JOB SCHEDULING HEURISTICS FOR GRID

by

Mustafa Özgür Erbaş

B.S., Computer Engineering, Işık University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
The requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2007

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Assoc. Prof. Can Özturan, for his guidance, encouragement, kindness, and support through the whole study. I would like to mention his patience, giving me support and hope when I was stuck at dead-ends. Also, I would like to thank Haluk Bingöl and Tamer Dağ to be there for me.

I would like to thank my family, especially my father, for their patient, trust and sacrifices without hesitation. I will be indebted to you not only for this part of my life, but forever.

This thesis is dedicated to the memory of my beloved Zeynep Zengin, who was passed away and left me behind, always missing her.

Finally, I would like to mention my friends Ali Haydar Özer, Itır Karaç, and H. Gül Çalıklı for their patience. I thank them all.

ABSTRACT

JOB SCHEDULING HEURISTICS FOR GRID

Grid computing can be expressed as a set of services for sharing data, computation capacity and other resources like special equipment. Improvements in networking enabled grid technology to progress quite fast. Resource selection for jobs submitted in a grid, also called matchmaking, is one of the most important tasks needed for operating a grid. Matchmaking is a process that tries to assign jobs to available resources. One important goal of matchmaking is to maximize grid throughput. This is a difficult goal to realize because of the existence of heterogeneous resources in a grid. A widely used approach is Condor's matchmaking algorithm, which is used by SEE-GRID, EGEE and TR-Grid infrastructures. The goal of this study is to improve this algorithm to obtain better algorithms for the matchmaking process.

We propose two new polynomial algorithms for matchmaking. The idea shared by both of our proposed heuristic algorithms is that our heuristics take the collection of jobs and try to match as many jobs to available resources. One of our heuristics, called Scarce Resource First Matchmaking (SRFM), assigns by first trying to match scarce resources. The other heuristic called Linear programming Based Matchmaking (LBM) solves relaxed version of the NP-hard integer program and assigns resources by using relaxed solution values.

Our simulation results show that our collective matchmaking schemes work quite well by improving the number of completed jobs.

ÖZET

GRID İÇİN KAYNAK TAHSİS MODELLERİ

Hesaplama; veri, hesaplama kapasitesi ve diğer kaynakların(özel araçlar) paylaşımıın sağlayan servisler topluluğu olarak tanımlanabilir. Bilgisayar ağlarındaki gelişme Grid'in ilerlelemesinde çok etkili olmuştur. Grid'in çalışması için en önemli işlemlerden biri, işler için kaynak seçimi, diğer adıyla eşleştirmedir. Eşleştirme, yapılacak işlerin boştaki kaynaklara atanması işidir. En önemli hedeflerden biri grid'in verimliliğini artırmaktır. Kaynakların birbirinden çok farklı olduğu göz önünde bulundurulduğunda bu zor bir işlemdir. Eşleştirme için en popüler yaklaşımlardan biri SEE-GRID, EGEE ve TR-GRID'in de kullandığı Condor'un eşleştirme algoritmasıdır. Araştırmamızın amacı bu algoritmayı geliştirerek eşleştirme için daha verimli algoritmalar tasarlamaktır.

Bu çalışmada, eşleştirme için iki yeni algoritma öneriyoruz. Her iki algoritmada da yapılmaya çalışılan şey, işleri tek tek almak yerine, bir takım olarak almak ve mümkün olduğunca fazla işi boş kaynaklara atamaktır. Algoritmalarımızdan birisi, İlk Kıt Kaynaklar Eşleştirme (SRFM) algoritması, kıt kaynakları eşleştirmeye öncelik verir. Diğer algoritma is Lineer Programlama Tabanlı Eşleştirme (LBM) algoritması olarak adlandırılmıştır.

Simulasyon sonuçlarına göre, önerdiğimiz bu algoritmaların, tamamlanan iş sayısını atrırarak daha iyi bir şekilde çalıştığı görülmüştür.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

d_{window}	Duration of matchmaking
r_{jc}	The request of cluster c by job j
q_j	The number of processors requested by job j
p_c	The number of processors available on cluster c
d_j	The amount of disk space requested by job j
d_c	The amount of disk space available on cluster c
m_j	The amount of memory requested by job j
m_c	The amount of memory available on cluster c
$D(j,c)$	Resource c desire value for job j
w_{jc}	Weight of job j over for cluster c

CE	Computing Element
CERN	English: European Organization for Nuclear Research
ClassAd	Classified Advertisement
CSP	Constraint Satisfaction Problem
DAG	Directed Acyclic Graph
DAGMan	DAG Manager
FCFS	First Come First Served
GIS	Grid Information Service
GMW	Grid Middleware
GRAM	Grid Resource Allocation Management
GriPhyN	Grid Physics Network
HTC	High-Throughput Computing
IPG	Information Power Grid
IS	Information System(s)

ISM	Information SuperMarket
iVDGL	The International Virtual Data Grid Laboratory
JDL	Job Description Language
L&B	Logging and Bookkeeping
LBM	LP Based Match-making
LDAP	Lightweight Directory Access Protocol
LDS	Logical Dataset
LHC	Large Hadron Collider
LJF	Longest Job First
LM	Log Monitor
NCSA	National Computational Science Alliance
NMI	the NSF Middleware Initiative
NS	Network Server
PPDG	the Particle Physics Data Grid
RB	Resource Broker
RMS	Resource Management System
SE	Storage Element
SEE	South East Europe
SJF	Shortest Job First
SRFM	Scarce-Resource-First Match-making
UI	User Interface
VO	Virtual Organization
WM	Workload Manager
WMS	Workload Management System
WN	Worker Node

1. INTRODUCTION

1.1 Grid Computing

Grid computing [1] can be expressed as a set of services for sharing data, special equipments and computation capacity. Basically, the idea is to create a virtual super computer by using many distributed computers and storage. Grid computing is a good solution not only for complex computational problems but also simple problems. The main goal [2] is to create a self managing virtual computer by using large number of computers that are connected to each other. These computers share their resources and jobs are assigned available resources for the VO (Virtual Organization) that the users belong to.

1.2 SEE-GRID and EGEE Projects

SEE-GRID [3] is a regional computing and storage grid infrastructure. SEE-GRID-2 [4], which is the second phase of the project, mostly focuses on grid usability and grid applications. The research and education communities of South East Europe, also called SEE, will benefit from this project and its applications. The main goals of the project are to improve and integrate the existing grid infrastructure and services that have been developed by SEE-GRID.

Another important grid project is the Enabling Grids for E-sciencE project, also called EGEE [5, 6], which is used by scientists and engineers in 32 countries. It provides grid infrastructure for e-science. EGEE is supported by the European Commission. The project is used in many applications areas such as geology, computational chemistry, energy physics and life sciences. EGEE infrastructure has over 30,000 CPUs and about 5 Petabytes of storage.

1.3 TR-Grid

TR-Grid [7] the national grid infrastructure of Turkey developed by ULAKBIM and a number of Turkish universities. TR-Grid is a member of the SEE-GRID and EGEE infrastructures. The basic model of clusters in TR-Grid infrastructure as follows; First of all, latest version of gLite[8] middleware is installed over Computing Elements (CEs), Worker Nodes (WNs), and Storage Elements (SEs). gLite is the next generation middleware software for a grid. It provides a framework for grid applications using distributed resources. The gLite is a service oriented grid middleware which is providing services for managing distributed computing (including security), auditing and information services. Another important node type is Workload Management System (WMS). WMS is set of services that are responsible for resource allocation. However, all the clusters do not have to have this node because there is a centralized WMS which is created by TR-Grid. Detailed information about WMS will be given in Chapter 3.

1.4 Motivation

Resource selection is a critical task for grid infrastructure and can be described as finding a set of resources that satisfy job needs at the current resource state of grid. We have to ensure that not only jobs achieve desired requirements but also efficient utilization of grid resources is realized. We have to prevent excessive resource consumption. In other words, we need an efficient resource selection scheme to maximize throughput, because finding a satisfying resource for the job does not mean finding a resource that best fits the job when we take other jobs' requirements into account. By using the most appropriate resources for the jobs we will be able to avoid excessive resource consumption. Consequently this approach will maximize system throughput. This can be a difficult task to realize because jobs are matched to heterogeneous resources which have dynamically changing characteristics such as memory, disk and CPU capacities.

We focus on attribute-based selection, namely Condor's Matchmaking, and Gangmatching algorithms. Matching is done between jobs and resources, and they are

described by using ClassAds. Matchmaking and gangmatching algorithms find compatible matches between job ClassAds and resource ClassAds. In addition, there is a *Rank* parameter which helps the algorithm in coming up with the best matching. Condor's Matchmaking algorithm is explained in detail in Chapter 2 but basically algorithm;

1. Iterates through the entire solution space,
2. Determines amongst all compatible ClassAds,
3. Chooses jobs with the highest "rank" value.

Assignments of jobs are done using FCFS (First Come First Served) scheduling in TR-Grid. Different scheduling methods can be used and details about these methods are given in Chapter 3. Here the problem is that these scheduling methods do not make the best match. Figure 1.1 demonstrates a simple scenario for scheduling.

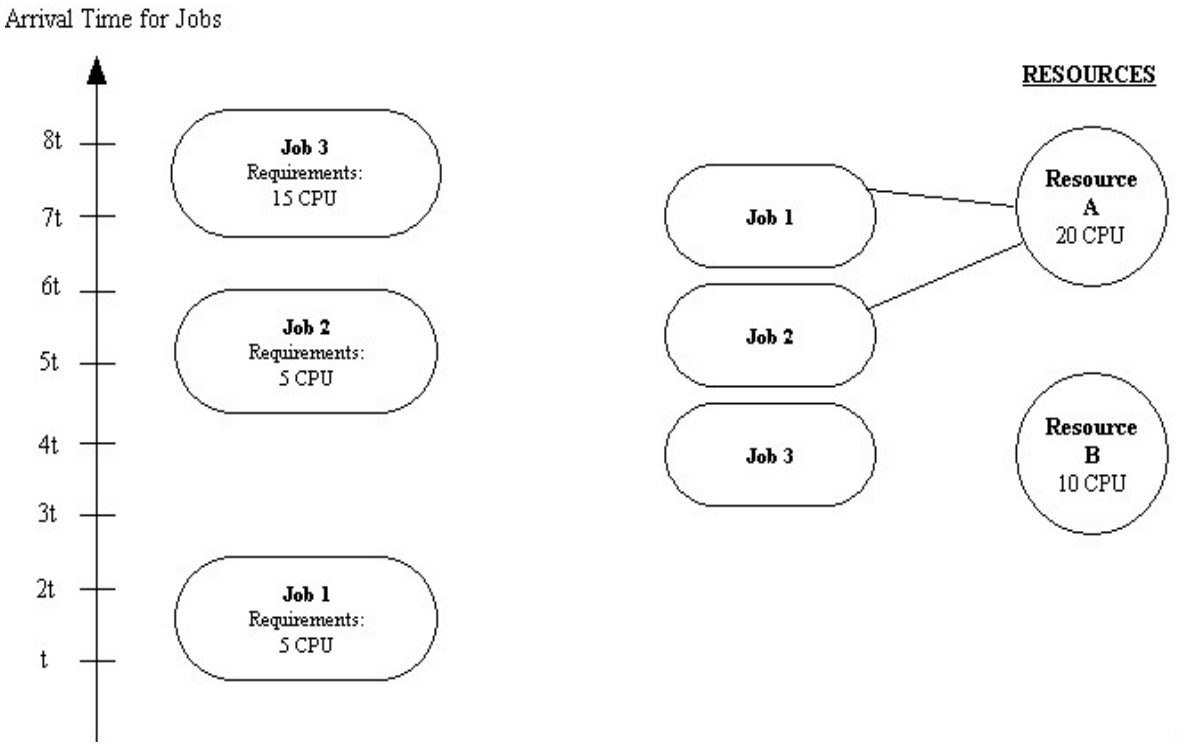


Figure 1.1. Simple Assignment Scenario

Consider the simple scenario of three jobs that have arrived one after the other as shown in Figure 1. Suppose that *job 1* has the highest, *job 2* has the second highest and *job 3* the lowest priorities. Suppose there are two resources: cluster machine *resource A* with 20 processors and *resource B* with 10 processors. Assume that these resources are vacant and are all equally suitable for running all the jobs. If we take jobs according to priority order, one at a time, Condor would first take job 1 and try to assign a resource to it. Since it is doing this without consideration of other jobs, it is possible that it may assign 5 processors of resource A to it. Then it would take job 2 and it is possible that it may assign 5 processors of resource A to it. Finally, when job 3, which needs 20 processors, is to be scheduled, it will fit neither of the remaining 10 processors of resource A or on resource B which has 10 processors. Assuming that a parallel job needs to run on a single cluster (due to for example networks being slow), then we will not be able to run job 3. Since it can only run on resource A, it will have to wait for jobs 1 and 2 to finish before it can run. If we take all the jobs and all the resources collectively, however, and try to match them simultaneously, we see that all three jobs can be assigned resources they need. Job 1 and Job 2 can be assigned to resource B and Job 3 can be assigned to resource A.

The main idea of our approaches is to avoid excessive resource consumption so that we increase the system throughput. Our study is concluded with experimental tests from real life scenarios by using GridCAM, which is a simulation tool developed by us. GridCAM simulates both FCFS algorithm and our two proposed approaches. In addition, GridCAM uses TR-Grid infrastructure real resource capacities and architectures such as disk space, number of CPUs, memory space, and operating system. GridCAM is inspired and named after “WebCam”. The name mentions “grid camera”. Details about GridCAM simulation tool will be given in Chapter 3.

1.5 Contributions

Contributions of our work can be expressed as follows;

- A simulation Tool, called GridCam, has been developed. GridCam creates a virtual environment for TR-Grid structure.

- Two new heuristics Scarce Resource First (SRFM) and LP Based Matchmaking (LBM) were introduced for matchmaking.
- Performance evaluations of FCFS, SRFM and LBM that are done by using GridCAM.

1.6 Outline

In Chapter 2, previous work related to matchmaking and scheduling will be considered. In the first section of this chapter, information about Condor Project, ClassAds and JDL (Job Description Language) will be presented. Then, different matchmaking methods and scheduling algorithms will be examined and finally these will be compared with our proposed algorithms.

In Chapter 3 will present information about the simulated model. In the first section of this chapter, we will give the details of the system that we are modeling. Basically, WMS, its components and working principles will be expressed. In the second part, there will be information about GridCAM, which is a simulation tool written by us in C++ that simulates both currently used algorithm and our approaches. The main components of GridCAM will be explained shortly. Most importantly, in this section, we will show how GridCAM simulates the architecture and how its model works.

In Chapter 4, our proposed approaches will be presented. First of all, related and important parts of Workload Management Service will be recalled. Secondly, necessary brief explanation of our study will be made. Finally, important part of our approaches will be expressed and two improved algorithms will be given.

In Chapter 5, there will be experimental test results from real life scenarios by using GridCAM. In the first part of this section, simulation parameter and inputs will be given and explained. Basically, distributions used and their parameters will be expressed. In the second part, test results of current approach will be shown. In the third part, test results of

our approaches will be presented. By comparing the test results of the current algorithm and ours, efficiency of our approach will be shown.

In Chapter 6, the research will be concluded by presenting a summary of work done and its contributions. Then, planned future work and improvements that can be done will be discussed.

2. Previous Work

In this chapter, previous work related to matching and scheduling will be examined. In the first part of this chapter, Condor Project and ClassAds (Classified Advertisements) will be explained. Secondly, JDL (Job Description Language) will be introduced. Then, different matching methods and scheduling algorithms will be examined. Finally, these architectures will be compared with our approaches.

2.1 Introduction to Matchmaking

2.1.1. Condor Project

Condor project employs 30 faculties, full time staff, graduate and undergraduate students working at the University of Wisconsin-Madison. The group has vast experience in distributed computing concepts and practices, systems programming and design, and software engineering [9]. Their group's focus areas and the tools are as follows [10]:

- Harnessing the power of opportunistic and dedicated resources (Condor),
- Job management services for grid applications (Condor-G, DaPSched),
- Fabric management services for grid resources (Condor, Glide-In, NeST),
- Resource discovery, monitoring, and management (ClassAds, Hawkeye),
- Problem-solving environments. (MW, DAGMan),
- Distributed I/O technology (Bypass, PFS, Kangaroo, NeST).

Condor is participating national and international grid research, development, and deployment efforts and especially development and deployment are very important for its success. The projects in which Condor team is very active, are as follows; The Grid Physics Network (GriPhyN)[11], the International Virtual Data Grid Laboratory (iVDGL)[12], the Particle Physics Data Grid (PPDG) [13], the NSF Middleware Initiative (NMI) [14], the TeraGrid [15], the NASA Information Power Grid (IPG) [16], National Computational Science Alliance (NCSA) [17] and Globus Project[18,19]

Despite being a research project, Condor has a significant software production component. Industry, government, and academia use the software routinely in critical settings. So, a part of the project works like a software company. Only the code base of the project contains nearly half-million lines, and significant pieces are tightly coupled to the underlying operating system. There are two versions of the software: a stable version and a development version, and these are developed simultaneously in multiplatform (UNIX and Windows) environment. New functionalities are not permitted before they mature and prove themselves within development series. The aim is to operate each stable version release of Condor in the field before making it available to the public. In order to reach this goal, every release in different development phases runs in different types of configurations.

The Condor project also focuses on the problems of production users. System support for standards in programming environments such as PVM [20], MPI [21], and Java [22] were added for this purpose. Condor project made necessary adaptation for the new protocols like Grid Resource Access and Management (GRAM) [23], and Grid Security Infrastructure (GSI) [24].

2.1.2. Condor and Condor-G

The term “Condor” is usually perceived as a software [9, 39]. The Condor High Throughput Computing (HTC) refers to not only the software development activities but also research group. Condor is a system for HTC whose responsibility is job and resource management system (RMS) [25] for compute-intensive jobs. Condor provides job management mechanism, scheduling policy, priority scheme, resource monitoring, and resource management [26, 27, 28]. When jobs are submitted to Condor, Condor decides when and where to run the jobs by looking at a policy. In addition, it may monitor their progress, and inform the user about their completion.

In addition to batch queuing systems that also provide similar functionality, Condor also performs well in environments in which a traditional RMS is weak in areas such as sustained HTC and opportunistic computing. We can explain the goal of a HTC as a platform providing large amounts of fault-tolerant computational power over long periods of time by effectively utilizing all resources available to the network. Moreover,

opportunistic computing tries to use resources whenever they are available, although they are not entirely available. The goals of HTC and opportunistic computing are naturally coupled.

One enabling mechanism of Condor is ClassAds mechanism that provides an extremely flexible and expressive framework for matchmaking between jobs and machines. ClassAds tell Condor desired conditions, policies, and then Condor creates a plan involving grid resources. Other enabling mechanisms are job checkpoint and migration [29] and remote system calls.

With all enabling mechanisms, Condor manages not only dedicated compute clusters effectively but also manages wasted CPU power. For example, Condor can be configured to run jobs only when the keyboard and CPU are idle. If a user hits a key while a job is running, this job can be migrated to a different workstation by Condor [29]. The job resumes by starting from last checkpoint.

Moreover, Condor supports preemptive-resume scheduling of dedicated compute cluster resources. This also makes it possible for Condor to support priority-based scheduling on clusters.

Production system in UW-Madison Department of Computer Sciences had the first version of Condor in 1987 and nowadays it is used by hundreds of organizations and academia successfully getting together thousands of workstations for each of them.

Condor has a number of universes such as the *Standard,* which is used for check pointing and redirecting file I/O, the *Vanilla,* which is used to run jobs simply, the *Java,* which is used to run java codes and finally the *Globus* universe which is used to run jobs via Globus.

On the other hand, Condor-G [30] can be considered as a union of Globus[18, 19] and the Condor projects, and it is a computation management tool for grids. Globus, which is part of this union, is a set of protocols for secure communications and standard access to

different remote batch systems. Condor part of this union is responsible for job submission, allocation, and error recovery. Figure 2.1 shows this union.

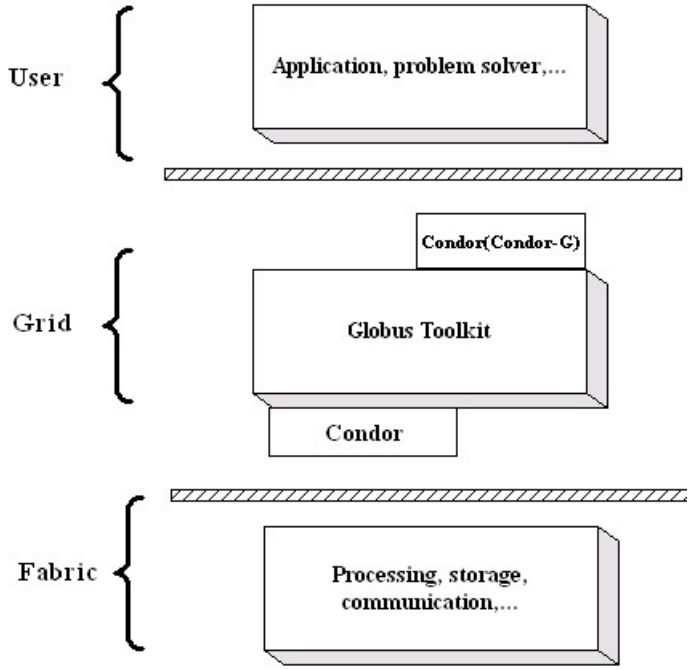


Figure 2.1. Condor and Globus Union [9]

For the job submission and management, Condor-G can be used and the Condor HTC system can be used as the grid generator service. In addition, Globus Toolkit services can be used for coupling HTC and Condor-G. Condor-G is better because Condor is designed to run jobs within a single domain, where as Globus Toolkit runs jobs across many domains. Condor-G gets these two abilities together. One good example of Condor-G usage is the European Union DataGrid [31] project's Grid Resource Broker.

2.1.3. Fundamental Structure of the System

The core components in the structure called the *kernel*, and these components are shown in Figure 2.2 which also describes how core components works.

User submits jobs to an agent and this agent called Workload Management System (WMS). WMS keeps jobs while finding appropriate resources to run them. Meanwhile, agents and resources advertise them in order to inform matchmaker, which is the component of WMS that actually performs matchmaking between jobs and machines.

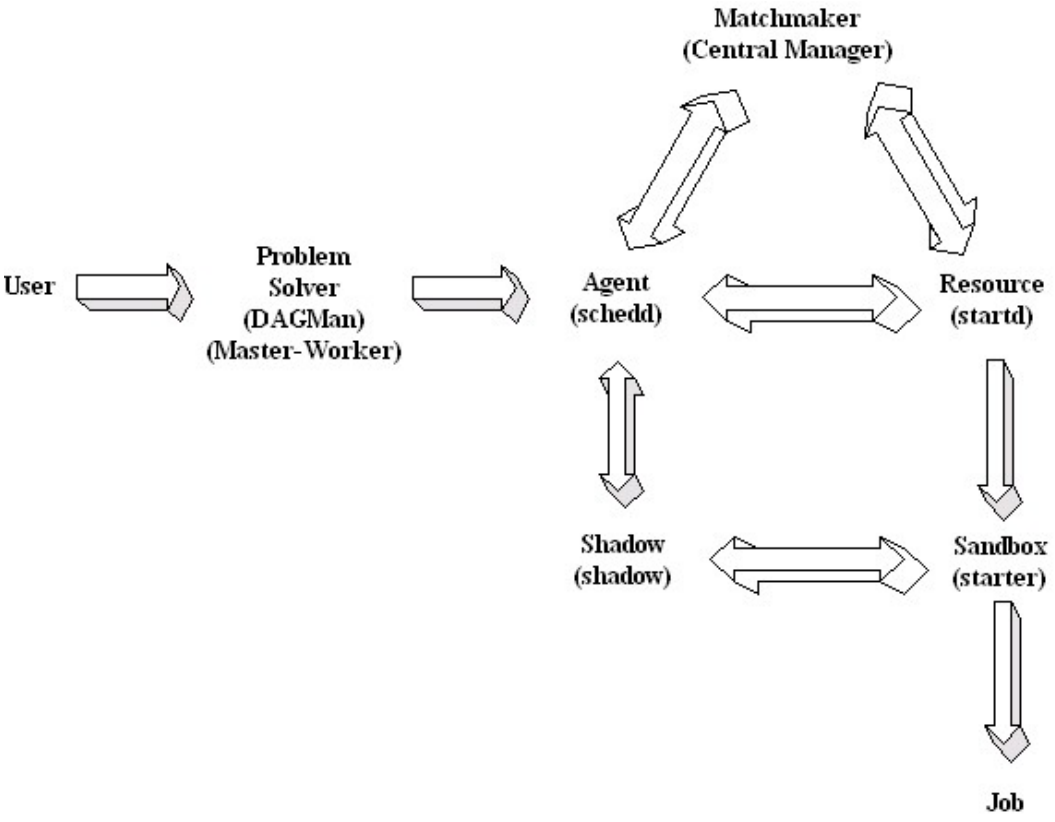


Figure 2.2. The Condor Kernel [9]

Each side creates a new process to execute a job. For agent side, shadow is responsible for executing a job. On the other hand, for resource side, a sandbox is responsible for job execution and protecting resource from any damage.

Working principle of Condor is shown in Figure 2.3. In the figure, “R” indicates Resource, “A” indicates agent, and “M” indicates matchmaker. Basically, the steps are as follows:

- The agent and the resource advertise themselves.
- The matchmaker informs the agent and resource about matching.

- Finally, agent contacts the resource to run the job.

Agents and resources separately create information about themselves and send it to a known matchmaker. This phase is called advertising. Then, this information will be available for the community. Furthermore, an agent and a resource daemon can be run over same machine. Thus, this machine will have the ability to not only submit jobs but also to execute them.

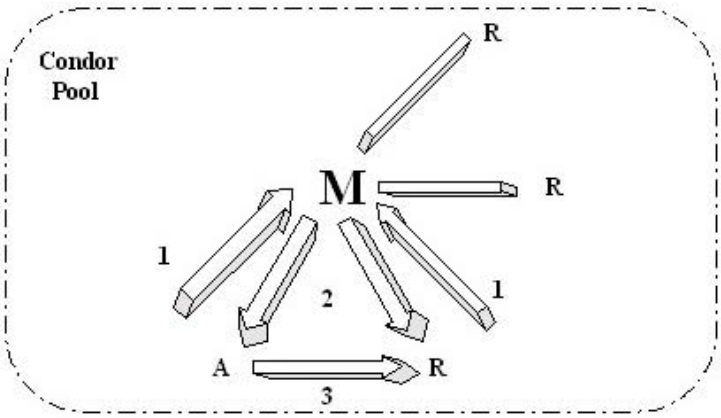


Figure 2.3. a Condor pool. [9]

In addition, agents and resources are logically different, so a single machine may run either or both. More then one instance of an agent may run over a single machine. For example, different users in a machine may run different agents. This is possible because no fixed socket or superuser privileges are necessary.

Agents, resources, and matchmakers are independent and they have their own policies. The agent asks user for policies that include what resources are appropriate and trusted for the job. The resource asks the machine for policies that includes which users are appropriate and trusted for job. Furthermore, matchmaker works as admission controller and it can admit or reject sides based on their definitions and limits. Resources, agents, and matchmakers are autonomous, but the community is a single entity.

Condor has six daemons to realize its operations and these daemons [32] are *master, schedd, shadow, startd, starter, negotiator*, and *collector*.

Master daemon runs over every machine in pool all the time. It creates all other daemons, and monitors each ones conditions. In such situation, if they need, master restarts them.

If a machine submits a job, it has to run *Schedd* and *Shadow* daemons. When a job is submitted, the Shadow monitors it and controls file I/O and remote calls. Schedd represents requests to the pool, and stores the job queue.

Startd runs over the machines which are capable of running jobs. It advertises the machines attributes to the resource that runs Master daemon, in order to be matched. Startd daemon creates the Starter daemon and the Starter creates the execution environment, then runs the job.

Starter daemon communicates with the shadow daemon in order to control the I/O for the job. Each job has its own Starter and shadow daemons and these daemons exist only for the lifetime of the job.

Collector daemon collects information about the Condor pool. All other daemons on all pool machines periodically send their ClassAds to this daemon. Collector basically knows whether the machine is idle, busy, matched or vacating. In other words, it works as Information Server (IS).

Finally, *Negotiator* is the backbone of the Condor system and it is responsible for matchmaking. It asks the Collector periodically for the current status of all resources in the pool. Then, the Negotiator contacts both job and resource Cchedd daemon for waiting job requests, and matches these requests with free resources that are appropriate for them.

As Condor software gets more and more developed, its usage is growing up rapidly all around the world. In the original design, in one community, it is easy to share resource because a participant was working with a single matchmaker to consume or provide resources. Moreover, a user could only participate in one community which is defined by a matchmaker. In other words, users express their needs which are limited by organizational boundaries and these organizations called Virtual Organization (VO).

2.1.4. ClassAds and JDL

Classified Ads, also called Condor ClassAds [33], is a language for Condor system that is used not only describing job requirements but also creating ClassAds for describing resources. In addition, ClasssAds are extensible and widely used. It has *attribute name* and *value pairs*. There are three possible values which are true, false and undefined for attributes. *Requirements* are constraints, and they must be evaluated to true for a match. Furthermore, Rank indicates the desirability of a match. Figure 2.4 shows example ClassAds for a machine and a jobs.

In short, a ClassAd maps attributes to expressions and these expressions can be constants (strings, numbers, etc.), expressions (eg. other.Memory > 200M), lists (eg. { “grida”, “gridb”, “gridc” }) and other ClassAds. It is semi-structured powerful tool for and matchmaking. In addition, JDLs which are created by users are converted to ClassAds in WMS and used for matching with machine ClassAds. Finally, ClassAd library is distributed via GNU Public License (GPL) and complete source code included library code [33].

On the other hand, in grid environment, users make their request by using JDL [34] which is close to natural language and fully extensible language. In this way, users are allowed to use attribute for the description of their jobs. Moreover, only a certain set of attributes are important for us in order to schedule and submit jobs.

Job ClassAd	**Machine ClassAd**
[MyType = ‘‘Job’’ TargetType = ‘‘Machine’’ Requirements = ((other.Arch==‘‘INTEL’’&& other.OpSys==‘‘LINUX’’) && other.Disk > my.DiskUsage) Rank = (Memory ∗ 10000) + KFlops Cmd = ‘‘simulation.exe’’ Department = ‘‘CompSci’’ Owner = ‘‘CMPE’’]	[MyType=“Machine” TargetType=“Job” Machine=“tnt.isi.edu” Requirements= (Load<3000) Rank=dept==self.dept Arch=“Intel” OpSys=“Linux” Disk=600000]

Figure 2.4. Sample ClassAds for both Job and Machine

Since grid users create their job requirements by using JDL, GridCAM also uses and processes JDLs for matchmaking. Fundamental attributes [34] of JDL are as follows;

- *JobType*

  This attribute defines the type of the job and its possible values are *Normal* (simple, sequential job), *Interactive*, *MPICH*, *Checkpointable*
- *Executable*

  This attribute is mandatory and it gives the name of the command.
- *Arguments*

  This attribute is optional and it is used to give arguments to executable
- *StdInput*

  This attribute is optional and it indicates the inputs of a job. In other words, it points what the necessary input files for this job execution are.
- *StdOutput*

  This attribute is optional and it indicates the outputs of a job. It tells which files will be created at end of the job execution.
- *StdError*

  This attribute is optional and it indicates the errors of a job. At end of job execution, this file will tell what was wrong with execution.
- *Environment*

  This attribute is optional and it keeps the list of environment settings.
- *InputSandbox*

  This attribute is optional and it keeps the list of files names located at User Interface machine (UI). These files needed by the job for execution and will be automatically copied to the remote resource.
- *OutputSandbox*

  This attribute is optional and it keeps the list of files names will be created by the job and will be retrieved to UI machine. The listed files will not be automatically copied from WN to the UI. User must run a script in order to do that.
- *VirtualOrganisation*

  This attribute is optional and it is different way to specify the VO of the user.

Basically, in a JDL file, user must specify the name of the executable, the files where to write the standard output and standard error of the job and the arguments to the executable if exists. If it is necessary, name of the files, which will be transferred from UI to WN and vice versa, must be also given in JDL. Figure 2.5 shows a simple JDL.

```
[
Executable = “ls”;
Arguments = “-al”;
StdError = “stderr.log”;
StdOutput = “stdout.log”;
OutputSandbox = {“stderr.log”, “stdout.log”};
]
```

Figure 2.5. Simple JDL

2.2. Related Work Done

Grids [35] are distributed and heterogeneous systems, that unite computational, and information resources connected over high speed networks. The main goal is to create a virtual system using these resources and this system will be available to users. At the center of the grid, there is ability to discover, allocate, and negotiate the use of resources.

Nowadays, many projects make use of grids, but there are still some grid services that are not mature enough such as the scheduling service. Therefore, we concentrate on grid scheduling problem [36]. In other words, we are looking for the best way of assigning jobs to resources that have dynamic characteristics.

The goal of our approaches is to increase throughput besides just finding resources that satisfy users’ jobs requirements. This is called matching [37] and matched pairs are consists of jobs and resources. For our approaches, resources are Computing Elements (CE). Resources and jobs advertise their characteristics and requirements in their ClassAds.

In order to provide detailed information about a matching process, or to support a matching between multiple jobs and resources [38], ClassAds are used. Therefore, this approach also can be considered as constrained-based scheduling. One problem is that we need information about the current states of distributed resources[39] for the matchmaking problem. In other words, we need to know about resources' states such as load of the system and the list of running jobs. ClassAds are well enough in order to get rid of grid monitoring problem. In other words, users do not need to check resources' states manually. Instead of querying states themselves, the users just define what they need for running their jobs in their ClassAds. Globus [19, 23] and the Open Grid Services Architecture [40] are working for us in order to do monitoring, dynamic discovery.

Rajesh Raman [41] mentions that traditional resource management systems use centralized scheduler for resource allocation which is not appropriate to use in distributed systems such as the HTC. Here the problem is heterogeneity of resources, which makes it difficult to make resource allocation, and to handle different allocation policies. Because of these problems, they developed and implemented flexible approach called "the classified advertisement (ClassAd) matchmaking framework". The main goal of resource management systems is to make sure that the assignments of resources to jobs are done efficiently. This problem is called *resource allocation* or *scheduling problem*.

The allocator receives information about states of resources at a time and uses this information to allocate resources to jobs in order to optimize performance metric. This is useful for high performance applications with tight constraints. Efficient scheduling of resources is critical in meeting these constraints. Moreover, the main concern of today's users is the throughput. In a distributed environment, in order to increase throughput, the resource owners may define their usage policies, and these policies may be quite complex. The following exemplifies some policies:

- job can be run if the it is related to aspecific subject
- job can be run between 1 p.m. and 10 a.m.
- job can be run if the load is less than 0.1
- job can be run if the keyboard is idle over ten minutes.

Creating a single unit system model is impossible because of distributed ownership. Therefore, a resource management system which does not need such a model and also it may work at an environment such that resource and job owners can create their own models dynamically. Solution is the matchmaking resource management paradigm which uses semi-structured data model also called ClassAd data model.

R. Raman [41] found that the matchmaking works fine in an environment that includes heterogeneous resources like workstations, tape drives, and network links. States of these resources may change without notice. In this situation, *opportunistic scheduling* which can be defined as using resources as soon as they are available and task migration when resources need to be preempted. If the main concern is high throughput rather than high performance opportunistic scheduling is a good solution.

According to Raman [41], idea behind matchmaking is as follows; entities (jobs, resources) advertise their characteristics and requirements in their ClassAds [33]. Matchmaking service, also called matchmaker, matches ClassAds in a manner that makes sure that both sides' requirements are satisfied and then informs the sides about matching. The matchmaker stops working at this point, and the matched jobs and resources establish connection to perform desired task.Raman believes that this framework is made up of five components and these components are as follows:

1. ClassAd specification.
2. The advertising protocol which defines rules related to matchmaker such as ClassAd content the way matchmaker receives ClassAd from the advertiser.
3. The matchmaking algorithm, which matches the contents of the ClassAds.
4. The matchmaking protocol, which is interested in information given "when did entities match?" and "how matched tasks and resources are notified?".
5. The claiming protocol, which contains set of actions for completing desired tasks after matching is done.

Raman also mentions the difference between their approach and the conventional resource allocation models. First of all, the conventional way supports only task related constraints but Raman's mechanism also allows service related constraints that affect tasks.

In addition, the matchmaker just creates a match between entities such as job *j* and resource *r*. After matching, these two entities run claiming protocols. Raman tells that advantage of this approach is that there are no states of the matchmaker and this makes it easier to recovery of failures.

Resources and jobs create their ClassAds which describes themselves and send them to the matchmaker which is shown as Step 1 in Figure 2.6. These ClassAds must be appropriate for the advertising protocol that defines how the entities send their ClassAds to the matchmaker. Then, matchmaking algorithm runs and to do that the matchmaker evaluates expressions in each ClassAd to access attributes of the other. This phase is shown as Step 2 in Figure 2.6. Furthermore, in a ClassAd, there can be references to ClassAd itself and other ClassAds.

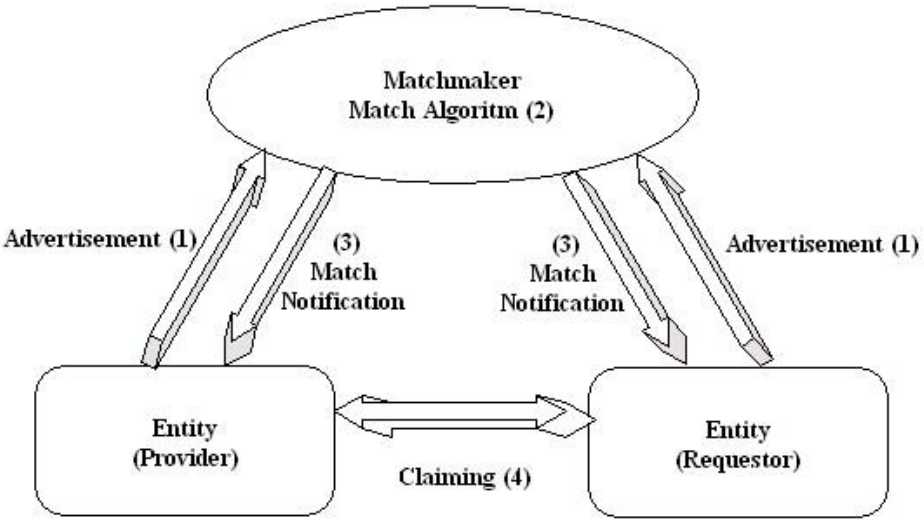


Figure 2.6. Matchmaking Process [41]

The ClassAds that submitted to matchmaking algorithm are incompatible unless their constraint expressions are evaluated to *true*. If there is a reference to a non-existent attribute, this is evaluated as *undefined* or *false*. If an attribute does not exists both in job and resource ClassAds, then the expression will be evaluated to undefined. At the end, matching process, matchmaker may give more than one result then decision is made by

looking at the *rank* attributes. The matchmaker selects the one with the highest rank value. In addition, if there are entities with same rank, random selection is done.

After matchmaking algorithm executed, the matchmaker uses matchmaking protocol to inform both sides about matching and sends matching ads to them. This phase is shown as Step 3 in Figure 2.6. The matchmaking protocol could also include the generation and hand-off of a session key for authentication and security purposes. Finally, Step 4 in Figure 2.6, both sides establishes communication using claiming protocol. Here the important point is that successful matchmaking does not mean job will get the matched resource. Result is clear when Step 4 is done. In addition, different approaches for matching and claiming has advantages such as weak consistency requirements, authentication, bilateral specialization and end-to-end verification [42].

In grid, matchmaking algorithm works over Resource Broker (RB), so RB finds the set of the suitable CEs for jobs. Then it finds the appropriate resource for a job by looking at rank attributes[43]. In the ranking phase the RB contacts directly the Lightweight Directory Access Protocol (LDAP) server and determines rank expression of the received JDL. Figure 2.7 shows this process.

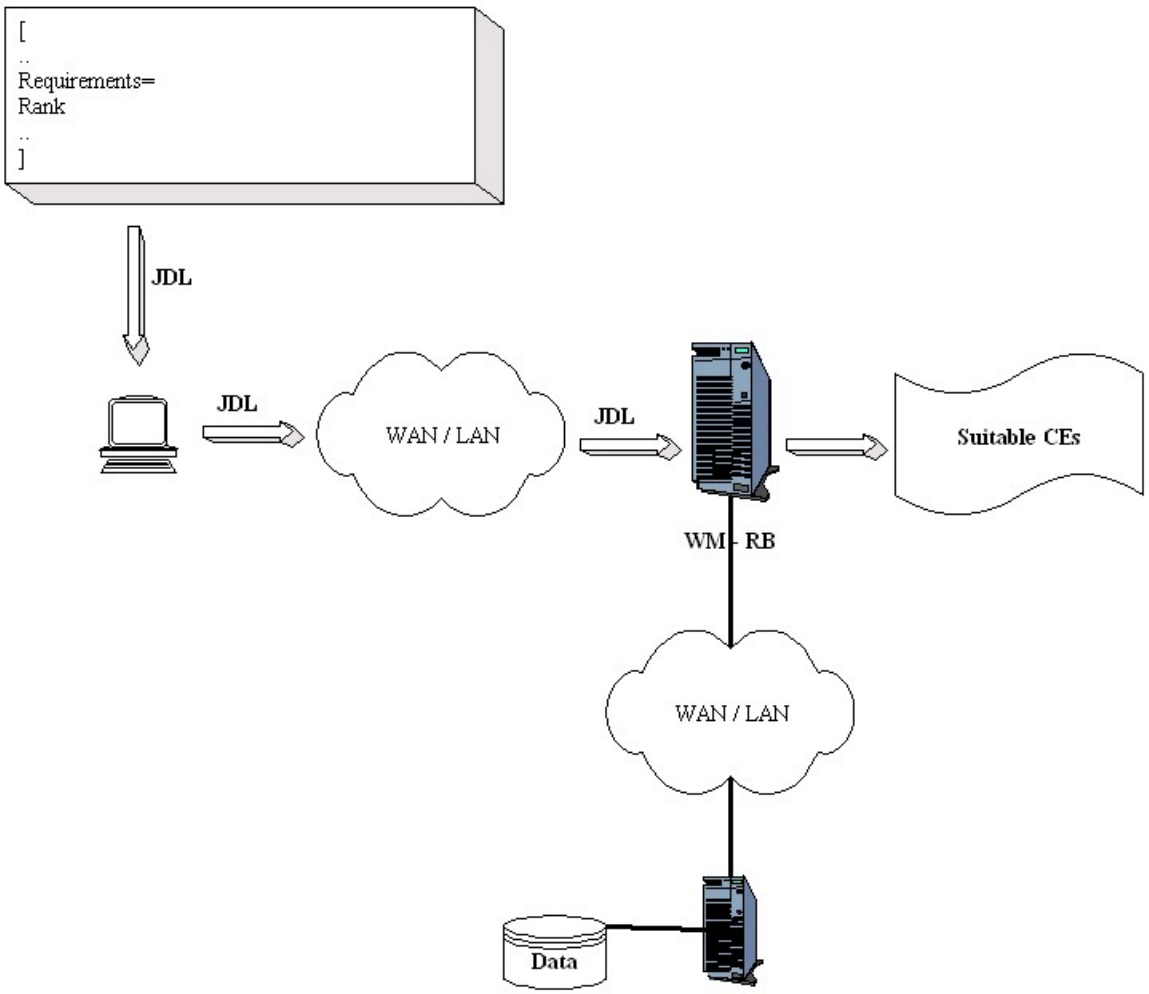


Figure 2.7. Matchmaking and Ranking

E. Imamagic [44] states that Condor-G Matchmaking mechanism is a good solution for grid scheduling, and he proposes a new approach by using this mechanism. Grid middlewares (GMW) [44], which realize integration of the resources, are the union of services and protocols. In addition, one of the features of GMW is grid scheduling also called super scheduling, metascheduling or grid brokering.

Imamagic[44] adds that grid scheduling has important issues because of grid environment and he focuses on the most important issues among them. First of all, a grid scheduler should support various job types such as serial (single processor) and parallel (more than one processor) jobs. In addition, a job management should support

checkpointing, preemption, job migration, rescheduling jobs, fault tolerance, and advance reservation of resources, which is reservation of resources for a specific period of time. Job and resource priorities may be assigned by scheduler and not only resource should have preferences but also jobs should be able to request specific features like hardware capabilities and rank the resources. In addition, users should be able to design custom scheduling algorithms. In other words, conventional algorithms must be supported besides supporting custom algorithm development. Advance reservations should be also supported. Moreover, scheduler should consider data location and replication for data dependent jobs. This approach called data-aware scheduling. Finally, it is clear that system scalability is important because of grid environment. Therefore, scheduler should also take into account load balancing. Most of these features are getting together in Condor-G Matchmaking [44, 45] and Imamagic uses Condor-G for its approach.

Central side of Imamagic's system consists of three components which include Grid Information Service (GIS) Adapter, Publisher and Condor-G Matchmakers. Figure 2.8 demonstrate this architecture.

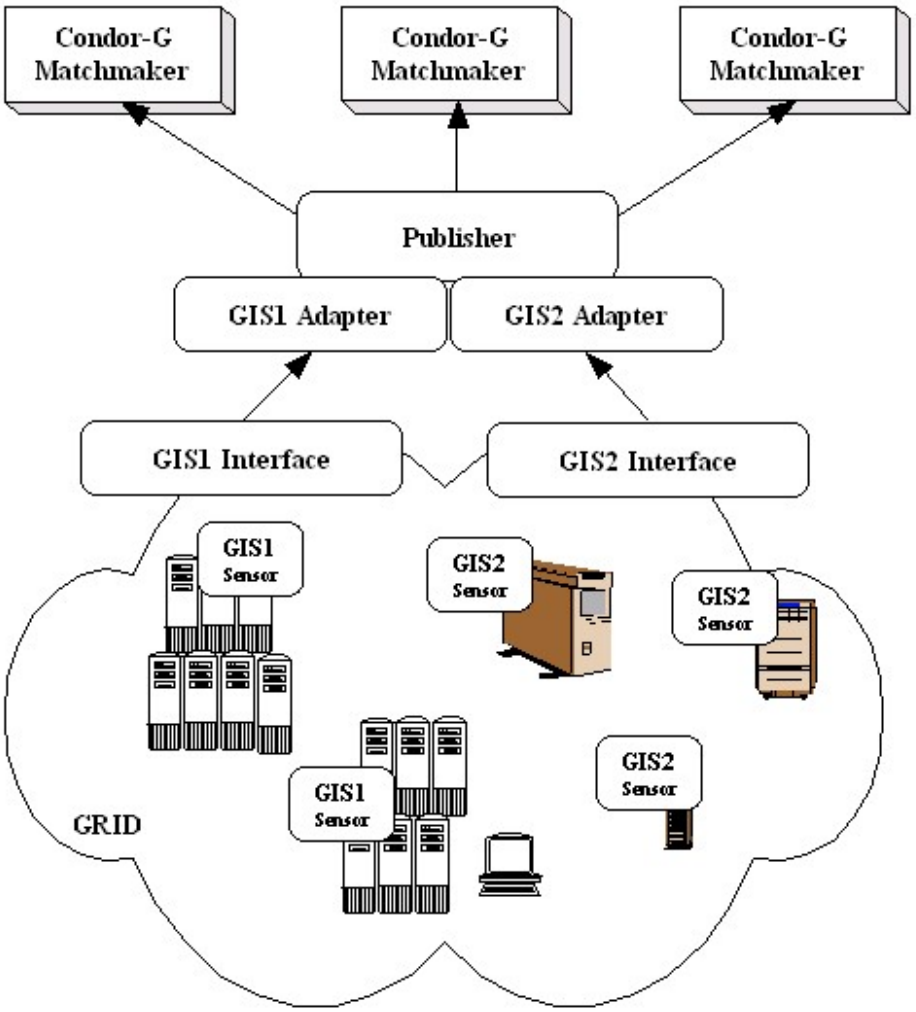


Figure 2.8. Condor-G Matchmaking [45]

Publisher component provides information to Condor-G matchmakers. GIS adapters let publisher to collect information from particular type of GIS. In addition, the most important contribution of Imamagic is the modifications over Condor-G Matchmaking. By this modification, it is possible to use multiple Condor-G Matchmakers, so greater scalability is achieved. Moreover, by focusing on GIS, there is no need for resource adapters, sensor management and information management. In other words, by this approach, there is no need to collect information from each resource type.

Heinz Stockinger [46] focuses on grid enabled physics analysis and for this reason they focus on WMS which is used to find suitable CEs to execute data intensive physics related

jobs. A dataset name indicates a certain physics event uniquely and this event includes several particles. The main idea behind their approach is that physicists will be able reach all the particles by using one dataset. Stockinger modifies WMS, particularly matchmaker, to allow physicists to express their job requirements in terms of datasets. This makes it necessary to modify data catalogues also. As a result, they come up with a simple "Data Location Interface" that make it possible for WMS, new dataset and file catalogues to work together.

They have made several changes and enhancements to the conventional WMS which will be described in Chapter 3, in order to use datasets in the physics analysis process. Basically, the WMS needs to allow for datasets in the JDL as attributes. In addition, the Matchmaker needs to interact with a dataset catalogue. They define a term called Logical Dataset (LDS) which points to an entity of data. It works like Logical File Names (LFNs) and Global Unique Identifiers (GUIDs) where both methods are used to show location of a file. The difference of LDS is that it can be considered as a file collection instead of a single file. Here the assumption of them is that physical files in a dataset are stored at the same SE and can be reached via the same protocol. Finally, by creating their own UI they make it possible to matchmaker work with LDS. Heinz Stockinger [46] approach can be classified as data-aware scheduling.

According to Nicholas Coleman [47], resource management for distributed systems is difficult and centralized allocators do not satisfy the grid environment. Matchmaking using ClassAds works fine in such situation, but due to the distributed polices and dynamics environments it is difficult to understand why some ClassAds are matched while others are not. [47] describes problematic sides of policies and offer some modifications in this manner. However, there is more interesting study in this paper which is about detecting conflict in ClassAds. It is possible to create a ClassAd that contains predicates which conflict with each other. These predicates may be satisfied side on their own, but are not satisfied on the other side. When one of them evaluates to false, there will be no matching. For example a ClassAd, which have two separate statements like other.Arch= "Intel" and other.Arch= "AMD", will not be matched.

Detecting these conflicts requires the evaluation of the individual expressions in the context of ClassAds. First, Coleman separates the predicates in a clause by attribute reference. Then for each attribute he converts predicates to points or intervals depending on the type of the values. If the intersection of these intervals is empty then there is a conflict. For this purpose, Minimal Failing Subexpression (MFS), and Maximal Succeeding Subexpression (MSS) sets are created [48]. Here the problem shown by Godfrey [48] is that finding all MFSs is NP-Hard, but Godfrey proposes a linear time algorithm to find one MFS and a polynomial time algorithm to find fixed number *k* MFSs.

Hongsuda T. [49] starts by defining the resource matching problem for the grid in a manner assigning resources to tasks in order to satisfy both sides. This is done by using requirements and policies which are expressed in as models. Thus, resource selector performs matching between task and resources. In [49], they propose a flexible and extensible approach for solving resource matching problem in the grid via semantic web technologies. Furthermore, they use an ontology-based resource selector that uses ontologies, and rules for matching.

Hongsuda mentions that [49] existing resource management systems for the grid are highly unnatural. Traditional resource matching like Condor Matchmaker [41] or Portable Batch System [50] does attribute based matching. Systems like these which perform matching between tasks and resources are not flexible and are difficult to improve. This is because it is difficult to ensure the syntax, semantics of resource and task descriptions. Hongsuda [49] proposes a flexible and extensible approach for matchmaking in grid. This approach is ontology based matchmaker which describes resource request properties based on symmetric flat attributes. Here the problem is the number of attributes grows too much, and hence handling will be difficult. Separate ontologies are created in order to describe resources and job requests using an expressive ontology language. In this approach, instead of exact syntax matching, their ontology based matchmaker performs semantic matching using predefined terms in ontologies. In addition, this approach can be easily improved by adding new terms, and rules to ontologies. Finally, Hongsuda created a matchmaker using existing semantic web technologies to use ontologies and rules for resource matching.

Rajesh Raman [51] starts by examining matchmaking which is a well formed and powerful resource management solution for grids. Although it has its advantages, matchmaking does not support multiparty policies like co-allocation. In [51], Raman proposes gangmatching which is a multilateral matchmaking.

The gangmatching modifies ordinary ClassAd's single implicit bilateral match rule with an explicit list of required bilateral matches. The ClassAd representing the gang-match request for the Job-Workstation-License example is illustrated in Figure 2.9

```
[  Type = “Job”
Owner = “CMPE”
Cmd= “simulation”
Ports = {
  [   Label = “cpu”
      ImageSize = 30M
      Rank = cpu.Kflops+cpu.Memory
      Constraint = “cpu.Type = = “Machine” && cpu.Arch = = “Intel” &&
       cpu.OpSys == “Linux”   ]
 [    Label = “license”
      Host = cpu.Name
      Constraint = “license.Type = = “License” && license.App = = Cmd    ]
        }
]
```

Figure 2.9. Gangmatching Example [51]

The ports attribute, which presents the required matches for job, indicates a list to satisfy the job. As it mentioned gangmatching is multilateral matching and this is realized with bilateral matching between ports of ClassAds.

In short, the idea is to find a compatible match amongst a gang of ClassAds where each of them contains a list of ports specifying requirements for matching ClassAds. For this purpose a recursive backtracking search is done by starting at a root ClassAd. Algorithm proceeds sequentially to satisfy ClassAd ports by finding compatible ClassAds. If a

matching ClassAd has unbound ports, it is considered as a root and recursively satisfies its ports. If no match is found, algorithm goes back to previous port. If all ports of the root are fulfilled, it returns with a valid solution. Furthermore, if the algorithm tries to get back from first port, it returns with no solution. Finally, when optimization is the issue; using indexing scheme to exclude incompatible ClassAds or heuristic to satisfies ports in ascending order of candidate set size can be used.

As described before, the Matchmaking framework [41] creates a link between a job and appropriate resource. Raman et al. [37] and Liu et al. [38] extend matchmaking in order to map a task to multiple resources and to support resource co-allocation, respectively, and they propose a number of heuristic algorithms to solve the mapping problem.

Raman et al [37] gives information about matchmaking and then continues with its bilateral matching disadvantages. In other words, it allows matching between a job and a single resource which means that it does not allow co-allocation. In this paper, they propose multilateral extended matchmaking model, also called gangmatching.

According to [37] matchmaking's most important disability is that it supports matching between single resource and job. Sometimes, it is necessary to use collection of resources in order to run a job. Another problem is that a Condor user has licenses for different software packages and jobs. In addition, some licenses may be valid only on some workstations, while others may be valid on certain networks. Because of this, it is logical to treat software licenses as resources. Gangmatching handles these at once. The matchmaking is not able to handle dependencies between jobs, resources and licenses. However, an approach with two steps can be used. The idea is that first, a match between a job and a resource is made. Then, this job and resource match is advertised to find an appropriate license. In the worst case scenario, a deadlock could rise if first and second matchmaker waits for the same resource.

Liu et al. [38] also proposes matchmaking framework like Gang-Matching. They propose a resource selection framework that solves matchmaking related problem described in [37] for grid environment. For this framework set matching is used, and this

framework extends the Condor matchmaking framework in order to not only support single job single resource model but also multiple resources for a job.

Liu also defines set extended ClassAds language [38] that allows users to request different resource properties like total memory, and minimum bandwidth. Set matching matchmaking algorithm, which supports set extended ClassAds language, is proposed also. This framework supports one job and many resource matching. Set extended ClassAd is identical to conventional ClassAd except that it can include both set expressions, and individual expressions. In this framework, not only application and resource requirements are defined but application performance models can be defined. According to these criteria, resource selector finds suitable set of resources.

According to set extended ClassAds [38], a successful match can be defined as relation between a single set request and a resource set. Set extended ClassAds are used to express set requests and individual expressions such as memory size, which have to be applied each resource in the set individually. The set-matching algorithm attempts to construct a resource set that satisfies both individual and set constraints. A set of resources is returned when the set extended matchmaking is successful. In addition, set extended ClassAd language extends conventional ClassAds are as follows; First, in ClassAds, a new type is created which is “Set” (e.g. Type= “Set”) this is used to tell matchmaker, this ClassAd is set extended ClassAds. Secondly, three aggregation functions, which are Max, Min, and Sum, can be used in set extended ClassAds. Their usage is like *aggregation_function(expression)*. For aggregation function max, it returns the highest value when the expression is applied to set of ClassAds. Min works like Max, but it returns lowest value. Finally, Sum returns the sum of the values returned by expression. Third extension is a boolean function *Postfix(V, L)* and it is defined such that returns true when one of the member of list L is the postfix of value V. For example, Postfix(“yildirim.cmpe.grid.boun.edu.tr”, {“boun.edu.tr”, “grid.boun.edu.tr”}) will return true. Finally, function SetSize that refers number of elements within the current resource set.

Set Matchmaking works with set-extended ClassAds. It performs matching between request and set of resource ClassAds then returns the resource set with highest rank. This

process is done in two steps. In the first step, which is called the filtering phase, some individual resources are filtered by looking at individual expressions in the request. For example, consider the individual expression like "other.arch == intel && other.memory>=10M". For this sample expression, resources with non-intel architecture and memory with less than 10 MB will be removed from consideration list. For filtering purpose, Set Matching algorithm uses index to obtain high performance. In the second step, also called set construction phase, Set Matching algorithm try to locate a resource set that best fits job requests. Here the problem is, result set may be too large, and it's not feasible to evaluate all possible combinations. Instead of doing that, paper proposes to use a greedy heuristic algorithm after first step to create a resource set. Figure 2.10 gives this algorithm.

Basically, a candidate pool is created and the resource with the highest rank is selected. Then, this resource is added to candidate pool. Algorithm repeats this. If the candidate set has higher rank than the best set created, then the candidate set become the new best set. This is repeated until resource pool is empty. Finally, the best resource set is returned as result. Algorithm returns a failure if no resource is found.

Liu et al. [38] mentions that the used greedy algorithms for Set Matching does not guaranty finding the best solution if it exists. The set-matching problem is NP-hard under some circumstances. In other words, it is difficult to find a general algorithm to solve this problem efficiently, especially for large resource set. However, in this paper proposed framework has an efficient heuristic algorithm with time complexity $O(N^2)$.

```
CandidateSet = NULL;
BestSet=NULL;
LastRank = -1; Rank = 0;
while (ResourceSet != NULL)
{
Next = {X : X in ResourceSet && for all Y in ResourceSet,
rank(X+CandidateSet) > rank(Y+CandidateSet); }
ResourceSet = ResourceSet - Next;
CandidateSet = CandidateSet + Next;
Rank = rank(CandidateSet);
If (requirements(CandidateSet)==true && Rank > LastRank)
BestSet=CandidateSet;
LastRank=Rank;
}
if BestSet ==NULL return failure
else return BestSet
```

Figure 2.10. Greedy heuristic algorithm [38]

C. Liu [52] proposes a new approach to symmetric matching that brings important advantages when compared to conventional ClassAds. Important point in their research is that they consider matching as a constraint satisfaction problem (CSP) and they try to solve by using constraint-solving technologies.

Liu describes a new matching system with new features which can not be realized by conventional ClassAds. First feature is that they describe resources with different levels of generality in order to make their system more flexible. Normally, ClassAds perform only exact matches via examining attributes. However, in this approach descriptions with varying levels of generality and complexity can be defined. Secondly, in this approach matching advertisements are based on not only properties but policies. Policies are important resource selection criterion and in this approach policy queries can be used in

expressions. For example, a user may ask: “Find all machines that allow access between 7:00 PM and 9:00 PM.”. Finally, this approach supports one to many matching, also called multilateral matchmaking, like Gang-Matching [37, 38]. They have created a prototype system for experiments and it is called RedLine.

Alain Roy [53] focuses on preemptive resume scheduling. Condor is a batch job system that works with not only dedicated computers but also non-dedicated computers like desktop computers. Roy does not see these non-dedicated computers as a problem. They call them as “opportunistic resources”. Usage of these non-dedicated computers creates extra computing power, but this approach brings an additional responsibility to Condor which is called preemption. In other words, Condor will checkpoint and preempt jobs if the resource that the job runs is unavailable and move this job to available computer. Here the important point is that job can be resumed on the available computer without loss of work because checkpointing saves the state of running job. This can be done easily by Condor libraries. Jobs that require checkpointing should re-linked with libraries provided by Condor. Running a job in a Condor pool is a cooperative process between a job, a machine, and the matchmaker as described in [41]. Therefore, when preemption is considered, each of these parties is allowed to preempt a running job in order to satisfy their requirements. Another issue is when checkpointing will be done. Condor can checkpoint jobs when they are preempted from a machine, or user can request checkpoints manually, or , Condor can do checkpoint jobs periodically to avoid future failures.

Sinaga [54] also focuses on co-allocation like [37, 38, 52] does. In Sinaga, it is mentioned that existing co-allocation mechanisms DUROC component of the Globus Toolkit, does not provide resource-brokering or fault tolerance functionalities. In [54], these two features are coupled in the form of a software layer on the top of DUROC and propose as a new system.

The job management can be expressed as a set of operations that is used to control how capabilities provided by grid resources are made available to users. In other words, resource management does not focus on core client related functions of a resource, but rather it focuses on when a requested operation starts or how long it takes to complete.

In addition, standard functionalities of job management systems are as follows: checkpointing, preemption, job migration, fault-tolerance and rescheduling. Existing management systems, which have these features, are as follows; Condor, Nimrod/G and Gridbus Broker [55, 56 57], AppLeS Parameter Sweep Template [58], GridWay [59], CSF [60], and Moab/Silver [61].

Effective scheduling over the distributed resources is mandatory in order to overcome overheads in the grid. In addition, a Directed Acyclic Graph (DAG) based scheduling algorithm can be very effective for improving the performance because it reduces redundant transmission of input and output data from execution of related client requests.

In [62], the main idea is the study of the architecture specification of the Workload Management System. In particular they focus on how resource co-allocation and job partitioning are addressed. It is stated that job partitioning requires mechanisms to address the problem of job dependencies. It is further stated that it is possible to define dependencies on a set of program executions, building a Directed Acyclic Graph (DAG), whose nodes are jobs, and whose arcs represent dependencies between them. They use DAGMan for the WMS, in order to manage a DAG schedule. Here, the main purpose of the DAGMan is to determine which nodes are free of dependencies, and monitor the execution of the corresponding jobs. The owner of documents believes that minimizing the dependencies between the different components requires that the simplification of the flow of control within the Workload Management System.

Condor Project is used to develop, implement, deploy, and evaluate mechanisms and policies that support HTC [28] on large collections of distributive computing resources. It can be used by scientists and engineers in order to increase their computational throughput. DAGMan (Directed Acyclic Graph Manager) is a meta-scheduler for Condor which manages dependencies between jobs. DAG can be used to represent a set of jobs where the input, output, or execution of one or more jobs is dependent on other jobs. Here, the jobs will be denoted by nodes, and the edges will indicate dependencies of these nodes. DAGMan submits jobs to Condor. There is an input file which describes the DAG. Each node in the DAG needs its own Condor submit description file. As DAGMan submits jobs

to Condor, it uses a single Condor log file to enforce the ordering required for the DAG. The DAG itself is defined by the contents of a DAGMan input file. In addition, DAGMan is responsible for scheduling, recovery, and reporting for the set of programs submitted to Condor.

[63] focuses on general the issues involved in mapping complex workflows onto the grid. An abstract workflow is the order of the execution of the jobs. Since the workflow is determined, next step is that to map this workflow onto the available grid resources, performing resource discovery and selection. Paper gives Condor-G/DAGMan [32] as an example.

There are two main approaches: First one lets the planner make an exact plan of computation based on the current information about the system. The planner decides where the tasks need to execute, the exact location of input. This is called full-plan-ahead. In the second approach, on the other hand, planner allows executer make many decisions such as compute platforms to use.

The benefit of the first approach is that planner can optimize the scheduling by using entire structure of the DAG. Since execution environment is very dynamic, when a task in the DAG is ready to execute, the environment might have changed so that necessary data may no longer be available at the location assumed by the planner (execution-time error). Quickly re-plan may be a solution to this problem. On the other hand, using second approach causes more trouble then the first one. It is obvious that user can find out about the state of the resources and the location of the data and make a locally optimal decision. However, it could make potentially expensive decisions, because the users do not have global information. Another approach is deferred scheduling, where the user and the planner work together to come up with a plan. The planner provides an abstract workflow for jobs and ask user for confirmation. In this way, user would reach the global information because plan will be created by using the most up-to-date information. Here the drawbacks are high communication overheads and a large amount of computation because of re-planning.

[64] on the other hand focuses on efficient scheduling of requests in order to use grid resources that must adapt to dynamic environments. Jank-uk [64] describes a framework called SPHINX that can administer grid policies, and schedule complex and data intensive scientific applications. They show that SPHINX can effectively schedule work across the large number of distributed clusters. Main component of the system is SPHINX server. Their idea is as follows; First of all, the server determines which way is the best way to allocate resources in order to complete the jobs. Second, it maintains catalogs of data, executables and their replicas. Then, it determines completions time of the requests that requires these resources. Finally, the server monitors the status of its resources. SPHINX adapts finite automaton for scheduling status management. The scheduler moves a DAG through predefined states to complete resource allocation to the jobs in the DAG. The server has a control process, which completes the scheduling by managing several SPHINX inner service modules such as resource monitoring interface, DAG reducing and planning. The DAG reducer reads an incoming DAG, and eliminates previously completed jobs in the DAG. Then, it determines whether the output files of each job exists, and if they all exist, the job and all precedence of the job can be deleted.

[65] is about the ATLAS experiment which is a large, multipurpose experiment designed to be sensitive to a broad range of phenomena that are expected to appear at the energies of the Large Hadron Collider at CERN. As part of the studies on this experiment, a prototypical resource broker for have been developed for testing some of the scheduling concepts by Atlas group. According to this, there is a database (MySQL) that keeps predictors of CPU execution time and data transfer time for jobs. The jobs in the grid environment, which use the GLOBUS toolkit, have production files that are called DAGs. DAGs manage the job input and output. The DAGs that are input to the job execution can be created via a package called “Chimera”. Database keeps the information about job execution parameters and by using these parameters and available resources, the scripts that run the resource broker are used to modify the DAGs to optimize the job execution. This idea is mentioned as “virtual data” in grid context by ATLAS group.

2.3. Summary

The main purpose of grid computing is to let users run their jobs upon a pool of shared resources. Since there is rarely a one to one correspondence between the jobs and resources, a scheduling model is necessary. This model will know about both jobs and resources and will make decisions with respect to requirements. Our research is based on this idea, and we are looking for the best way to allocate the resources in order to increase throughput of the system. For this purpose, many approaches investigated and we found out that matchmaking area needs further contribution.

First of all, we addressed the adaptation of whole matchmaking structure that was explained in [41]. Like [44], our model will support various job types. Basic types are serial and parallel jobs. Serial job is a job that needs single processor for execution. On the other hand, parallel job requires more then one processors for execution. In addition, a job management should support checkpointing, preemption, job migration, rescheduling jobs, fault tolerance, advance reservation of resources, which is about the reservation of resources for a specific period of time. We are not interested in these features because our focus on this thesis is just matchmaking problem.

In our model, job and resource priorities are assigned in order to prevent the starvation problem. According to researchers, custom scheduling algorithms should be supported but this can not be applied for us since we try to develop a new scheduling algorithms. In addition, (e.g. jobs that waited for long time) advance reservations mechanism may be used.

Moreover, data-aware scheduling can be done at some level by using requirements in ClassAds. In this way, jobs that require large data will be assigned to resources closer to data in order to avoid heavy network traffic. However, our approach will not examine characteristics of network links like data-aware scheduling does.

Another approach presented in [46] focuses on physics related jobs. They modify WMS, particularly matchmaker, to allow physicists to express their job requirements in terms of datasets. They also created their own modified user interface. WMS is modified

in order to use datasets in the physics analysis process. Basically, the WMS needs to allow for datasets in the JDL as attributes. In addition, the Matchmaker needs to interact with a dataset catalogue. However, this approach is only helpful if we need ClassAd or WMS modifications because performance is not main concern of this paper but we are.

Information obtained from [47] is important because they try to detect conflict in ClassAds. This is helpful for our research because it is possible to create a ClassAd that contains predicates which conflict with each other. Paper proposes a solution for this problem. This model was not adopted, and it is given as future work.

Hongsuda T. [49] proposes a flexible and extensible approach for solving resource matching problem in the grid via semantic web technologies. According to [49] existing resource management systems for the grid are highly unnatural. Traditional resource matching like Condor Matchmaker [41] does attribute based matching which is not flexible and is difficult to improve. They create ontologies and corresponding attributes. Here the problem is the number of attributes grows too much thus handling will be difficult. In this approach, instead of exact syntax matching, their ontology based matchmaker performs semantic matching using predefined terms in ontologies. This can be also considered as future work because it focuses on ClassAd modifications rather than matchmaking

Rajesh Raman [51] proposes a model that supports co-allocation, and it's called Gang-Matching. Difference between Matchmaking and Gang-Matching is that Gang-Matching does multilateral matchmaking. Liu et al. [38] and Raman et al. [37] also extend Matchmaking in order to mapping a task to multiple resources and to support resource co-allocation. All these similar approaches and our approach will be based on conventional Matchmaking which is also used by TR-Grid.

In the next related approach, Alain Roy [53] focuses on preemptive resume scheduling and examines how Condor does this. Preemptive resume scheduling is necessary especially for non-dedicated computers. There are no non-dedicated computers in TR-Grid, therefore this is not the main concern of our approach.

Finally, in [62, 64] DAG and DAGMan problem solver is offered in order to improve scheduling performance. DAGMan can also be used in our approach when we re-order the job queue. It can submit jobs in new order into Condor queue if necessary.

3. SIMULATED MODEL and GridCAM

3.1 Introduction

In this chapter, simulated grid model will be explained. In the second part of this chapter, we will give the details of the system that we are modeling. Basically, WMS, its components and working principles will be expressed. In the next section, there will be brief information about GridCAM, which is a simulation tool developed by us in C++. GridCAM simulates both Condor's matchmaking algorithm and our proposed algorithms. Finally, the main components of GridCAM will be explained briefly.

3.2 Simulated Model

The goal of grid systems and its applications is to integrate, virtualize and manage distributed resources and services across different VOs that are defined as individuals or institutions having direct access to resources. Many VOs need sharing of resources through services like accessing, allocating, monitoring, and accounting and these are realized by using grid middleware which can be defined as a layer between services and physical resources. On the TR-Grid platform that we are modeling, gLite [66, 67] middleware is used.

Users may interact with CEs and may find an appropriate resource for job submission. On the other hand, Workload Management System (WMS) [68, 69] can do that for us. Basically, its goal is to submit a given job to an appropriate CE found by a matchmaking process which was described in detail in Chapter 2. Note that CE must not only advertise its availability but also collect user accounting information.

In addition, there are two job submission (Task Queue) models according to user requests and site policies

- *PUSH (Eager Scheduling):* In this model, job is pushed to CE without checking to see whether CE is available.

- *PULL (Lazy Scheduling):* In this model, job is submitted by WMS when CE is available

PULL model is used in TR-Grid and therefore, we also use it in our GridCAM simulation tool.

WMS [69] can be considered as set of middleware components which are responsible for distribution and management of jobs across grid resources. There are two main components of WMS. Figure 3.1 shows the components of WMS.

- *Workload Manager (WM):* This is the component that accepts and satisfies requests for jobs. The matchmaking is done in this component.
- *Logging and Bookkeeping (L&B):* This component keeps track of job execution in terms of events such as submitted, running, and done.

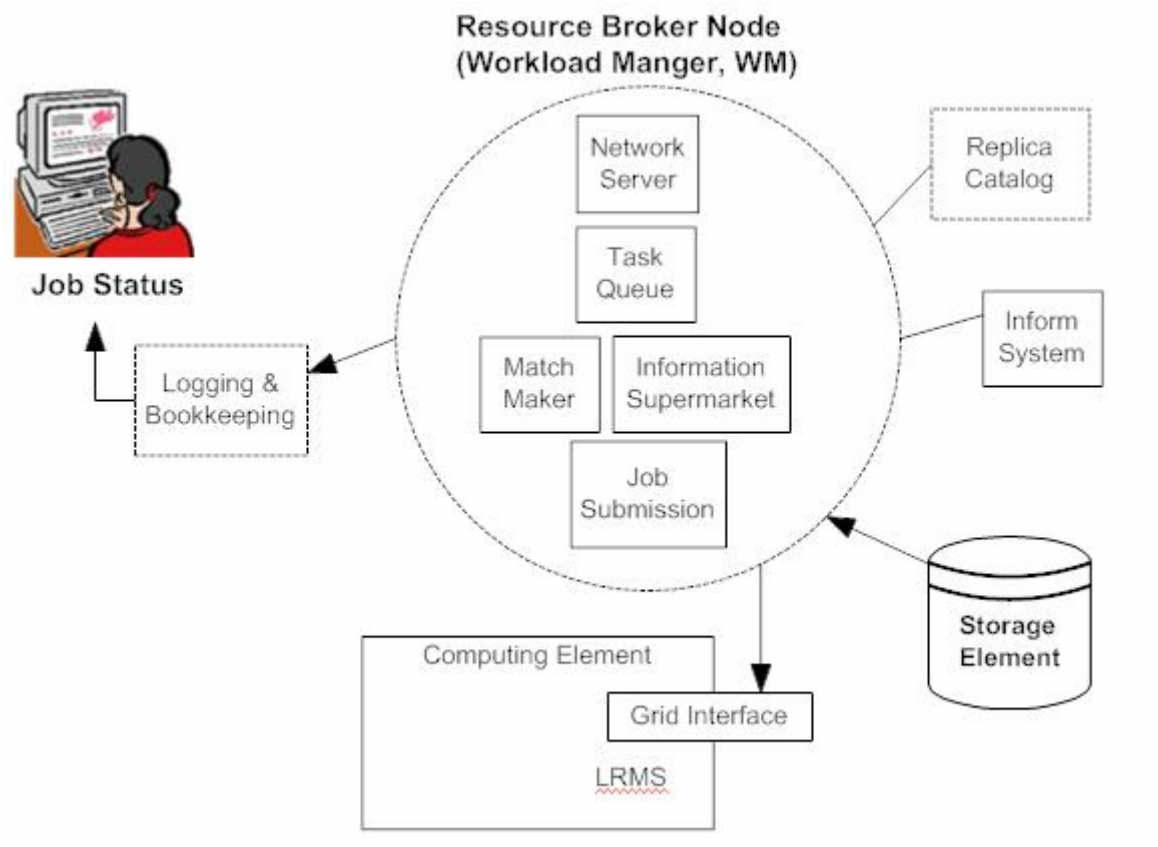


Figure 3.1. WMS Architecture [68]

As stated earlier, WMS is a union grid middleware components responsible for distribution and management of tasks over grid resources. It accepts jobs via WM and it passes the job to an appropriate CE for execution by considering job and resource requirements in JDL. Appropriate CE is chosen by performing matchmaking. Figure 3.2 shows that how these phases work.

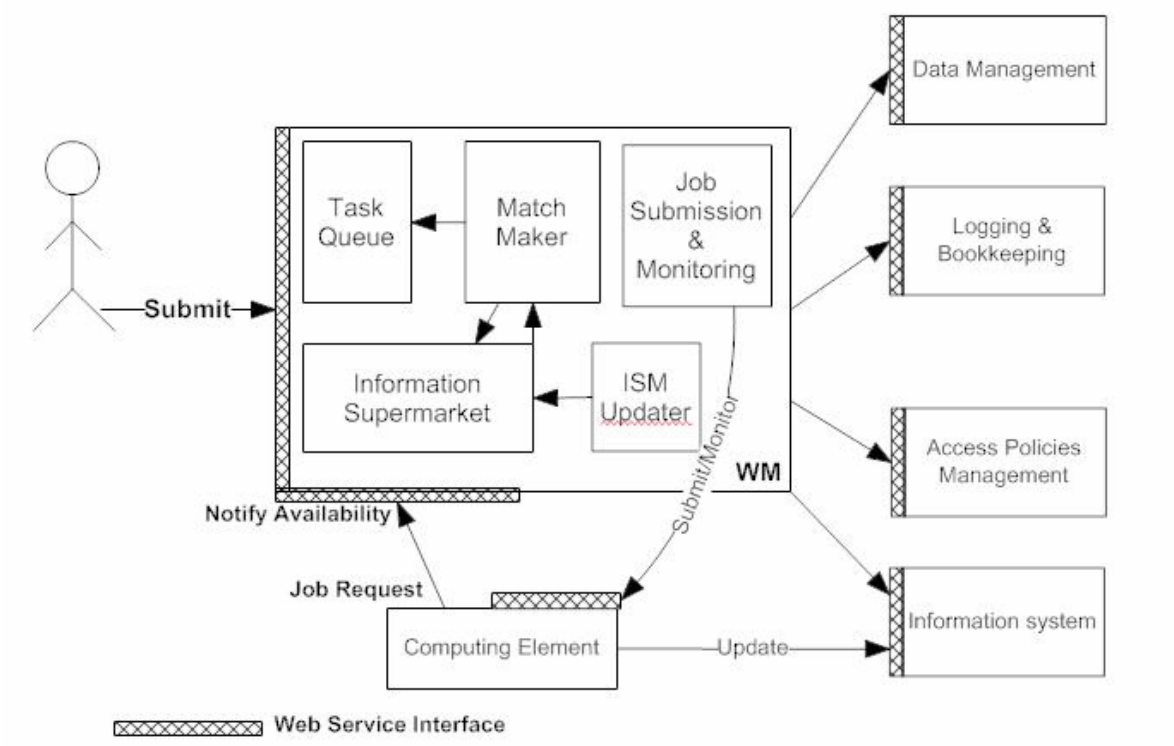


Figure 3.2. Job's Life [68]

In Figure 3.2, job management requests a job submission, or cancellation which is expressed with JDL. Task queue keeps submission requests. If no resource is available for matching a request, these requests are stored in this queue. It works with FCFS but different models like Shortest Job First (SJF), Longest Job First (LJF) can be also used. Information Supermarket is the repository of resource information and provides information to the matchmaker. Information in this component is updated via notifications or polling methods. Then, the matchmaker tries to find an appropriate resource, i.e. a CE, for each request. Meanwhile, Job Submission and Monitoring component performs actual job submission and monitoring operations.

Other components of WMS are as follows [68, 69];

- *Network Server (NS)* is responsible for accepting request from UI
- *Resource Broker (RB)* is responsible for matchmaking. Its goal is to find resources appropriate for job requests.
- *Information SuperMarket (ISM)* is a repository of resource information which is accessible by RB.

In our model, NS and RM are implemented but ISM is not explicitly implemented.

Monitoring and submission related components are as follow;

- *Job Adapter* is responsible for making final modifications over a JDL expression for a job before it is submitted to Condor. This is because, Condor works with ClassAds, and therefore, Job Adapter converts JDLs to equivalent ClassAds.
- *Condor* is the component that is responsible for doing the actual job management operations such as job submission and removal.
- *DAG Manager (DAGMan)* which is described in Chapter 2 in detail.
- *Log Monitor (LM)* is responsible for watching Condor log file, catch the events for active jobs. It is interested in events like job done and job cancelled

Job Adapter, Condor, DAGMan and LM are not implemented in our model,

In the real WMS architecture [68, 69], jobs states and transition between them are shown in Figure 3.3. Our model supports all the states except *Aborted* and *Canceled*.

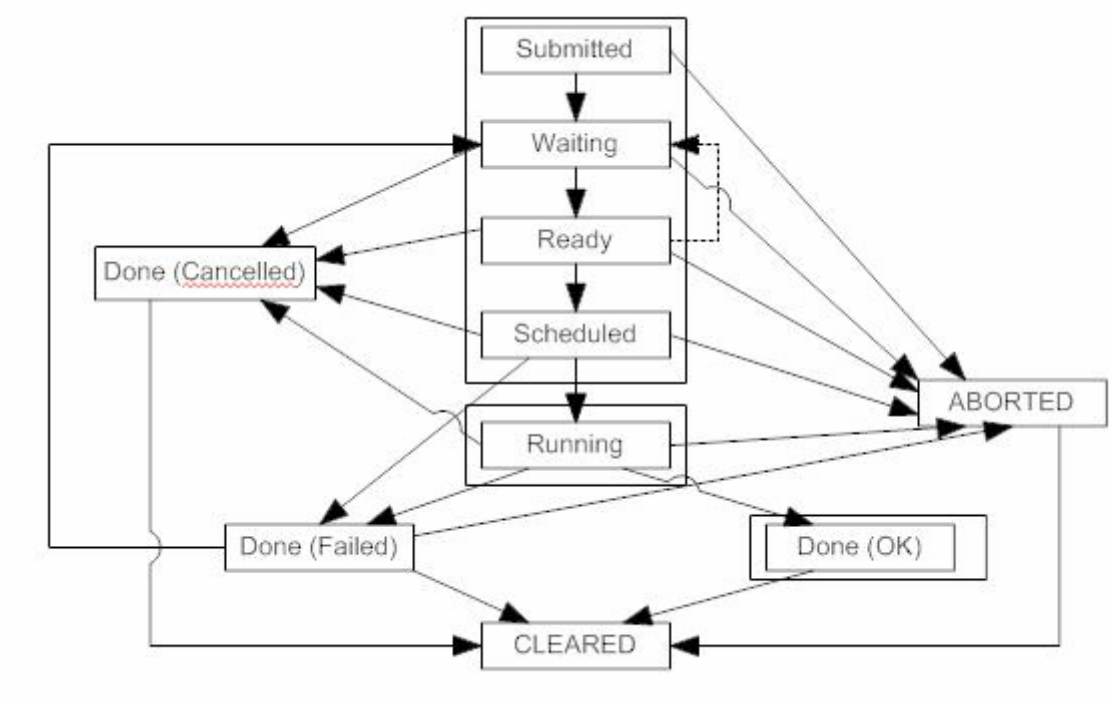


Figure 3.3. Job States [68]

- *Submitted* state indicates that, job is submitted by the user but it is not transferred to NS yet.
- *Waiting* state indicates that job is accepted by NS and waiting for WM for processing.
- *Ready* state indicates that job is processed by WM but not yet transferred to the CE queue.
- *Scheduled* state indicates that job is waiting in the CE queue.
- *Running* state indicates that job is running on CE.
- *Done* state indicates that job is done successfully or failed.
- *Aborted* state indicates that job was aborted by WMS because it may exceed WM queue or CE queue timeout.
- *Cancelled* state indicates that owner of the job canceled the request.
- *Cleared* state indicates whether job owner collected the outputs or whether the outputs are removed due to the timeout.

If something goes wrong, the WMS tries to schedule the job again. Therefore, matchmaking is performed again for this job and the job is now assigned to a different

resource. Number of resubmissions can be defined in JDL with attributes *RetryCount* for minimum and *MaxRetryCount* attribute for maximum number of trials. Job resubmission can be disabled by assigning zero to RetryCount attribute. In our simulated model, we do not implement this feature because our automatically generated JDLs always have at least one appropriate resource.

RB must find the most suitable CE where the job will be executed, so it collects information from Data Management Services and Information Services which provide all the information necessary for matching. In other words, RB chooses a CE with respect to job requirements. There could be more than one available resource. In this case, CE with the highest rank is chosen. In the worst case scenario, if there exists a resource with the same rank, random selection between these resources is done to find appropriate CE for the job.

Furthermore, there are three possible matchmaking scenarios which are as follows;

- *Direct job submission:* User defines the CE, which she wants to run job over, in the JDL file. There is no matchmaking process in this type of scenario.
- *Job submission with computational requirements only:* There is no *InputData* or *OutputSE* attribute defined in the JDL. Matchmaking is necessary for this type of scenario.
- *Job submission with data access requirements:* The attributes *InputData* and/or *OutputSE* are defined in JDL.

Our model supports “Job submission with computational requirements only”. In addition, users are assigned priorities based on past resource usage, policies. In this schema, users with better priority have higher probability to get what they want. However, in our model, the users are not assigned priorities.

In short, we simulate the WMS and UI with some exceptions in order to create grid environment. For the UI part of the system, only job submission is done. Job cancellation and data upload operations are excluded.

3.3 GridCAM

GridCAM is a simulation tool, which is developed by us in C++, that used to create environment explained in subsection 3.2. It simulates the WMS and UI with some exceptions in order to create grid environment. For the UI part of the system, only job submission is done. It creates random ClassAds with different distributions and keeps them in a queue. There are three different kinds of distributions which implemented in our tool. These are exponential, uniform (flat), and Gaussian distributions.

GridCAM has event oriented architecture and there are four different event types that are as follows;

- *Job Create Event* is responsible for creating a new job with given parameters and boundaries.
- *Job Done Event* is responsible for collecting outputs of completed jobs. It is also responsible for removing jobs from all queues.
- *Job Timeout Event* is responsible for removing jobs from queues hat exceed timeout limits.
- *Matchmaking Event* is responsible for matching resources and jobs by using not only Condor's matchmaking model but also our two proposed approaches. We tell the simulation which matchmaking model and parameters will be used via a configuration file.

In addition, for matchmaking purpose, GridCAM uses Condors Matchmaking Library Version 0.9.8 [33] and for the distributions GSL –GNU Scientific Library [70] with some modifications.

4. PROPOSED SCHEDULING HEURISTICS

In this chapter, our scheduling approaches are going to be presented. In the first part, some important WMS related information will be given. Then, explanation of the scheduling problem will be made and finally, our two scheduling heuristics will be given. Our new approaches are as follows;

- Scarce Resource First Matchmaking (SRFM)
- LP Based Matchmaking (LBM)

4.1 Introduction

Resource management and scheduling of distributed resources in grid environment are difficult problems. In grid frameworks, improvements were achieved in the past few years but there are still problems that need to be solved. Researches mostly focus on the subjects which includes workload management, resource discovery and matchmaking (brokering).

Significant results have been achieved on the problem of scheduling. In other words, efficient way of handling large number of jobs while mapping them to heterogeneous resources. In the area of scheduling and resource management, there are many unsolved issues that need to be tackled.

Basically, WMS comprises a set of grid middleware components responsible for the distribution and management of tasks over grid resources, in such a way that both sides are satisfied. The most important component of the WMS is the Workload Manager (WM) which is responsible for accepting and satisfying requests for job management. Requirements are expressed via a ClassAd-based JDL. Another important component is the job logging and bookkeeping which keeps tracks of jobs. For a job, there are two main types of requests which are *submission* and *cancellation*. Basically, the meaning of the job submission is passing the responsibility of the job to the WMS. The, the WM will pass the job to an appropriate CE that satisfies job requirements described in the JDL. The WMS keeps submissions while there is no appropriate resource. Condor uses this approach in its

systems. In addition, the decision of which resource will be used is the outcome of a matchmaking process which is done between resources and jobs. Our approaches contribute at this point. Basically, we propose two different approaches and the following sections in this chapter will give detailed information about these approaches.

4.2 Definition of the Problem

Matchmaking in grid environment is similar to matching in graphs in that in a given graph we try to obtain the largest subset of it edges with the property that no two are connected to same vertex [71]. This is also known as *maximum cardinality bipartite matching* problem. The problem is easier with some restrictions. For instances, matching between jobs and resources is an example of bipartite matching. Furthermore, perfect bipartite weighted matching is known as *Assignment Problem.* Here the purpose is that to find a perfect matching of minimum weight in a bipartite graph. In addition, assignment problem can be solved via network flow algorithms.

The following example will help to understand how to model this problem. Suppose that there are five jobs and five resources. The goal of the matchmaker is to find compatible matches between jobs and resources in a way that maximizes the number of jobs that are granted resources. Table 4.1 shows jobs and their compatible resources. The graph, corresponding to this table, represents the problem of maximizing the number of compatible matches.

Table 4.1. Compatibilities for Matching

	Resource 1 (C1)	**Resource 2 (C2)**	**Resource 3 (C3)**	**Resource 4 (C4)**	**Resource 5 (C5)**
Job 1 (J1)	-	Compatible	-	-	-
Job 2 (J2)	Compatible	-	-	-	-
Job 3 (J3)	Compatible	Compatible	-	-	-
Job 4 (J4)	Compatible	Compatible	-	-	Compatible
Job 5 (J5)	-	-	Compatible	Compatible	Compatible

Figure 4.1 shows the graph that represents Table 4.1. In figure, there is a weight which is equal to 1, for each compatible pairs.

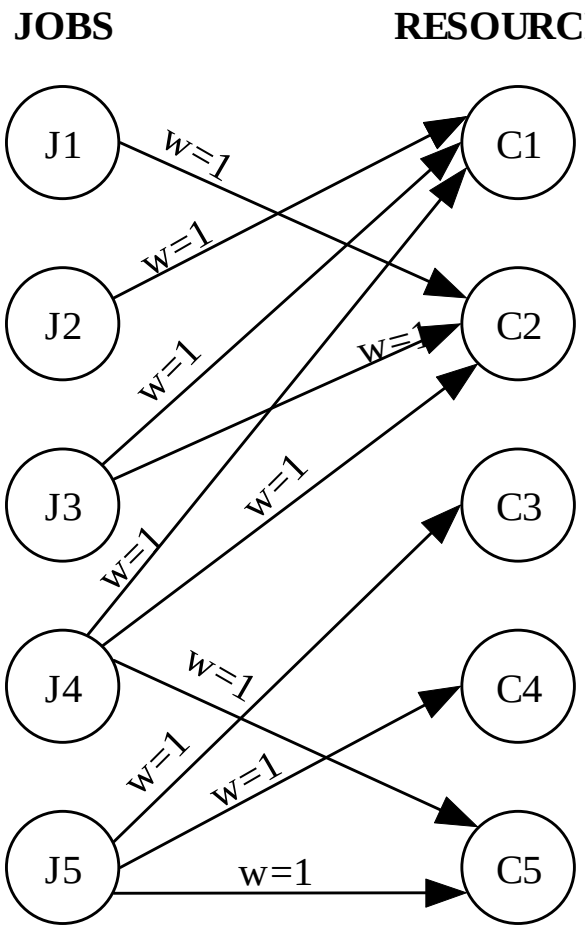


Figure 4.1. Graph Representation of the Problem.

4.3 Mathematical Model for Matchmaking

Suppose that P(J,R,C) represents our matchmaking problem where:

- $J = \{j_1, j_2, .., j_m\}$ is the set of jobs ,
- $C = \{c_1, c_2, .., c_m\}$ is the set of resources, each resource being a cluster.
- $R = \{ r_{jc}$: job j requests cluster c$\}$is the set of requests ,

Let r_{jc} represent the request of cluster c by job j. If $r_{jc} = 1$, it will mean job j will be assigned to cluster c and if $r_{jc} = 0$, job j will not be assigned to cluster c. The number of processors requested by job j will be denoted by q_j. The number of processors available on

cluster c will be given by p_c. In order to make our mathematical formulation general, we also associate a weight w_{jc} with each request r_{jc}. Given these, we try to maximize the number of jobs that have been assigned to clusters by setting all w_{jc} =1 and subject to the conditions that a job is assigned to at most 1 cluster and the total number of processors requested by jobs assigned to the same cluster is no more than the number of available processors on that cluster. Our objective and constraints can be expressed respectively as follows:

$$\text{Maximize} \quad \sum_{j \in J} \sum_{c \in C} w_{jc} r_{jc} \qquad (1)$$

subject to

$$\sum_{c \in C} r_{jc} \leq 1 \quad \text{for all } j \in J, \qquad (2)$$

$$\sum_{j \in J} q_j r_{jc} \leq p_c \quad \text{for all } c \in C, \qquad (3)$$

$$r_{jc} \in \{0,1\} \quad \text{for all } j \in J \text{ and } c \in C.$$

Note that this problem is similar to the generalized assignment problem [72] where the constraint (2) appears as

$$\sum_{c \in C} r_{jc} = 1.$$

This problem can be shown to be NP-hard by a transformation from the knapsack problem [73]. Therefore, fast heuristics need to be devised that will run fast enough in order to schedule submitted jobs rapidly. According to our simulated model match-making *window*, which is a duration denoted by d_{window}, must satisfy following;

$$T_{alg}(P(J,R,C)) < d_{window}$$

where T_{alg} denotes execution time of matchmaking. In other words, simulated model waits an amount of time (in seconds) called matching window or simply d_{window}, while, it accumulates jobs in the job queue. After this duration, matchmaking event is executed for all jobs, and algorithm tries to find appropriate resources for the jobs. Here the important point is that the duration of this matching process must be less than the duration of matchmaking window. For instance, suppose that matchmaking *window* is 120 seconds. Since *window* does not vary during the simulation, we have to ensure that duration of each matchmaking event must take less amount time then 120 seconds. In this way, we can ensure that we have fast heuristics in order to schedule submitted jobs rapidly. The matchmaking heuristics we describe below, tries to achieve these goals. The aim of these heuristics is to eliminate Condor's matchmaking algorithm's disadvantages. We illustrated one important disadvantage in Figure 1.1 which actually motivated us to propose following heuristics. For both SRFM and LBM heuristics, the idea that we try to implement is that rather than taking jobs one at a time, after specified amount of time, called *matching window*, we take all the jobs in the queue and try to match them

4.4 Scarce Resource First Matchmaking (SRFM).

The first approach is SRFM matchmaking and this heuristic works as follows: Jobs are treated in descending order of priority. There are three kinds of jobs and these are jobs with no compatible resource, jobs with only one compatible resource and jobs with more than one compatible resource. Actually this is only a logical method to group jobs. In other words, jobs are kept in the same job queue. In this approach, a quantity called the *degree*, is computed for each job which gives the number of compatible resource that can be assigned to a job. If no compatible resource (cluster) exists for a job or job cannot be assigned a resource (because the resource is no longer available), then the priority of the job is increased in order to prevent starvation problem. This is repeated in every matchmaking process until a timeout event of the job is triggered which causes it to be deleted from job queue. This job's degree is equal 0. However, if a job has only one compatible resource (degree=1), then it is assigned to the resource it requests. Here, our purpose is to avoid creating timeout events for these kinds of jobs. In other words, suppose there are jobs that not only have more than one compatible resource but also high priority. Furthermore, suppose that there are jobs that have only one compatible resource. Note that

these two groups may have a common resource for execution. Under these circumstances, jobs with high priority will get what they request. Job with one compatible resource must wait for a while. In addition, timeout event will be created for this job and it is possible that the job will reach this timeout threshold and will be removed for queue before execution. Furthermore, jobs that have more than one compatible resource are collected in a list. In other words, compatible and available resources for each job are determined and stored. Meanwhile, a resource desire value is computed for these jobs according to the following formula:

$$D(j,c) = (q_j * d_j * m_j) / (p_c * d_c * m_c) \quad (4)$$

Here, q_j, d_j and m_j refers to the required number of processors, amount of disk space and memory respectively requested by job *j*. Note that a high value of *D(j,c)* indicates an almost full resource and hence can be thought of as a scarce resource. Similarly, p_c, d_c and m_c are the available number of processors, amount of disk space and memory respectively on resource (cluster) *c*. These jobs with more than one compatible resource are then treated in descending order of resource desire value and assigned resources. Pseudocode for SRFM algorithm is presented in Figure 4.2

SRFM algorithm complexity can be examined in two parts. First part of the algorithm is the ranking phase and matching phase for the jobs with no resource and only one compatible resource. Complexity of the part is O(M*N) where M and N are total number of jobs and resources respectively. Second part is the matching phase for the jobs with more than one compatible resource. Complexity of the part is O(K) where K is the total number of jobs ($K \leq M$) that have more than one compatible resource. Therefore, overall time complexity of the algorithm is O(M*N). Note that necessary sorting operations are done by using priority queues so no additional sorting mechanism is implemented. The sorting operation takes O(MlogM).

```
Algorithm SRFM
Input: Jobs in queue
Output: Matched pairs (jobs and resources)

/* Ranking Phase */
for each job j ∈ J
{
        degree=0;
        for each resource c ∈ C
        {
            if check_feasibility(job j, resource c, q_j , d_j , m_j , p_c, p_c, p_c) = true
            {
                degree++;
                D(j,c) = ( q_j * d_j * m_j ) / ( p_c * d_c * m_c ) ;
            }
        }

/* Matching Phase for the jobs with no resource*/

        // No available resource for the job.
        if (degree = 0)        increase_priority(job j);

/* Matching Phase for the jobs with exacly one resource*/
        // Job has only one proper resource for execution.
        if (degree = 1)        assign_job (job j, r_jc );
        // first phase for the jobs that have more than one compatible resource
        if (degree >1 )        enter_job( job j, multi_resource_queue, D(j,c));
}

/* Matching Phase for the jobs that have more than one compatible resource */

    Sort jobs in descending order of resource desire value

    // Second phase for the jobs that have more than one compatible resource
    for each job j ∈ multi_resource_queue
    {
        if check_feasibility(job j, resource c, q_j , d_j , m_j , p_c, p_c, p_c) = true
        {
            assign_job (job j, c );
        }
    }

Done=1;
Return Done;
```

Figure 4.2. Pseudocode for SRFM Algorithm

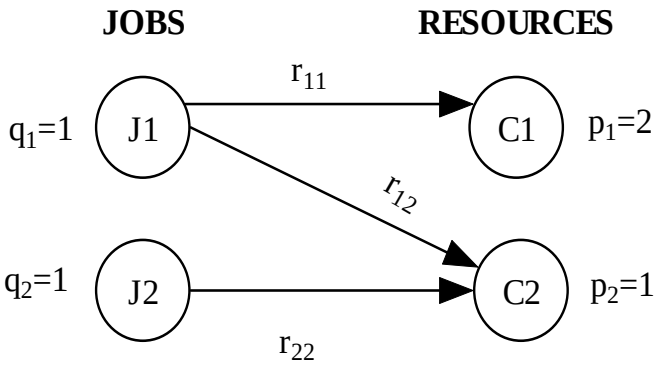
4.5. Linear Programming (LP) Based Matchmaking (LBM)

LBM heuristic uses Lp-solve which is a non-commercial linear programming code. It is written in ANSI C and is reported to solve problems as large as 30,000 variables and 50,000 constraints. Lp-solve can also handle smaller integer and mixed-integer problems. It is available at [74].

LBM matchmaking heuristic works as follows: First, jobs are treated in descending order of their priorities. After each matching window, heuristic starts collecting jobs from job queue and finds compatible resources. If no compatible resource (cluster) exists for a job or job cannot be assigned a resource (because the resource is no longer available), then the priority of the job is increased in order to prevent starvation problem. This is repeated in every matchmaking process until a timeout event of the job is triggered which causes it to be deleted from the job queue.

If a job has compatible resource or resources, its feasible requests are collected in a list. In other words, compatible and available resources for each job are determined and stored. Feasibility is determined via checking q_j, d_j and m_j *versus* p_c, d_c and m_c on resource *c.* After finding feasible job-resource pairs, they are converted into LP format input for Lp solver by using formulation expressed in section 4.3. This input is created in sparse matrix format in order to increase efficiency of the algorithm.

LP format contains three parts which includes objective function, constraint and declaration. First, *objective function* is a linear combination of optional variables and constants. In this part, we tell lpsolver what we try to maximize or minimize. This part is required, but can be empty. Secondly, *constraint* is optional and it is linear combination of variables and constants followed by a relational operator. The relational operator can be any of the followings: "<" "<=" "=" ">" ">=". Finally, *declaration* is the part that we define variables. Example shown in Figure 4.3 can be considered as a simple example problem for the Lp-solve.



(a)

LP Data

Max: $r_{11} + r_{12} + r_{22}$;
r_{11} <= 1;
$r_{12}+r_{22}$ <= 1;
r_{11} <= 2;
$r_{12}+r_{22}$ <= 1;
r_{11} <= 1;
r_{12} <= 1;
r_{22} <= 1;

(b)

Figure 4.3. Simple Matching Problem and its Input for Lp-solve

Suppose we have the following example in which:

- There are two jobs J_1 and J_2.
- There are two clusters C_1 with 2 processors and C_2 with 1 processor.
- The compatibilities are as shown in Figure 4.3 (a)

The linear program that corresponds to this example is as shown in Figure 4.3 (b). Solution for the problem is r_{11}=1, r_{12}=0, and r_{22}=1 as expected. In other words, edges r_{11} and r_{22} will be used so that *job 1* and *job 2* will be assigned to *cluster 1* and *cluster 2* respectively. Note that the result set is relaxed so solutions do not have to be integers. In other words, we do not have declaration part in our data. If we do not do this, then this problem is NP-hard.

Finally, requests (r_{jc}) are treated in descending order of Lp solver solutions and checked whether they are still feasible. If they are, jobs are assigned to resources. Pseudocode for LBM algorithm is presented in Figure 4.4.

LBM algorithm complexity can be also examined in two parts. In the first part of the algorithm, feasible job-resource pairs are determined. Complexity of this part is O(M*N) where M and N are total number of jobs and resources respectively. Second part is the matching phase and complexity of this part is dominated by complexity of the Lp-solve.

Note that the necessary sorting operations are done by using priority queues so no additional sorting mechanism is implemented. The sorting operation takes O(MlogM).

```
Algorithm LBM
Input: Jobs in queue
Output: Matched pairs (jobs and resources)

/* Determining feasible job-resource pairs  */
for each job j ∈ J
{
        degree=0;
        for each resource c ∈ C
        {
            if check_feasibility(job j, resource c, qj , dj , mj , pc, pc, pc) = true
            {
                degree++;
                Add request to Request_Queue;
            }
        }

/* Matching Phase for the jobs with no resource*/

        // No available resource for the job.
        if (degree = 0)        increase_priority(job j);
}

/* General Matching Phase */

    Lpdata=Using Request_Queue create input for Lpsolver;
    Lp_solve(Lpdata);

    Sort requests in descending order of lpsolver solution ;

    // Second phase: Assignment of jobs to resource
    for each request r ∈ Request_Queue
    {
        if check_feasibility(request r, resource c, qj , dj , mj , pc, pc, pc) = true
        {
            assign_job (job j, c );
        }
    }
Done=1;
Return Done;
```

Figure 4.4. Pseudocode for LBM Algorithm

5. SIMULATION RESULTS

5.1 Introduction

In this chapter, we will present experimental test results by using GridCAM. There are three main parts. In the second part of this chapter, simulation parameters will be given. Basically, distributions used and their parameters will be given. In the third part, the test results of our proposed approaches will be given.

5.2 Simulator Configuration

GridCAM uses three fundamental distributions which are exponential, uniform and gaussian.

Table 5.1. Simulation Parameters

	JOB Arrival (s)	**JOB Done (s)**	**Number of CPU**	**DISK (Byte)**	**MEMORY (Byte)**
Distribution	Exponential	Uniform	Uniform	Uniform	Uniform
Seed	Time	Time	Time	Time	Time
Mean	60	N/A	N/A	N/A	N/A
Parameter a	N/A	10	2	1000	100
Parameter b	N/A	3600	30	10000	1000
Model : FCFS			**Matchmaking Interval (s):** 1		
Model : SRFM and LBM			**Matchmaking Interval (s):** 120		

GridCAM, which is a tool to simulate Workload Management Service (WMS), uses GSL – GNU Scientific Library [70] for different distributions. In this simulation tool, time for job arrival is determined via an exponential distribution with mean 300 sec. Simulator determines when jobs will be done via uniform distribution with parameters 10 and 3600 sec. In other words, a time will be chosen for jobs between 10 and 3600 seconds, and the job will be done after this amount of time. In addition, if the type of the job is parallel (MPI) than the job needs more than one CPU. Simulator determines the required number of CPUs via a uniform distribution and its parameters are 2 and 30. In other words, MPI jobs will ask for between 2 and 30 CPUs. Disk space requirements for the jobs are also determined via a uniform distribution with parameters 1000 bytes and 10000 bytes. Finally, uniform distribution with parameters 100 to 1000 bytes is used to determine memory requirements of the jobs.

Configuration with d_{window}=1, given in Table 5.1, is used for FCFS algorithm. On the other hand, configuration with d_{window}=120, is used for both SRFM and LBM heuristics. The only difference between these two configurations is the d_{window} variable which was explained in Chapter 4.

5.3 Simulation Results

We have run simulations for four different time intervals: 1 hour, 2 hours, 10 hours and 24 hours. Simulations were run for FCFS, SRFM and LBM algorithms. Table 5.2 gives the simulation results for these algorithms. In the table, completed jobs and scheduled jobs indicate weighted-average of percentage of the number of the jobs completed and scheduled jobs respectively. The difference between the completed and scheduled jobs is that scheduled jobs were assigned to resources and they are still running whereas completed jobs are finished. In addition, waiting time for completed and scheduled jobs columns indicate the amount of time passed before a job is assigned to a resource. Since both completed and scheduled jobs do not wait for resources anymore, average waiting time values are computed not only for completed jobs but also scheduled jobs.

Table 5.2. Simulations Results of FCFS, SRFM, and LBM Algorithms

Schedule Scheme	Sim. Time	Jobs			Waiting Time for Completed and Scheduled Jobs (s)	
		Number Submitted	Percentage Completed	Percentage Scheduled	Max	Avg.
FCFS	1 Hrs	60	52,97	45,28	277	16,33
SRFM		69	53,18	45,24	163	61,87
LBM		60	48,51	51,23	120	59,71
FCFS	2 Hrs	115	81,16	16,49	574	20,04
SRFM		137	77,29	18,33	1289	84,68
LBM		129	70,97	26,40	751	75,37
FCFS	10 Hrs	567	62,42	4,87	1406	19,03
SRFM		530	84,59	3,23	5667	157,80
LBM		574	90,09	4,65	3214	72,10
FCFS	1 Day	1523	14,12	1,42	2361	40,43
SRFM		1414	83,84	5,03	5190	90,39
LBM		1429	88,01	2,27	6083	69,99

We have done 20 iterations for each simulation case except 1 day simulations. For one day simulations, we have done 4 iterations because of memory limitations of the machine we are using.

When you look at the results in Table 5.2, it is clear that SRFM and LBM are more successful then FCFS for long simulations. Our proposed heuristics' simulation results are quite encouraging and show that in long simulations with our SRFM and LBM scheme, we can have more jobs completed than the FCFS scheme. One disadvantage of our heuristics is the average waiting time values for both completed and scheduled jobs are higher than FCFS scheme. This was expected because we have more completed and scheduled jobs than FCFS. Furthermore, in our proposed heuristics we use a matchmaking window that also affects weighted-average value of maximum waiting times and average waiting times.

6. CONCLUSION

More and more people and industries are moving their applications to grids because grid computing presents with us an infrastructure where various resources can be shared. Therefore, the topic of resource schedulers that provide effective usage of the grid resources is becoming more and more important. Matchmaking process in a resource scheduler tries to assign jobs to available resources. We have to ensure that not only jobs achieve desired requirements but also efficient utilization of grid resources is achieved. In other words, we need an efficient resource selection scheme to maximize throughput. It is difficult to realize that, because jobs are matched to heterogeneous resources which have dynamically changing characteristics such as disk and CPU capacities.

Main idea of our approaches is to avoid excessive resource consumption, so we may increase the system throughput. In this study, we proposed two new polynomial-time heuristics for matchmaking based on this idea. A common idea shared by our heuristics is that both take a collection of jobs at each step and try to match as many jobs to available resources by using their own strategy. The first one is a collective matchmaking heuristic called Scarce Resource First Matchmaking (SRFM). In this heuristic, jobs are assigned by first attempting to match scarce resources. The second approach is an LP based heuristic that will solve relaxed version of our optimization problem and obtain matchings from the relaxed solution.

Our study is concluded with experimental tests by using GridCAM simulation tool. Simulation results are quite encouraging and show that in long simulations with our schemes, throughput of the system is increased. Thus, we can have more completed jobs than the FCFS scheme.

We have identified the following that can be done as future work;

- Dynamically change matchmaking window in order to optimize our heuristic algorithms.
- Adopt a conflict detection mechanism for ClassAds that is proposed in Coleman [47]. This might be helpful for our researches because users may create ClassAds that

contains predicates which conflict with each other. Colemon proposes a solution for this problem.

- DAGMan can be used to assign jobs to resources in case jobs have been submitted with precedence relations indicated by a DAG. Especially, DAGMan can be utilized when we change the execution order of jobs.

REFERENCES

1. “Wikipedia, the Encyclopedia”, http://en.wikipedia.org/wiki/Grid_computing, 2007

2. Fundamentals of Grid Computing, IBM Redbooks Paper by Viktors Berstis, November 2002.

3. “SEE-GRID”, http://www.see-grid.org, 2007

4. “SEE-GRID2”, http://www.see-grid.eu, 2007

5. “EGEE”, http://public.eu-egee.org, 2007

6. “EGEE2”, http://eu-egee.org, 2007

7. “Tr-Grid”, http://www.ulakbim.gov.tr/trgrid/, 2007

8. “gLite”, http://glite.web.cern.ch/glite/, 2007

9. Condor and the Grid, Douglas Thain, Todd Tannenbaum, and Miron Livny , University of Wisconsin-Madison, Madison, Wisconsin, United States, 2003

10. “Condor Researches”, http://www.cs.wisc.edu/condor/publications.html

11. “The Grid Physics Network (GriPhyN)”, http://www.griphyn.org, May 2007.

12. “The International Virtual Data Grid Laboratory (iVDGL)”, http://www.ivdgl.org, 2007.

13. Particle Physics Data Grid (PPDG), http://www.ppdg.net, August, 2002. IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing, 1988, pp. 62–82. Techniques in Physics Research (ACAT), 2000, pp. 161–163.

14. “NSF Middleware Initiative (NMI)”, http://www.nsf.gov/pubs/2003/nsf03513/nsf03513.htm, 2007.

15. “TeraGrid Project”, http://www.teragrid.org, 2007.

16. “NASA Information Power Grid”, http://www.gloriad.org/gloriad/projects/project000053.html, 2007.

17. “The National Computational Science Alliance”, http://www.ncsa.uiuc.edu/ , 2007.

18. Foster, I. and Kesselman, C. (1998) The globus project: a status report. Proceedings of the Seventh Heterogeneous Computing Workshop, March 4–19, 1998, citeseer.nj.nec.com/foster98globus.html.

19. Foster, I. and Kesselman, C. (1997) Globus: a metacomputing infrastructure toolkit. International Journal of Supercomputer Applications, 11(2), 115–128.

20. Pruyne, J. and Livny, M. (1994) Providing resource management services to parallel applications. Proceedings of the Second Workshop on Environments and Tools for Parallel Scientific Computing, May, 1994.

21. Wright, D. (2001) Cheap cycles from the desktop to the dedicated cluster: combining opportunistic and dedicated scheduling with Condor. Conference on Linux Clusters: The HPC Revolution, Champaign-Urbana, IL, June, 2001.

22. Thain, D. and Livny, M. (2002) Error scope on a computational grid: theory and practice. Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC), July, 2002.

23. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. (1988) A resource management architecture for metacomputing systems. Proceedings of the 5th ACM Conference on Computer and Communications Security Conference, 1998, pp. 83–92.

24. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S. (1998) A security architecture for computational grids. Proceedings of the 5th ACM Conference on Computer and Communications Security Conference, 1998, pp. 83–92.

25. Ferstl, F. (1999) Job and resource management systems, in Buyya, R. (ed.) High Performance Cluster Computing: Architectures and Systems. Vol. 1. Upper Saddle River NJ: Prentice Hall PTR.

26. Tannenbaum, T., Wright, D., Miller, K. and Livny, M. (2001) Condor – a distributed job Scheduler, in Sterling, T. (ed.) Beowulf Cluster Computing with Linux. Cambridge, MA; MIT Press.

27. Tannenbaum, T.,Wright, D., Miller, K. and Livny, M. (2001) Condor – a distributed job scheduler, in Sterling, T. (ed.) Beowulf Cluster Computing with Windows. Cambridge, MA; MIT Press.

28. Basney, J. and Livny, M. (1999) Deploying a high throughput computing cluster, in eds-Buyya, R. (ed.) High Performance Cluster Computing: Architectures and Systems. Vol. 1. Upper Saddle River NJ: Prentice Hall PTR.

29. Krueger, P. E. (1988) Distributed Scheduling for a Changing Environment, Technical Report UW-CS-TR-780, University of Wisconsin – Madison, Computer Sciences Department, June, 1988.

30. Frey, J., Tannenbaum, T., Foster, I., Livny, M. and Tuecke, S. (2001) Condor-G: a computation management agent for multi-institutional grids. Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC), San Francisco, CA, August, 7–9, 2001.

31. “European Union DataGrid Project”, http://www.eu-datagrid.org, 2007.

32. “Condor”, http://www.cs.wisc.edu/condor, 2007

33. “Condor ClassAds”, http://www.cs.wisc.edu/condor/classad, 2007

34. EGEE-JRA1-TEC-590869-JDL-Attributes-v0-8, Activity: JRA1 – Middleware, https://edms.cern.ch/document/590869/1, 2006

35. Ian Foster and Carl Kesselman, editors. The Grid: Blueprint for a Future Computing Infrastructure. Morgan-Kaufmann, 2004.

36. Pavel Fibich, Ludek Matyska, and Hana Rudová. Model of grid scheduling problem. In AAAI'05 Workshop on Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing. AAAI Press Technical Reports, 2005.

37. R. Raman, M. Livny, and M. Solomon. Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching. In Proceedings of the Twelfth IEEE International Symposium on High-Performance Distributed Computing, June 2003.

38. C. Liu, L. Yang, I. Foster, and D. Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In IEEE International Symposium on High Performance Distributed Computing (HPDC-11), July 2002.

39. M. J. Litzkow and M. Livny. Experience with the Condor Distributed Batch System. IEEE Workshop on Experimental Distributed Systems, 1990.

40. I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. IEEE Computer, 35(6), June 2002.

41. R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In The Seventh IEEE International Symposium on High Performance Distributed Computing, July 1998.

42. J. H. Saltzer, D. P. Reed, and D. D. Clark. End-To-End Arguments in System Design. ACM Transactions on Computer Systems, 2(4):277.288, Aug. 1984.

43. Data Grid WP1 – WMS Software Administrator and User Guide, Doc. Identifier: DataGrid-01-TEN-0118-1_28, November 2003.

44. An approach to grid scheduling by using Condor-G Matchmaking Mechanism E. Imamagic, B. Radic, D. Dobrenic University Computing Centre, University of Zagreb, Croatia

45. Imamagic E, Radic B, Dobrenic D. CROGRID Grid Execution Management System. In: Proceedings of the 27th International Conference on Information Technology Interfaces; 2005 Jun 20-23; SRCE University Computing Centre; 2005. p. 77-83.

46. Matchmaking, Datasets and Physics Analysis. Heinz Stockinger, Flavia Donno, Giulio Eulisse, Mirco Mazzucato, Conrad Steenberg Heinz.Stockinger@cern.ch

47. Distributed Policy Management and Comprehension with Classified Advertisements Nicholas Coleman, Rajesh Raman, Miron Livny and Marvin Solomon University of Wisconsin, 1210 West Dayton Street, Madison, WI 53703, April 2003

48. P. Godfrey. Minimization in cooperative response to failing database queries. International Journal of Cooperative Information Systems (IJCIS), 6(2):95. 149, June 1997.

49. Ontology-based Resource Matching in the Grid - The Grid meets the Semantic Web Hongsuda Tangmunarunkit, Stefan Decker, Carl Kesselman Information Sciences Institute University of Southern California

50. “The portable batch system”, http://pbs.mrj.com, 2007

51. Resource Management through Multilateral Matchmaking, Rajesh Raman, Miron Livny and Marvin Solomon University of Wisconsin, 1210 West Dayton Street, Madison WI 53703

52. A Constraint Language Approach to Grid Resource Selection, Chuang Liu Ian Foster

53. Condor and Preemptive Resume Scheduling, Alain Roy and Miron Livny Department of Computer Science, University of Wisconsin-Madison

54. A Dynamic Co-Allocation Service in Multicluster Systems, J.M.P. Sinaga, H.H. Mohamed, and D.H.J. Epema Faculty of Electrical Engineering, Mathematics, and Computer Science Delft University of Technology

55. Buyya R, Abramson D, Giddy J. Nimrod-G Resource Broker for Service-Oriented Grid Computing. IEEE Distributed Systems Online, Volume 2, Number 7. November 2001.

56. “The Gridbus Project”, http://www.gridbus.org, 2007

57. Venugopal S, Buyya R, Winton L. A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In: Proceedings of the 2nd International Workshop on Middleware for” Grid Computing; 2004 October 18; Toronto, Canada. ACM Press, 2004, USA.

58. Berman F, Wolski R, Casanova H, Cirne W, Dail H, Faerman M, Figuiera S, Hayes J, Obertelli G, Schopf J, Shao G, Smallen S, Spring N, Su A, Zagorodonov D. Adaptive Computing on the Grid Using AppLeS., IEEE Transactions in Parallel and Distributed Systems, Volume 14, Number 5. May 2003.

59. “GridWay”, http://www.gridway.org/, 2007

60. Open source metascheduling for Virtual Organizations with Community Scheduler Framework (CSF). Technical whitepaper, 2004. http://sourceforge.net/projects/gcsf/

61. Moab Grid Scheduler. http://www.clusterresources.com/products/mgs/

62. WP1 document of DataGrid (WP1: Workload Management, 25/09/2007) ISFN

63. From Metadata to Execution on the Grid Pegasus and the Pulsar Search, Ewa Deelman James Blythe Yolanda Gil Carl Kesselman Gaurang Mehta Karan Vahi, USC Information Sciences Institute, Marina Del Rey, CA 90292 Scott Koranda University of Wisconsin Milwaukee, Milwaukee, WI 53211, Albert Lazzarini Caltech, Pasadena, CA 91125, Maria Alessandra Papa, Albert Einstein Institute, Germany

64. SPHINX: A Fault-Tolerant System for Scheduling in Dynamic Grid Environments, Jang-uk In, Paul Avery, Richard Cavanaugh, Laukik Chitnis, Mandar Kulkarni and Sanjay Ranka University of Florida

65. “Atlas Experiment”, http://atlasexperiment.org/, 2007

66. “gLite Middleware”, http://www.glite.org, 2007

67. EGEE Middleware Architecture, EGEE-DJRA1.1-476451-v1.0 August 26, 2004. Document can be downloaded from https://edms.cern.ch/document/476451/

68. Tr-Grid, an Introduction to Grid Computing, http://www.grid.org.tr/etkinlikler/egitim/

69. Workload Management, in particular WMS User & Admin Guide and JDL docs http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/

70. “GSL – GNU Scientific Library”, http://www.gnu.org/software/gsl, 2007

71. Robert Sedgewick, Algorithms in C Third Edition, Princeton University, 2002, p. 67-68, p. 419-420

72. R. Ahuja, T. L. Magnanti, J. B. Orlin, Network Flows, Theory, Algorithms and Applications, Prentice Hall, 1993

73. M. Garey, D. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

74. “LP Solve Library”, http://lpsolve.sourceforge.net/5.5/