is shown in Figure 6.3. The SDL file specifies the minimum system requirements and the input and output parameters for the task. This is used by the Matchmaker for filtering and to create the resource specification language (RSL) string for task submission through GRAM. The log generated for this case is shown in Figure 6.6.

The SDL for use case 2a and for case 1 is the same but the final set of returned resources are completely different. In case 1 the final set of resources contained eight machines but in use case 2a the final set contains only two resources which fulfill all four requirements presented, including data locality and fine grain access permissions.

```
2003-07-02 15:41:03,001 [main] INFO  Matchmaker.Filters.Initiator -
        *************************Entering Matchmaker***************************
2003-07-02 15:41:03,193 [main] INFO  Matchmaker.Filters.CoarsegrainFactory - <--Coarsegrain
        filtering starts-->
2003-07-02 15:41:04,396 [main] INFO  Matchmaker.MDS.AuthenticUserSearch - Now
        searching ldap://nazgul.bioinformatics.vt.edu:2135 for (&(object
class=MDSJKExtension)(Mds-authentic-user-DN=/EMAIL=jkasarko@vt.edu/CN=Jeevak
        Kasarkod/OU=Virginia Tech User/OU=Class 2/O=vt/C=US))
2003-07-02 15:41:11,485 [main] INFO  Matchmaker.Filters.Initiator - Rank 5 :aagrawal
2003-07-02 15:41:11,485 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :dana
2003-07-02 15:41:11,486 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :txue
2003-07-02 15:41:11,486 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :jkasarko
2003-07-02 15:41:11,486 [main] INFO  Matchmaker.Filters.Initiator - Rank 6 :wsun
2003-07-02 15:41:11,486 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-02 15:41:11,491 [main] INFO  Matchmaker.Filters.SysReqFactory - <--System
        Requirements filtering starts-->
2003-07-02 15:41:11,497 [main] INFO  Matchmaker.MDS.SystemSearch - Now searching
        ldap://nazgul.bioinformatics.vt.edu:2135 for (&(objectclass=M
dsComputer)(Mds-Cpu-Smp-size>=1)(Mds-Cpu-Free-1minX100>=10)(Mds-Fs-
        freeMB>=10000)(Mds-Memory-Vm-freeMB>=40)(Mds-Memory-Ram-
        freeMB>=10)(Mds-Os-name=Linux))
2003-07-02 15:41:14,559 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
        1minX100: nazgul value:198
2003-07-02 15:41:14,559 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
        1minX100: wsun value:100
2003-07-02 15:41:14,559 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
        1minX100: dana value:100
2003-07-02 15:41:14,559 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Cpu-Free-
        1minX100: jkasarko value:100
2003-07-02 15:41:14,560 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :jkasarko
2003-07-02 15:41:14,560 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :dana
2003-07-02 15:41:14,560 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :wsun
2003-07-02 15:41:14,560 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-02 15:41:14,564 [main] INFO  Matchmaker.Filters.DataLocalFactory - <--Data Locality
        filtering starts-->
2003-07-02 15:41:14,573 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :jkasarko
2003-07-02 15:41:14,573 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :dana
2003-07-02 15:41:14,573 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-02 15:41:14,576 [main] INFO  Matchmaker.Filters.FineGrainFactory - <--Fine Grain
        Access filtering starts-->
2003-07-02 15:41:41,321 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :jkasarko
2003-07-02 15:41:41,322 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-02 15:41:41,322 [main] INFO  Matchmaker.Filters.Initiator - RESULT: Final
        Hostnames of machines which match all the requirements
2003-07-02 15:41:41,322 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 Resource jkasarko
2003-07-02 15:41:41,322 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 Resource nazgul
2003-07-02 15:41:41,322 [main] INFO  Matchmaker.Filters.Initiator -
        ***********************Exiting Matchmaker************************
```

Figure 6.6 : Matchmaker service log for use case 2a

Use case 2b: Ranking based on free filesystem space

To demonstrate the ranking functionality of the system requirements module a second use case has been presented with a different ranking criteria. The SDL file shown in Figure 6.7 is similar to the one in Figure 6.3 but the ranking attribute has changed. The log for use case 2b is presented in Figure 6.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<simulation>
    <host></host>
    <no_procs>1</no_procs>
    <working_directory>/home/jkasarko</working_directory>
    <executable>/home/condor/data/jkasarko/remote.bin/BLAST/blastall</executable>
    <stdout>OUT</stdout>
    <input_files nfiles="2">
        <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/vector</file>
        <file>/home/condor/data/jkasarko/remote.bin/BLAST/db/sample.fasta</file>
    </input_files>
    <cpufree average="1min" equality="greaterthan">10</cpufree>
    <fsfree equality="greaterthan">10000</fsfree>
    <ramfree equality="greaterthan">10</ramfree>
    <vmfree equality="greaterthan">40</vmfree>
    <osname>Linux</osname>
    <rank>fsfree</rank>
</simulation>
```

Figure 6.7: SDL for ranking based on free filesystem space

The final set of resources is the same as in use case 2a but the ranking has changed.

The two cases presented exemplify the need to cover all aspects of minimum requirements filtering when choosing the right set of resources for the task to execute. System requirements alone do not fulfill all the needs; other requirements such as the existence of a user account and the appropriate fine grain access permissions must be considered as well.

```
2003-07-02 15:32:18,629 [main] INFO  Matchmaker.Filters.Initiator -
          ************************Entering Matchmaker**************************
2003-07-02 15:32:19,059 [main] INFO  Matchmaker.Filters.CoarsegrainFactory - <--Coarsegrain
          filtering starts-->
2003-07-02 15:32:21,118 [main] INFO  Matchmaker.MDS.AuthenticUserSearch - Now
          searching ldap://nazgul.bioinformatics.vt.edu:2135 for (&(object
class=MDSJKExtension)(Mds-authentic-user-DN=/EMAIL=jkasarko@vt.edu/CN=Jeevak
          Kasarkod/OU=Virginia Tech User/OU=Class 2/O=vt/C=US))
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 6 :aagrawal
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 5 :jshah
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :dana
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :txue
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :jkasarko
2003-07-02 15:32:30,195 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :nazgul
2003-07-02 15:32:30,211 [main] INFO  Matchmaker.Filters.SysReqFactory - <--System
          Requirements filtering starts-->
2003-07-02 15:32:30,235 [main] INFO  Matchmaker.MDS.SystemSearch - Now searching
          ldap://nazgul.bioinformatics.vt.edu:2135 for (&(objectclass=MdsComputer)(Mds-Cpu-
          Smp-size>=1)(Mds-Cpu-Free-1minX100>=10)(Mds-Fs-freeMB>=10000)(Mds-Memory-
          Vm-freeMB>=40)(Mds-Memory-Ram-freeMB>=10)(Mds-Os-name=Linux))
2003-07-02 15:32:33,303 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Fs-
          freeMB: jkasarko value:33265
2003-07-02 15:32:33,304 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Fs-
          freeMB: nazgul value:31828
2003-07-02 15:32:33,304 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Fs-
          freeMB: aagrawal value:14048
2003-07-02 15:32:33,304 [main] INFO  Matchmaker.MDS.SystemSearch - Max Mds-Fs-
          freeMB: dana value:13232
2003-07-02 15:32:33,305 [main] INFO  Matchmaker.Filters.Initiator - Rank 4 :dana
2003-07-02 15:32:33,305 [main] INFO  Matchmaker.Filters.Initiator - Rank 3 :aagrawal
2003-07-02 15:32:33,305 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :nazgul
2003-07-02 15:32:33,305 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :jkasarko
2003-07-02 15:32:33,311 [main] INFO  Matchmaker.Filters.DataLocalFactory - <--Data Locality
          filtering starts-->
2003-07-02 15:32:33,363 [main] INFO  Matchmaker.Filters.Initiator - Rank 6 :dana
2003-07-02 15:32:33,363 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :nazgul
2003-07-02 15:32:33,363 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :jkasarko
2003-07-02 15:32:33,369 [main] INFO  Matchmaker.Filters.FineGrainFactory - <--Fine Grain
          Access filtering starts-->
2003-07-02 15:33:00,809 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 :nazgul
2003-07-02 15:33:00,810 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 :jkasarko
2003-07-02 15:33:00,810 [main] INFO  Matchmaker.Filters.Initiator - RESULT: Final
          Hostnames of machines which match all the requirements
2003-07-02 15:33:00,810 [main] INFO  Matchmaker.Filters.Initiator - Rank 2 Resource nazgul
2003-07-02 15:33:00,810 [main] INFO  Matchmaker.Filters.Initiator - Rank 1 Resource jkasarko
2003-07-02 15:33:00,810 [main] INFO  Matchmaker.Filters.Initiator -
          ***********************Exiting Matchmaker***********************
```

Figure 6.8 : MatchMaker service log for user case 2b

Chapter 7. Conclusion and Future Work

Grid computing is the emerging infrastructure for next generation computing. Utility computing will be a direct outcome of advances made in grid computing middleware services. This thesis is an effort in providing user level middleware services for resource management that provides a flexible job scheduling and initiation system for any kind of application and computing environment.

7.1 Conclusion

In the previous chapters we have outlined a system architecture for decoupled job scheduling and initiation on the grid. In Chapter 1 we emphasized that MFATIC is designed to provide flexibility in resource brokers to accommodate multiple users and computing environments and to provide user control over every phase of the scheduling process.

MFATIC provides a choice of scheduling policies, requirements filtering and job submission mechanisms to the user. In addition to choice in every phase of the scheduling process, MFATIC provides a choice of components to plug into the system architecture. For example, the user can prefer to use a component provided by another vendor in place of the Resource Allocator to obtain a service level agreement (SLA). This provides a pluggable task scheduling and initiation system.

MFATIC components run on the user node and hence provide a decentralized scheduling mechanism. Decentralized scheduling mechanisms are useful in scalable and dynamic computing environments like the grid.

Current resource brokers replicate most of the functionality. MFATIC provides all its functionality in packages and classes which are reusable. Developers can add their code to the framework which can then be used by other developers. The idea of factory services fits into the Open Grid Services Architecture (OGSA) [40] and with a few changes the Matchmaker library can be implemented as a grid service.

7.2 Future Work

The design for the entire system has been laid out but the system has not yet reached completion as far as implementation is concerned. The reason for this is that the implementation depends on a number of standards and technologies that are yet to be released (e.g, DRMAA, scheduling using SLA, GRAAP). The DRMAA implementation will be released soon; hence work on the Job Submission Module should be the next step to MFATIC. The Job Submission Module can be tested with resource management systems which have DRMAA support provided by its vendors. The Globus team is continuing work on GRAM to support scheduling with SLAs [27]. Depending upon the level of support for advance reservation provided by GRAM, the Resource Allocator functionality could change. Factors like two phase commitment protocols, if not supported by GRAM, need to be incorporated into the Resource Allocator.

Within the Matchmaker framework four default requirement filtering modules are supplied, but a few more modules based on economy and deadline constraints could be added. The current implementation of the data locality module is based on an ASCII file which maps the global data names to locations on the grid resources. This needs to be replaced by a grid file replication service which contains an LDAP directory containing all this information. (At the time of implementation there were problems adding entries to the Globus data replica catalog.) With the availability of a better replica catalog, the "naming" problem could be solved in a more general way. For example, the user interface module, Matchmaker, and Resource Allocator could use generic, global names for files, while only the job submission service would use the replica catalog to map these generic names to fully qualified machine and path names. The most general format for universally qualified pathnames would be URLs. However, the current DRMAA proposal assumes that file names adhere to the file naming scheme of the target host rather than to a general solution such as URLs. Hence, the replica catalog would have to record system-specific naming conventions for each file it records.

Another area of active research in grid computing is in the field of automatic checkpointing and migration of grid jobs. Currently Déjà Vu, a transparent checkpointing and migration framework for any parallel communication library that uses TCP/IP, is being developed at Virginia Tech. In the MFATIC framework once a node fails there is no possibility of rescheduling the job on a different resource. The programs can be linked against Déjà vu to enable checkpointing and migration. A monitoring framework such as the Distributed Monitoring Framework (DMF) can be used to detect a node failure which is reported to the MFATIC system. The checkpointed process can then be rescheduled on a different node chosen by MFATIC. This provides automatic fault tolerance for distributed applications.

Bibliography

1. **I. Foster,C. Kesselman,S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal For SuperComputing Applications, 2001. 15 (3).**

2. **R.Buyya, *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. 2002, Monash University: Melbourne, Australia, April 12, 2002.**

3. **Francine Berman,Henri Casanova,Graziano Obertelli,Rich Wolski. *User-Level MiddleWare for the Grid*. in *Proceedings of the Super Computing Conference (SC'2000).***

4. **A. Chervenak,I. Foster,C. Kesselman,C. Salisbury,S. Tuecke, *The Data Grid:Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets.* Journal of Network and Computer Applications, 2000. 23(Special Issue on Network-Based Storage Services): p. 187-200.**

5. **Michael Wan and Reagan Moore Arcot Rajasekar. *MySRB & SRB - Components of a Data Grid*. in *The 11th International Symposium on High Performance Distributed Computing (HPDC-11)*. 2002. Edinburgh, Scotland.**

6. **C. Kesselman . Foster, J. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. in *Global Grid Forum*. 2002.**

7. **Dan Gisolfi, *Web Servics Architect, Part 1: An introdtuction to dynamic e-business*, in *IBM developerworks (http://www-106.ibm.com/developerworks/webservices/library/ws-arc1/)*. 2001.**

8. **Dimitrios Katramatos Steve J. Chapin, John Karpovich, Andrew Grimshaw. *The Legion Resource Management System*. in *5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99)*. 1999.**

9. **N. Spring R. Wolski, C. Peterson. *Implementing a performance forecasting system for metacomputing: The network weather service.* in *SuperComputing 97*. 1997. San Jose,USA.**

10. **Miron Livny Michael Litzkow. *Experience With The Condor Distributed Batch System*. in *IEEE Workshop on Experimental Distributed Systems*. 1990.**

11. **Rajesh Raman Nicholas Coleman, Miron Livny, Marvin Solomon, *Distributed Policy Management and Comprehension with Classified Advertisements*. April 2003, University of Wisconsin-Madison Computer Sciences.**

12. J.Frey,T.Tannenbaum,I.Foster,M.Livny,S.Tuecke. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* in *Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10)*. 2001: IEEE Press.

13. Ben Segal. *Grid Computing: The European Data Project*. in *IEEE Nuclear Science Symposium and Medical Imaging Conference*. 2000. Lyon, France.

14. F. Giacomini,F.Prelz,M.Sgaravatto,I.Terekhov,G.Garzoglio,T.Tannenbaum, *Planning on the Grid: A Status Report.* 2002, Particle Physics Data Grid (PPDG).

15. Unknown, *DataGrid: Research and Technological development for an International Data Grid*. 2001.

16. I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit*. in *Workshop on Environments and Tools for Parallel Scientific Computing, SIAM*. 1996. Lyon, France.

17. Unknown, *Globus toolkit 2.2: MDS Technology Brief*. 2003.

18. I. Foster S. Fitzgerald, C. Kesselman, G. vol Laszewski, W. Smith, and S. Tuecke. *A Directory Service for Configuring High-Performance Distributed Computations.* in *Sixth IEEE International Symposium on High Performance Distributed Computing*. 1997.

19. Sudesh Agarwal, *Hardware software server in Netsolve*. 2002, Computer Science Dept, Univerisity of Tennesse, Knoxville: Knoxville,TN.

20. D. Abramson R. Buyya, and J. Giddy. *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid*. in *The 4th International Conference on High Performance Computing in Asia-Pacific Region*. 2000. Beijing, China.

21. I. Foster K. Czajkowski, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems*. in *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*. 1998.

22. V. Welch. K. Keahey. *Fine-Grain Authorization for Resource Management in the Grid Environment.* in *Proceedings of Grid2002 Workshop*. 2002.

23. Erich Gamma,Richard Helm,Ralph Johnson,John Vlissides, *Design Patterns*. p. 107 - 117.

24. J.M Schopf, *Ten Actions when superscheduling*. 2001.

25. GRAAP-WG, *Advance Reservations: State of the Art*. 2003, Grid Resource Allocation Agreement Protocol Working Group (Sched-GRAAP-WG).

26. H. Dail,H. Casanova,F. Berman. *A Decoupled Scheduling Approach for the GrADS Environment*. in *Supercomputing 2002*. Baltimore,USA.

27. Karl Czajkowski. *Grid Scheduling through Service-Level Agreement (http://www.gridsched.org/program.html)*. in *Global Grid Forum 7*. 2002.

28. V. Sander K. Czajkowski. *Grid Resource Management Protocol: Requirements*. in *GGF Scheduling working group*. 2001.

29. I. Foster K. Czajkowski, C. Kesselman, V. Sander, S. Tuecke, *SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems.* 2002.

30. Stephen Pickles Karl Czajkowski, Jim Pruyne, Volker Sander, *Usage scenarios for a Grid Resource Allocation Agreement Protocol*. 2003.

31. Hrabri Rajic, *DRMAA proposal (http://www.drmaa.org/docs/)*. 2001.

32. B.Tierney, *The Distributed Monitoring Framework (DMF) (http://www-didc.lbl.gov/DMF/)*. 2001.

33. Unknown, *Building Flexible, Large scale Application Architecture*. 2001.

34. Sheng Liang and Gilad Bracha. *Dynamic Class Loading in the Java Virtual Machine*. in *ACM Conf. on Object-Oriented Programming, Systems, Languages and Applications*. 1998.

35. USC/ISI, *MDS 2.2: Creating a Hierarchical GIIS*. 2002, http://www.globus.org/mds/hierarchical_GIIS.pdf.

36. A. Karnik and C. J. Ribbens, *Data and Activity Representation for Grid Computing*. TR-02-13, 2002, Computer Science, Virginia Tech.

37. REP-RG, *An Architecture for Replica Management in Grid Computing Environments.* 2001, Data Replication Research Group[REP], Global Grid Forum.

38. J. Bester, Foster, I., Kesselman, C., Tedesco, J. and Tuecke, S. *GASS: A data movement and access service for wide area computing systems*. in *Sixth Workshop on Input/Output in Parallel and Distributed Systems*. 1999.

39. W. Gish S. F. Altschul, W. Miller, E. W. Myers, and D. J. Lipman., *A basic local alignment search tool.* Journal of Molecular Biology, 1990.

40. C. Kesselman I. Foster, J. Nick, S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, in *Global Grid Forum(http://www.globus.org/research/papers.html)*. 23/08/2002.

Vita

Jeevak Kasarkod

Education

- M.S. in Computer Science and Applications (July 2003)
  Virginia Polytechnic and State University (Virginia Tech), Blacksburg, VA
  Thesis: A Configurable Job Submission and Scheduling System for the Grid.

- B.E. in Computer Engineering (August 2001)
  VESIT, Bombay, India
  Final Year Project: A message authentication system for network traffic

Work Experience

- May 2002 – July 2002: Collaboratory Technologies, DSD NERSC, Ernest Orlando Lawrence Berkeley Laboratory. Third party workflow submission tool and a visualization tool for the workflow engine as part of the PCCE project. (http://www-itg.lbl.gov/Collaboratories/pcce.html)

- Jan 2001 – present**:** Working as a Research Assistant at the Mendes Laboratory, Virginia Bioinformatics Institute (VBI)**.** The work involves research in Grid computing. Responsibilities: Setup a condor pool. Automate the installation of the Globus toolkit across VBI NOW. Authentication module for the BLAST portal. Provide the condor pool as a resource through Globus. Centralized authentication information about Globus users using LDAP. Condor flocking. Installing Condor-G. Implementing a grid accounting service using GT3. Implementing a virtual cluster service using user mode linux for checkpointing and migration mechanisms

Selected Projects

- Working with Markus Lorch on a project sponsored by IBM. Porting MyProxy (credential repository) server to use PKCS11 for the IBM4758 PCI cryptographic coprocessor
- N-Body Problem simulation in MPI with a visualization process.
- Implementing the selective repeat sliding window protocol for an unreliable transmission media.
- Implementing routing protocols distance vector and linkstate.
- J2EE E-Commerce Protocol: An online shopping stop for 802.11a/b PCMCIA cards.
- Developed a Library management system using C++
- Developed a lexer & parser for a subset of the C++ language

Relevant Courses

Graduate: Research Methods in CS, Advanced Operating Systems, Computer Network Architecture, Information Visualization (HCI), Parallel Computation, Network Performance Design and Management , Advanced course in grid computing and numerical software, Internet Software

Undergraduate: Computer Graphics, Database Management Systems, Software Engineering, Object Oriented programming, Computer Methodologies and Algorithms, Computer Networks and Communication, Advanced computer architecture, Systems Programming, Compiler Construction, Theoretical Computer Science

Skills

Languages	C, C++, Java, Pascal, Assembly, Python
Operating Systems	All flavours of Unix, Windows NT/95/98/2000, DOS
DataBases	Oracle8i, MySQL
Web Design	HTML, XML(JDOM and JAXP), J2EE,JSP, Apache Axis, Java beans
Grid MiddleWare	Globus, Condor
Other Technologies	OpenLDAP, Tomcat and Apache Webserver administration, MPI, OPNET, Matlab, Lex and Yacc, log4j, Multithreading in java and lpthreads in C, dll programming in Linux.
IDE	Eclipse