Chapter 14

GRID MULTICRITERIA JOB SCHEDULING WITH RESOURCE RESERVATION AND PREDICTION MECHANISMS

Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, Jan Weglarz
Poznan Supercomputing and Networking Center, Poland
naber@man.poznan.pl

Abstract

Grids link together computers, data, sensors, large scale scientific instruments, visualization systems, networks and people. They can provide very large pools of computer resources, enable distributed collaborations and deliver increased efficiency and on-demand computing capabilities. The complexity of Grids on one hand and the requirements towards performance and capability on the other hand call for efficient resource management and scheduling mechanisms. Such mechanisms must take into account not only the hardware and software resources, user jobs and applications, but also policies of the resource owners. Policies usually describe cost models for the resource usage, security mechanisms, quality of service of resource provisioning etc. The problem of scheduling jobs in real Grid environments is very difficult. Due to lack of time characteristics of jobs, and difficulties in characterizing the overall system, traditional OR techniques usually fail or achieve very weak results. Usually, best effort scheduling is the best option. There are, however, some ways to deal with the problems described above.

The main goal of this paper it to present some practical issues of scheduling Grid jobs. Methods and techniques described in the paper are used in a Grid scheduling system, called GRMS (Grid Resource Management System) developed at Poznan Supercomputing and Networking Center. GRMS is widely used in many Grid infrastructures worldwide.

Keywords: Grid computing, Grid resource management and scheduling, multicriteria decision support.

14.1 Introduction

Grid computing can be defined as coordinated resource sharing and problem solving in dynamic, multi-institutional collaborations. More simply, Grid computing typically involves using many resources (compute, data, I/O, instruments, etc.) to solve a single, large problem that could not be performed on any one resource. Often Grid computing requires the use of specialized middleware to mitigate the complexity of integrating of distributed resources within an Enterprise or as a public collaboration.

Generally, *Grid resource management and scheduling* is defined as the process of identifying requirements, matching resources to applications, allocating those resources, and scheduling and monitoring Grid resources over time in order to run Grid applications as efficiently as possible. Grid applications compete for resources that are very different in nature, including processors, data, scientific instruments, networks, and other services. Complicating this situation is the general lack of data available about the current system and the competing needs of users, resource owners, and administrators of the system.

Grids are becoming almost commonplace today, with many projects using them for production runs. The initial challenges of Grid computing–how to run a job, how to transfer large files, how to manage multiple user accounts on different systems–have been resolved to first order, so users and researchers can now address the issues that will allow more efficient use of the resources.

While Grids have become almost commonplace, the use of good Grid resource management tools is far from ubiquitous because of the many open issues of the field. Some of the issues include:

- **Multiple layers of schedulers.** Grid resource management involves many players and possibly several different layers of schedulers. At the highest level are Grid-level schedulers that may have a more general view of the resources but are very "far away" from the resources where the application will eventually run. At the lowest level is a local resource management system that manages a specific resource or set of resources. Other layers may be in between these, for example one to handle a set of resources specific to a project. At every level additional people and software must be considered.

- **Lack of control over resources.** Grid schedulers aren't local resource management systems; a Grid-level scheduler may not (usually does not) have ownership or control over the resources. Most of the time, jobs will be submitted from a higher-level Grid scheduler to a local set of resources with no more permissions than the user would have. This lack of control is one of the challenges that must be addressed.

- **Shared resources and variance.** Related to the lack of control is the lack of dedicated access to the resources. Most resources in a Grid environment are shared among many users and projects. Such sharing results in a high degree of variance and unpredictability in the capacity of the resources available for use. The heterogeneous nature of the resources involved also plays a role in varied capacity.

- **Conflicting performance goals.** Grid resources are used to improve the performance of an application. Often, however, resource owners and users have different performance goals: from optimizing the performance of a single application for a specified cost goal to getting the best system throughput or minimizing response time. In addition, most resources have local policies that must be taken into account. Indeed, the policy issue has gained increasing attention: How much of the scheduling process should be done by the system and how much by the user? What are the rules for each?

- **Missing time characteristics of jobs and tasks to be scheduled.** In Grids it is not possible to know most of time characteristics of jobs a priori. Time characteristics depend strongly on the performance and workload of a resource that is finally assigned to a job. The exact times are known only after the job is finished. Sometimes the users are able to give an estimate for their jobs. However, these estimates are very often far from the actual execution times. Time prediction methods might be also used to minimize the impact of this issue on a schedule quality. Another issue is a job ready time parameter. The job ready time depends on performance of the network and size of the job in terms of data that has to be moved from a local resource to a destination resource. The size of the data is also very often not know a priori. Very often, especially when we deal with jobs with precedence constraints, that size of the data to be moved from e.g. job n to job $n+1$ is known after the job n is finished and all the files this job generates are written to a disk.

- **Lack of resource reservation mechanisms.** Another issue is lack of resource reservation mechanisms for most of the resources in Grids. Although there is a lot of work being done in this area there are still huge limitations and technical constraints when it comes to resource reservation mechanisms. We must say that most of the local resource management systems, those that are responsible for scheduling on *destination resource*, i.e. a resource which actually runs the job do not support resource reservation for remote Grid schedulers. Usually the only way to make a resource reservation is to make a phone call to a resource admin.

In this paper we will focus on the last two issues: missing time characteristics of jobs to be scheduled and lack of reservation mechanism in Grid systems. Generally, there are a few main motivations behind an adoption of resource reservation and prediction mechanisms in Grid resource management. First, additional knowledge about job start and completion times helps to improve an efficiency of scheduling in Grid since a Grid resource broker can make more appropriate decisions. These mechanisms are also essential for providing a Quality of Service (QoS) for end-users. This is important especially for certain classes of applications and scenarios, e.g. interactive applications or scheduling with deadlines, and if resource usage is charged because end-users want to know what they are charged for. In addition, use of knowledge about job start and completion times enables a Grid resource broker to schedule the whole set of jobs at the same time that should lead to a better overall allocation of resources.

Having the above as a main motivations behind this paper we will go further and will present the whole problem as a multicriteria choice problem, in which a scheduler, or resource broker, choses one of many schedules generated upfront, while scheduling some sets of jobs waiting in the system global queue.

The issues of Grid resource management and scheduling have been addressed only in part by the relevant literature in the field of Grid computing. The first book on Grid computing, *The Grid: Blueprint for a New Computing Infrastructure* by Foster and Kesselman, and its updated second edition, available in 2004, are a good starting point for any researcher new to the field. In addition, the book by Berman, Fox, and Hey entitled *Grid Computing: Making the Global Infrastructure a Reality*, presents a collection of leading papers in the area, including the "Anatomy of the Grid" and the "Physiology of the Grid", two papers that provide an overview of the shape, structure, and underlying functionality of Grid computing. The most complete set of approaches to resource management in Grids was presented in the book by J. Nabrzyski, J. Schopf and J. Weglarz entitled *Grid Resource Management: State of the Art and Future Trends, Kluwer Academic Publishers, November 2003 (Nabrzyski et al (2003))*. Research results on the topic of resource management in Grid environments are presented regularly in selected sessions of several conferences, including Supercomputing (SC), the IEEE Symposium on High-Performance Distributed Computing (HPDC), and the Job Scheduling Strategies for Parallel Processing workshop, as well as in the Global Grid Forum, a standards body for the field. The chapter is structured as follows: Section 2 gives an overview of resource reservation in Grid systems. Section 3 shows how prediction mechanisms can help with scheduling jobs in Grid environment. Next, in Section 4 we present a scenario of Grid job scheduling with predictions and resource reservations. We introduce a general model of multicriteria choice problem, which is one of the scheduling strategies in the GRMS (Grid Resource Management System) scheduling framework. GRMS itself is described in Section 5 and its practical

usage example in Section 6. We conclude with the summary section in which we also sketch out our future research.

14.2 Resource Reservation in Grid Systems

Why is resource reservation in Grids so important? Let us give an example showing the main issues of resource reservation in Grids. A possible Grid scheduling scenario is presented in the Figure 1 below. The picture is not very much different from classical scheduling of machines. Of course none of time characteristics given in the picture are known a priori in Grids, as already explained above.

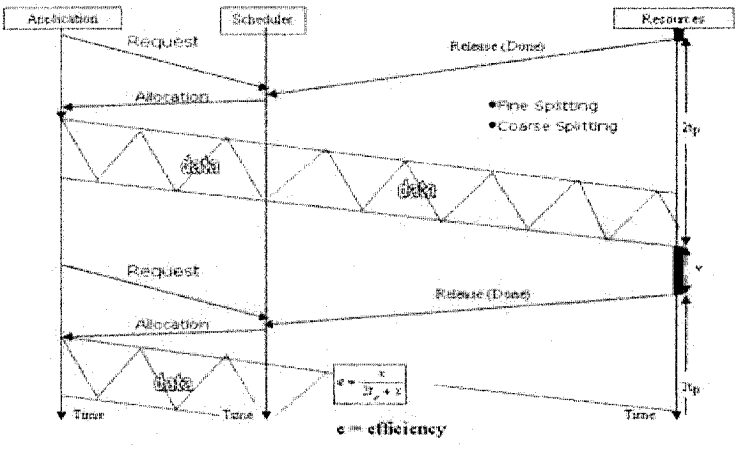

Figure 14.1. Most common Grid scenario: resource reservation is not supported by resources.

In the example a resource is allocated to a job for an unspecified amount of time, starting at the time the job-to-resource assignment is decided. As far as the information available at scheduler is concerned, the resource remains allocated to a job until the scheduler is informed about the job completion. The scheduler has to wait for a *release message* from the resource before it can allocate the resource to a new job. When a release message is received at the scheduler, an *allocation message* is sent to the application informing the user-side of where the available data for the execution of the job should be sent. The data are then transferred, which takes some additional communication time. During the transmission of the allocation message and during the transmission of the data to the resource, the resource remains (unnecessarily) inactive. When all the data are submitted the job begins execution on the resource. After the job is completed, the resource remains again (unnecessarily) inactive until the scheduler receives the release message so that it can then allocate it to another job. The figure shows actually the scheduling scenario when no resource reservation is applied.

We denote by $2t_p$ the average time that elapses between the time the resource sends the release message to the scheduler to inform it about the completion of a job until the time all the data required to execute the next job arrive at the resource. We also denote by x the average execution time of a job. It is then clear that Grid scheduling without resource reservation is performed the efficiency with which a resource is used is at most

$$e = \frac{x}{2t_p + x} \quad .$$

Note that the efficiency factor e may be considerably smaller than 1, and it also gets smaller as X decreases or as $2t_p$ increases ($2t_p$ is at least as large as the roundtrip propagation communication delay). In order for the Grid to be useful for a number of different applications, we would like to be able to use fine grain computation (where x is small) and also be able to use remote resources (where $2t_p$ is large), both of which correspond to small values for the efficiency factor e.

This scenario shows also that some additional overhead must be taken into account when scheduling on the Grid. This overhead is caused by additional communication that must take place between the scheduler and the jobs that are to be run on potentially remote machines.

Let us now see how the efficiency factor e would look like with resource reservation supported by the remote resource. In the example we assume that a remote resource can be reserved in advance. It means we know when the resource will be fully available for our job so we can send all the data to be placed on the resource when the time to run a job comes.

The main idea behind advance/timed resource reservation is that the resources are reserved only for the time during which they are actually used for a job. In order to do so, the scheduler needs to reserve some execution time in the resources in advance. Of course we still do not know exactly how long will it take to run a particular job, but resource reservation maximizes the efficiency of the resources and the efficiency factor e can get very close to 1.

14.3 Scheduling Grid Jobs Using Prediction Information

Most of existing available resource management tools use general approaches such as load balancing (Shirazi et al (1995)), matchmaking (e.g. Condor Condor (www)), computational economy models (Abramson et al (2002)), or multi- criteria resource selection (Kurowski et al (2000b)). In practice, the evaluation and selection of resources is based on their characteristics such as load, CPU speed, number of jobs in the queue etc. However, these parameters can be related to the actual performance of applications, which may be not known a priori by end users. Users usually do not know what is the exact influence of these parameters on job start (e.g. local queue waiting) and execution times at

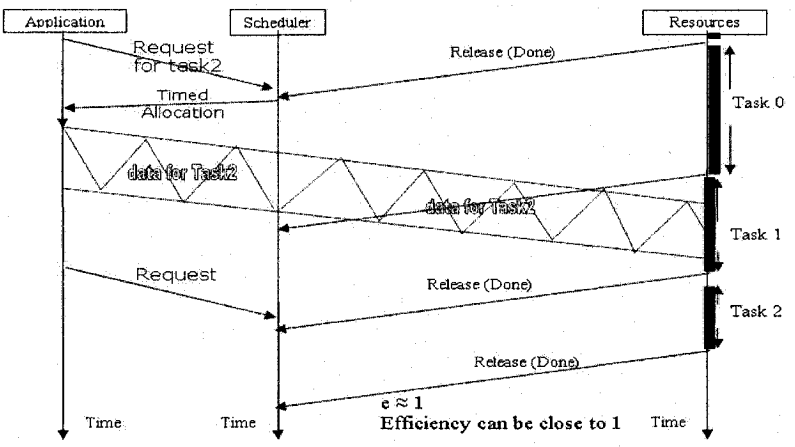

Figure 14.2. Resource reservations when advance and timed reservations are used.

different machines. Therefore, available estimations of job start and run times may significantly increase a quality of scheduling and, consequently, the overall performance. Nevertheless, due to incomplete and imprecise information, results of performance prediction methods may be accompanied by considerable errors (see Gibbons (1997), Smith et al (1999)). The more distributed, heterogeneous, and complex environment the bigger prediction errors may occur. Thus, these errors should be estimated and taken into consideration by a Grid scheduler while evaluating available resources or schedules. Our approach to estimating job start and run times has been presented in (Kurowski et al (2005)). This method takes into account estimated prediction errors to improve decisions of the Grid scheduler and to limit their negative influence on overall performance. In the method, the predicted job start- and run-times are generated by the Grid Prediction System (GPRES), developed in our collaboration with Wroclaw Supercomputing Center. Prediction techniques can be applied in a wide area of issues related to Grid computing: from the prediction of the resource performance in a near future to the prediction of the queue wait time (Smith et al (1999)). Most of these predictions are applied to resource selection and job scheduling. Prediction techniques can be classified into statistical, AI, and analytical. Statistical approached are based on applications that have been previously executed. These can be time series analysis (Dinda (2001), Wolski et al (1999)), categorization (Smith et al (1999), Downey (1997), Gibbons (1997), Kurowski et al (2000a)), and in particular correlation and regression have been used to find dependencies between job parameters. Analytical techniques con-

struct models by hand (Schopf and Berman (1998)) or using automatic code instrumentation (Taylor et al (2001)). AI techniques use historical data and try to learn and classify the information in order to predict the future performance of resources or applications. AI techniques that can be used here are, for instance, classification (decision trees (Quinlan (1986)), neural networks (Rumelhart et al (1986))), clustering (k-means algorithm (Darken and Moody (1990))). Predicted times are used to predict resource information to guide scheduling decisions. Such scheduling process can be oriented to load balancing when executing in heterogeneous resources (Dail (2001), Figuiera and Bermann (2001)), or resource selection (Kurowski et al (2000b)) or used when multiple requests are provided (see Czajkowski et al (1997)). For instance, in Liu et al (2002) the 10-second ahead predicted CPU information is provided by NWS (Wolski (1997), Wolski et al (1999)). Many local scheduling policies, such as Least Work First or Backfilling, also use user provided or predicted execution time to make scheduling decisions (Lifka (1995), Feitelson and Mu'alem Weil (1998), Feitelson (www)).

In the approach presented in Kurowski et al (2005) we use the GPRES Expert System, which uses very simple template approach for predictions (Smith et al (1999)). The template is the set of job attributes, which are used to evaluate jobs similarity. Attributes for templates are generated from historical information after tests. Prediction process consists of several steps:

1 Initialize empty job category set C_z

2 For every template $T_i \in T$

- generate job category c_i
- add c_i to C_z

3 Initialize empty decision category set C_d

4 For every category c_i to C_z

- select categories from Knowledge Base corresponding to category c_i
- add categories to C_d

5 select best category from C_d

Where: d – decision attribute, T – template set, C – category set.

The method of selecting the best rule (category) can be set as a parameter to prediction module. Actually there are avaliable two methods in GPRES. The first one is based on number of condition attributes in rules. The most specific rule is chosen, i.e. the rule which has attributes of the job matched to the condition in the best way. The second strategy prefers a rule generated from

the largest amount of history jobs. GPRES allows to mix these two methods in the way that if the first one gives still several rules the second is used. If both methods don?t give the final selection, the rules are combined and arythmetic mean of decision attribute is returned. Experiments presented in Kurowski et al (2005) proved that use of knowledge about estimated job completion times may significantly improve resource selection decisions made by Grid scheduler and, in this way, the performance of applications and the whole Grid system. Nevertheless, estimated job completion times may be insufficient for effective resource management decisions. Results of these decisions may be further improved by taking the advantage of information about possible uncertainty and inaccuracy of prediction. In the next section we will present the multicriteria Grid job scheduling model in which both, resource reservation and prediction mechanisms are used to improve scheduling performance.

14.4 Grid Job Scheduling with Predictions and Resource Reservations

Figure 3 presents a general Grid resource management scenario with resource reservation and time prediction mechanisms.

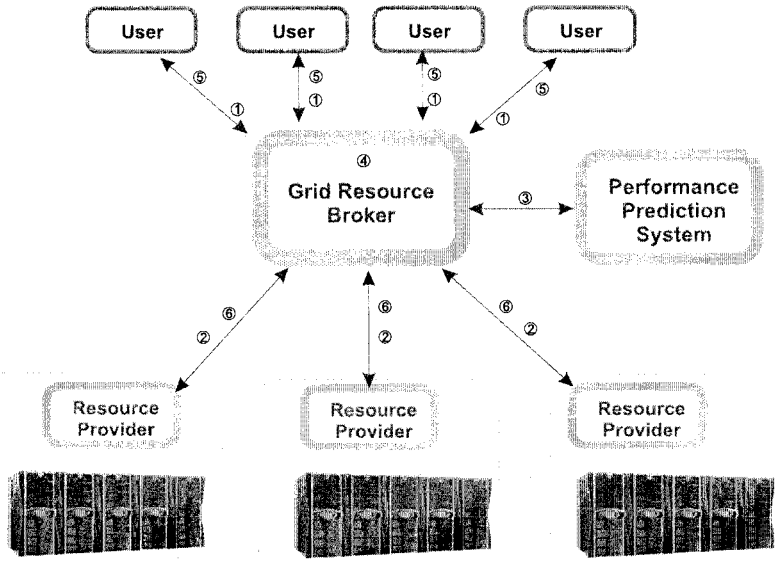

Figure 14.3. Grid resource management using resource reservation and performance prediction mechanisms

Resource broker after receiving resource requests from users (step 1) asks resource providers about their offers. Offers are returned in a form of lists of

amounts of available resource units in various time slots (step 2). Providing an offer a resource provider agrees to initially reserve resources for a certain period. If a reservation is not confirmed before the end of this period the reservation is canceled. This approach guarantees that resources will not be reserved by other consumers. In the next step a performance prediction system provides knowledge about estimated job start and completion times (3). The prediction system calculate estimations based on historical information containing traces of previous job submissions. Following this a resource broker filters offers according to constraints defined by end-user, choses the best schedule (4) and returns this information to users or software acting on behalf of them (5). For users that have accepted a schedule given by the broker the reservations are confirmed with appropriate providers (6).

14.4.1 Model of the Scenario

In this section we define more formally a model adopted for the described scenario. End-users from a finite set $U = u_1, u_2, ..., u_{|U|}$ need to run their jobs $J = j_1, j_2, ..., j_{|J|}$ on resources belonging to resource providers from the set $RP = rp_1, rp_2, ..., rp_{|RP|}$. For each job resource requirements are defined. They are modeled as a set of hard constraints that must be met as explained in the previous sections. They consist of amounts of resource units RU^{req} that must be reserved for a given job (e.g. 3 CPUs, 1GB of disc space, etc.) and required resource attributes Q^{req} (e.g. CPU speed at least 1TFlops).

In this model we assume that a scheduler has knowledge about job start times. Thus, each resource provider must provide information about its offers in a form of lists containing available resource units in certain time slots in a given time period (t_0, t_f): $RT_i(t_o, t_f) = rt_{i1}, rt_{i2}, \ldots, rt_{ik}$, $k = |RT_i|$, $i = 1, \ldots, |RP|$, $rt_{ik} = (t_i^{start}, t_i^{end}, RU_{ik}, Q_{ik})$, where $RU_{ik} = (ru_{ik1}, ru_{ik2}, \ldots, ru_{ik|RU_{ik}|})$, and ru_{ikl} is an amount of the available resource unit l for resource provider i and time slot k (e.g. 100MB of free memory) that can be reserved for an end-user. Q_{jk} is a set of resource attributes as described in the previous sections (e.g. CPU speed, operating system, etc.).

In addition to knowledge of deterministic (guaranteed) job start times, information about estimated job execution and completion times is assumed to be available as explained above. Therefore, the Grid scheduler can take the advantage of the list of estimated job execution times, which can be calculated by the prediction system on the basis of resource attributes provided by each resource provider for a certain reservation: et_{ijk}^{exec} where $i = 1, ..., |J|, j = 1, ..., |RP|, k = 1, ..., |RT_i|$. Estimations are calculated on the basis of Q_{ik} - a specification of parameters describing a resource belonging to a given resource provider. Since job execution times are available and we assume that reserved jobs can start earlier if there is such a possibility, real job start times

can also be estimated. These times, denoted as et_{ijk}^{start} where $i = 1, ..., |J|, j = 1, ..., |RP|, k = 1, ..., |RT_i|$, may be significantly shorter than the guaranteed ones. They can be provided either by a prediction system if this information cannot be taken from resource providers or by resource providers themselves (in the latter a broker or prediction system should estimate possible errors of predictions delivered by resource providers).

14.4.2 Multicriteria Choice Problem

The problem to be solved is to find the best time slot (resource providers' offer) for each job according to end-user's requirements. Each assignment of a job to a time slot ($j \rightarrow rt$) is a candidate solution (also called action using a decision support terminology) and denoted as $a \in A$, where A is a set of all candidate solutions. Requirements consist of constraints that must be satisfied (hard constraints) and preferences concerning a choice of the best solution (soft constraints).

In the first step offers of resource providers must be filtered according to hard constraints defined by an end-user. This step can be performed by resource providers themselves since they retrieve information about job requirements from a resource broker in order to decide according to their local policies if there are any offers for a job. To this end two issues are cross-checked. For each job k and offer $rt_{ij}, i = 1, ..., |RP|, j = 1, ..., |RT_i|$ a resource broker (or provider) checks if requirements $Q_k^{req}, k = 1, ..., |J|$ concerning resource attributes Q_{ij} are met, i.e. whether $\forall_{q_{ijl} \in Q_{ij}} (q_{ijl} \propto q_{kl}^{req})$. $\propto$ denotes a relation between resource attribute (q_{ijl}) and a job requirement concerning this attribute (q_{kl}^{req}). This relation occurs if and only if q_{ijl} matches q_{kl}^{req}, e.g. is less, equal or greater than required values depending on particular attributes. In the second step it is checked if a sufficient amount of resource units can be reserved, i.e. whether $\forall_{ru_{ijm} \in RU_{ij}} (ru_{ijm} \geq ru_{km}^{req})$. It can be done again by a resource provider or a resource broker.

In addition to hard constraints (C) that must be satisfied, a Grid resource broker needs criteria (soft constraints) that define how the best resources should be selected. End-user can specify more than one soft constraint, e.g. time and cost. To handle such requests modeling and exploitation of multi-criteria users' preferences is needed.

Various models of preference modeling can be adopted in Grid resource management (Greco et al (1998)). In general we can distinguish two ways of preferences acquisition: (i) preferences are given explicitly by an end-user, e.g. in the form of criteria weights or criteria ranking, and (ii) end-users' preferences are discovered on the basis of their decisions (comparison of potential solutions). Which method should be used depends on two major aspects: first, whether users are familiar with basic concepts of decision support theory and

Table 14.1. Criteria

No	Symbol	Description
Cr_1	gt^{start}	Guaranteed job start time (according to an agreement concerning advance resource reservation)
Cr_2	$cost$	Total cost of reservation
Cr_3	$mean\ et^{exec}$	Estimated mean job execution time (based on job description and parameters of selected resource)
Cr_4	$stdev\ et^{exec}$	Estimated standard deviation of job execution time
Cr_5	$max\ et^{exec}$	Estimated maximal value of job execution time
Cr_6	$err\ et^{exec}$	Estimated prediction error of job execution time
Cr_7	$mean\ et^{start}$	Estimated mean job start time (based on estimated execution times of previously scheduled jobs)
Cr_8	$stdev\ et^{start}$	Estimated standard deviation of job start time
Cr_9	$max\ et^{start}$	Estimated maximal value of job start time
Cr_{10}	$err\ et^{start}$	Estimated prediction error of job start time

aware how to express preferences and, second, whether their preferences are relatively stable. If preferences change for each job submission, e.g. due to different application requirements or certain unpredictable aspects, an automated learning of users' preferences is very difficult. Then methods based on utility theory or lexicographic order of criteria can be applied.

In the presented model criteria considered in the decision support process are time and cost related. Nevertheless, in addition to the main criteria: guaranteed (reserved) job start time and cost, criteria that define estimated job execution time and start time are also taken into consideration. For both of these metrics the imprecision measures such as standard deviation, maximal value, and estimated prediction error are defined (as another criteria). The estimated execution time let differentiate the quality of available resources. The estimated start time can be significantly less then the guaranteed one since we assumed in the model that resource providers can shift jobs if previous ones have finished earlier. These values can be returned by a prediction system but also provided by a resource provider itself. The complete list of criteria used in the model is presented in Table 1. Of course, additional criteria can be easily added without reducing the generality of the model.

Although the set of criteria is quite big, in most cases only a subset of them is used. For instance, probably only one of prediction imprecision measures is relevant at the same time for an end-user. As mentioned above, different methods of preference modeling can be applied, however, here we present a

procedure of resource selection using a utility function. For each pair: a job and time slot a utility function is calculated according to the formula:

$$F(j_k, rt_{ij}) = \sum_{l=0}^{|CR|} w_{kl} f_l(rt_{ij}), \tag{14.1}$$

$$f_l(rt_{ij}) = \frac{(Cr_{kl} - min_{kl})}{(max_{kl} - min_{kl})}, \tag{14.2}$$

where $i = 1, ..., |RP|, j = 1, ..., |RT_i|, k = 1, ..., |J|$, w_{kl} is a normalized weight of l^{th} criterion concerning job k, and $|CR|$ is a number of criteria (in this case $|CR| = 10$)

The max_k and min_k values are essential for appropriate scaling of criteria values. The function (1) is the simplest utility function consisting of one linear section. The advantage of this preference model is that this can be relatively easily and quickly defined by an end-user and calculation of utility function is immediate. Using this function a resource broker evaluates resources for one job only.

Based on the definitions, notations, and considerations described above the problem can be generally defined in the form of a multi-criteria decision support problem as follows:

$$min\{f_1(a), f_2(a), ..., f_{|CR|}(a)\}, \tag{14.3}$$

s.t.

$\forall_{q_{ijl} \epsilon Q_{ij}} (q_{ijl} \propto q_l^{req})$,
where $q_{ijl} \epsilon Q_{ij}$, $q_l^{req} \epsilon Q_k^{req}$, $l = 1, ..., |Q|$

$\forall_{ru_{ijm} \epsilon RU_{ij}} (ru_m \geq ru_m^{req})$,
where $m = 1, ..., |RU|$

$a = j \rightarrow rt_{ij}, i = 1, ..., |RP|, j = 1, ..., |RT_i|$

14.4.3 Scheduling of Job Sets

Knowledge about job completion times gives to a Grid resource broker a possibility to schedule more that one job at the same time. Doing this a Grid resource broker can try to optimize a whole schedule like in classical scheduling approaches. Otherwise, using online scheduling, an order of jobs in a queue may strongly influence a quality of a schedule. Additionally, if a broker schedules multiple jobs at once resource providers are asked to make preliminary reservations of their resources only once for all jobs. Note that in the presented model the main goal is to maximize a total satisfaction of end-users instead of fixed global criteria.

When multiple jobs are being scheduled a resource broker must get resource providers' offers not only for a single job. Therefore, resource provider should specify jobs that can be run in given time slots. Thus, for each time slot the following list must be provided: $JT(rt_{ij}) = \{j_1, j_2, ..., j_{|J|}\}$. Note that one time slot can be reserved for multiple jobs if there are enough resource units available in this slot.

A consequence of scheduling sets of jobs, which have come in a certain time interval, is a need for solutions that satisfy in the best possible way objectives of multiple end-users. Therefore, a total users' satisfaction must be evaluated. To this end, preferences of all end-users have to be aggregated into a measure that allows a resource broker to select the best schedule. A method of aggregation depends on an approach used for modeling user's preferences. If a utility function is used for criteria aggregation, an evaluation of the whole schedule is performed according to the following formula:

$$FT(J, RT) = \frac{1}{|J|}\sum_{k=0}^{|J|} F(j_k, rt_{ij}), \tag{14.4}$$

where $i = 1, ..., |RP|, j = 1, ..., |RT_i|, k = 1, ..., |J|$ and rt_{ij} is a time slot chosen for job k.

In order to make this aggregation fair and reasonable $min_k l$ and $max_k l$ values in formula (2) must be specified carefully. They should define very bad and very good values of a criterion respectively from an end-user's perspective. Otherwise, if for example minimal and maximal values from a set of candidate solutions are taken as $min_k l$ and $max_k l$, utility functions of different users are totally incomparable. In spite of the same values of these functions for two solutions evaluated by two different users, the real assessment of these solutions may significantly differ. Therefore, if $min_k l$ and $max_k l$ cannot be accurately defined other methods of aggregation should be used instead. For instance, if dependencies between solutions are given in a form of partial preorder of solutions the aggregation procedure based on Net Flow Score (Greco et al (1998)) can be applied.

The formulation of this problem differs from that defined for single-job scheduling described above. In this case a candidate solution is an assignment of jobs to time slots offered by resource providers. Generally multiple jobs can be assigned to one time slot. Additionally resource providers should return information which jobs can be assigned to a given time slot.

$$min\{f_1(a), f_2(a), ..., f_{|CR|}(a)\}, \tag{14.5}$$

s.t.

$\forall_{i,j}\forall_{k:(j_k \to rt_{ij}) \epsilon a}(q_{ijl} \propto q_{kl}^{req})$,

where $q_{ijl} \epsilon Q_{ij}, q_{kl}^{req} \epsilon Q_k^{req}, l = 1, ..., |Q|$

$\forall_{i,j} (\sum_{k:(j_k \rightarrow rt_{ij}) \epsilon a} ru_{km}^{req}) \leq ru_{ijm}$,

where $ru_{ijm} \epsilon RU_{ij}, ru_{km}^{req} \epsilon RU_k^{req}, m = 1, ..., |RU|$

$\forall_{(j_k \rightarrow rt_{ij}) \epsilon a} j_k \epsilon JT(rt_{ij})$

$a = \{j_1 \rightarrow rt_{ij}, j_2 \rightarrow rt_{ij}, ..., j_{|J|} \rightarrow rt_{ij}\}, i \epsilon \{1, ..., |RP|\}$,
$j \epsilon \{1, ..., |RT_i|\}, k \epsilon \{1, ..., |J|\}$

The set a is a candidate solution (decision action). It consists of an ordered list of time slots assigned to every single job that belongs to the set J. The first constraint ensures that all time slots meet requirements of assigned jobs concerning resource attributes. The goal of the second constraint is to guarantee that sums of resource units that have to be allocated to assigned jobs do not exceed those offered by resource providers. As explained earlier Q_{ij} and RU_{ij} mean attributes of resources and amounts of resource units offered by resource providers respectively. Q_k^{req} and RU_k^{req} are corresponding job requirements concerning these values.

14.5 GRMS - An Example Grid Scheduling Framework

GRMS, (Kurowski et al (2001), Kurowski et al (2003), Kurowski et al (2004)), is an open source meta-scheduling system for large scale distributed computing infrastructures (Allen et al (2003)), developed at Poznan Supercomputing and Networking Center. Based on the dynamic resource selection, mapping and advanced Grid scheduling methodologies, it has been tailored to deal with job and resource management challenges in Grid environments, i.e. load-balancing among clusters, setting up execution environments before and after job execution, remote job submission and control, file staging, and more. GRMS was developed entirely in Java and thus can be installed on various kinds of operating systems and resources. GRMS is infrastructure independent and can be easily integrated with various Grid infrastructures, including all versions of Globus (Globus (www)), as well as enterprise Grids based on DRMAA-based infrastructures, including Condor, Sun N1GE (drmaa (www)) GRMS is able to take advantage of other middleware services, e.g. the Grid Authorization Service (GAS) or Replica Management Services, as well as to interoperate with infrastructure monitoring tools such as the GridLab's Mercury Monitoring System. One of the main assumptions for GRMS is to perform remote job control and management in the way that satisfies users' (job owners) requirements (Kurowski et al (2003)).

The main GRMS functionality includes: queuing submitted job, finding the best resources according to users' preferences, staging in/out files, submitting

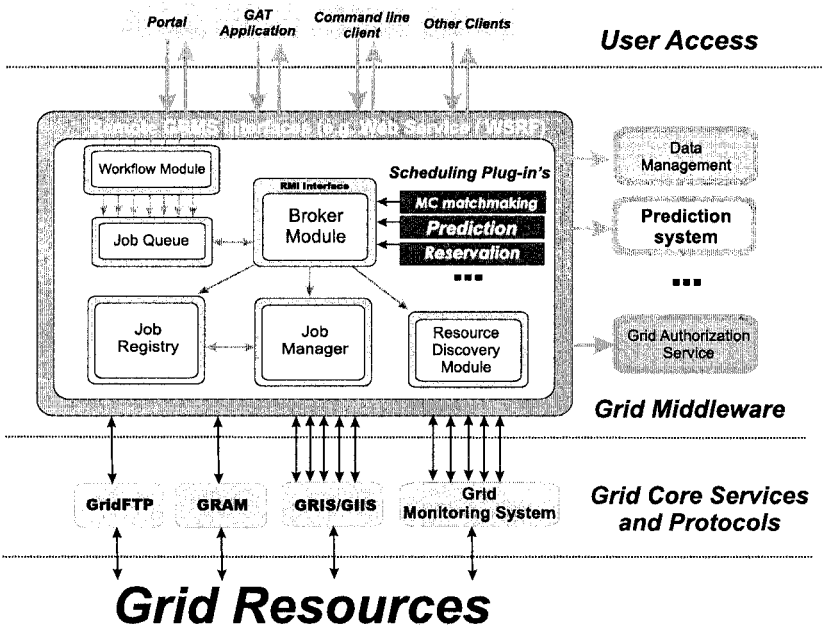

Figure 14.4. Detailed view of GRMS

job to, potentially remote computational resources, job migration, job canceling, logging, supporting workflow jobs.

Fig. 14.4 shows a more detailed view of GridLab GRMS with all its main modules and the Grid specific services, like *Replica Management*, *File Movement* and *Adaptive Components*.

As it is shown on fig. 14.4, GRMS consists of a set of various modules, including:

- **Broker Module**. The aim of the Broker Module is to control the whole process of resource and job management within the GRMS. This module steers a flow of requests to the GRMS and is also responsible for appropriate cooperation with other modules. Broker contains basic scheduling and policy strategies: matchmaking and multi-criteria matchmaking. The first strategy is a relatively simple but in fact very efficient approach for managing resources on which advanced reservation is not possible. The second strategy allows more flexible and accurate resource selections according to both users and administrator?s requirements and preferences. These two strategies can be easily modified and new scheduling and policy modules can be integrated as well.

- **Resource Discovery Module.** The Resource Discovery module monitors the status of distributed resources and therefore uses a flexible hierarchical

access to both central and local information services. This module uses various techniques to discover and get an efficient access to up-to-date and accurate (both static and dynamic) information about jobs and resources. The goal of Resource Discovery it to deliver all information in a form and on time required by the Broker and its scheduling and policy strategies.

- **Job Manager Module.** The Job Manager module is responsible for monitoring of job status changes within the GRMS and then storing information in a database together with many additional parameters including resource requirements of jobs, user names, job IDs, submission times, pending times, execution times, jobmanagers to which jobs were submitted, history of migration if jobs have been migrated, etc. Due to the importance of historic information, especially in multi-site or large scale resource management systems, the GRMS provides also the interface for users and administrator to receive information about past GRMS actives. The tracking of historic resource utilization for all users results in the ability to modify job priorities, ensuring a balanced access, and optimizing administrator criteria (e.g. job throughput or turnaround time).

- **Job Queue** All the user requests come into the Job Queue and wait processing. Jobs in the queue can be scheduled one by one, in a simplest case (first in first out), or in collections, i.e. a number of jobs, or all the jobs, are scheduled in parallel. The Job Queue can be distributed across various Grid domains to allow multi-domain scheduling. In such case each domain has its own Broker instance. All the Brokers can transfer jobs between all the domains. The overall system state is controlled by inter-broker communication mechanisms.

- **Job Registry** is responsible for maintaining the database of all jobs submitted to GRMS and all information concerning those jobs.

External to GRMS is a prediction system. The idea here was to be able to communicate with external prediction services, or systems and so far GRMS has been integrated with GPRES Prediction Expert System mentioned in previous section.

14.5.1 Resource Reservation in GRMS

All information related to time requirements of interactive jobs is passed to the system during the job submission process as a part of job description. Every job to be submitted can have an optional section that defines in a formal way the time requirements for the job to be computed. This gives a user the possibility to build descriptions of advanced execution schedule in a simple and flexible way. The "execution time" section consists of three subsections

```
<grmsjob appid = "interactive_example">
  <simplejob>
    <executable type="single" count="1">
      <file name="exec-file" type="in">
        <url>file:///${HOME}/interactive_test/interactive_exec</url>
      </file>
    </executable>
    <executionTime>
      <timeSlot>
        <slotStart>10:30:00</slotStart>
        <slotEnd>13:15:00</slotEnd>
      </timeSlot>
      <execDuration>P0Y0M0DT2H20M0S</execDuration>
      <timePeriod>
        <periodStart>2005-05-01T00:00:00-00:00</periodStart>
        <periodDuration>P0Y0M10DT0H0M0S</periodDuration>
          <excluding>
            <weekDay>Saturday</weekDay>
            <weekDay>Sunday</weekDay>
          </excluding>
      </timePeriod>
    </executionTime>
  </simplejob>
</grmsjob>
```

Figure 14.5. Example job description with time reservations

defining following requirements: optional slot within the day when a job must be executed, mandatory execution time and optional time period when a job must be executed. The slot within the day is specified by start time of the slot and optionally end time of it or time duration. Specifying this time slot a user can require that the job must be started after some time and not later then some other time of a day. Mandatory information concerning duration of the job execution determines length of the period when a resource reservation is needed for a job. It is the only time characteristic that can be changed by the user after the job was submitted. If it doesn't violate the schedule it is possible to extend the execution time of the previously submitted and running job. Planing the job execution a user can specify time period when a job must be executed. The presented job description (see fig. 14.6) illustrates usage of this functionality specifying liberal requirements that the job should be executed within the first ten days of May except Saturdays and Sundays.

Based on dynamic resource selection and discovery, mapping and advanced scheduling methodology, combined with a feedback control architecture and support from other Grid middleware services, it deals with dynamic Grid environments and resource management challenges, such as load-balancing among clusters and various work-load systems, remote job control, file staging, advance reservation, scheduling jobs with precedence relations etc.

One of the main requirements for GRMS development was to perform remote job control and management in the way that would satisfy job and resource owners in terms of their preferences. Therefore, GRMS implements multicrite-

ria procedures and optimization techniques to define and build various flexible resource management strategies.

14.5.2 Multicriteria Approach in GRMS

One of the most important modules of the broker is the multicriteria schedule evaluator (*MCEvaluator*). *MCEvaluator* implements various multicriteria models and tools that are applied to real Grid resource management and scheduling problems, including those with resource reservations and predictions. *MCEvaluator* is a framework that anyone can plugin into with new abstraction models. Main entities in GRMS include:

- Criteria (objectives, soft constraints)
- Hard constraints
- Solutions (e.g. resources, schedules etc. along with description parameters)
- Evaluator (decision point)

The framework contains also some multicriteria methods that GRMS uses for selection of best schedule. *MCEvaluator* provides support for a Grid scheduler to:

- Identify and select the best resource that a particular job will be computed on. In a workflow applications this process is repeated for every job being part of the workflow.
- Assign every available resource to predefined alternatives (classifying or sorting problem) or to order the alternative resource, as in ranking problem.
- Provide a performance tableau. In Grids, when the AI techniques are used it is often necessary to identify major distinguishing features of the resource or the whole schedule.

In order to solve the above mentioned tasks *MCEvaluator* can use many different methods, starting from outranking ELECTRE methods, through the utility functions aggregating the partial preferences on multiple criteria (MAUT, UTA, AHP) etc and finishing on the rules based methods. The features described above can be used for:

- Selecting the best resource to run a job on. This feature allows to choose best machine for a job, taking into account user preferences and host parameters, such as CPU load, total and free memory available for a job, number of CPUs, CPU speed, operating system etc.

- Select best queue at the remote resource to submit a job to. Potentially every resource is managed locally by the local resource management system (LRMS). LRMSs are usually based on queues that have different lengths and represent different policies of the resource owners. Selection of the best queue by *MCEvaluator* is based on the estimated job runtime and queue waiting time.
- Selection of the best job to be migrated. GRMS allows to use various dynamic strategies to manage jobs and resources, including job checkpointing and migration. By migrating a number of small jobs a Grid scheduler may allow to run bigger jobs on particular resource, which otherwise would have to wait longer in a queue. Cost of migration and resource characteristics are taken into account before decision is made.

Along with the *MCEvaluator* GRMS comes with a specialized multicriteria meta-language for expressing job descriptions and user preferences.

14.6 GRMS in Action

Knowledge acquired by the prediction techniques described in section 3 can be utilized in Grids, especially by resource brokers. Information concerning job run-times as well as a short-time future behavior of resources may be a significant factor in improving the scheduling decisions. A proposal of the multicriteria scheduling broker that takes the advantage of history-based prediction information is presented in this section. For our experimental considerations we have chosen the Minimum Completion Time algorithm, which is one of the simplest algorithms that require estimated job completion times. It assigns each job from a Job queue to resources that provide the earliest completion time for a particular job.

Nevertheless, apart from predicted times, the knowledge about potential prediction errors is needed. The knowledge coming from a prediction system shouldn't be limited only to the mean times of previously executed jobs which fit to a template. Therefore, we also consider minimum and maximum values, standard deviation, and estimated error. These parameters should be taken into account during a selection of the most suitable resources. Mean time stays as the most important criterion, however, relative importance of all parameters depends on user preferences and/or characteristics of applications. For instance, certain applications (or user needs) may be very sensitive to delays that can be caused by incorrectly estimated start and/or run times. In such case a standard deviation, minimum and maximum values become considerably important. Therefore, a multicriteria resource selection is needed to accurately handle these dependencies. In our case we used the functional model for aggregation of preferences. That means that we used a utility function and all

For each job J_i from a head of the queue

For each resource R_j, at which this job can be executed

Retrieve from the GPRES prediction system the estimated completion time of job C_{J_i,R_j}

Assign job J_i to resource R_{best} so that

$$C_{J_i,R_{best}} = \min_{R_j} \left(C_{J_i,R_j}\right)$$

Figure 14.6. Algorithm MCT (Abramson et al (2002))

$$F_{J_i,R_j} = \frac{1}{\sum_{k=1}^{n} w_k} \sum_{k=1}^{n} w_k * c_k \tag{14.6}$$

resources were ranked based on values of utility function. In detail, criteria are aggregated for job J_i and resource R_j by the weighted sum given according to formula (6).

where the set of criteria C (n=4) consists of the following metrics:
C_1 – mean completion time
C_2 – standard deviation of completion time
C_3 – difference between maximum and minimum values of completion time
C_4 – estimated error of previous predictions
and weights w_k that define the importance of the corresponding criteria. This method can be considered as a modification of the MCT algorithm to a multicriteria version. In this way possible errors and inaccuracy of estimations are taken into consideration in MCT. Instead of selection of a resource, at which a job completes earliest, the algorithm chooses resources characterized by the best values of the utility function F_{J_i,R_j}. As described above the function is calculated taking as an input values of four criteria: $time_{J_i,R_j}$, err_{J_i,R_j}, $stdev_{J_i,R_j}$, max_{J_i,R_j}, -min_{J_i,R_j}.

These two algorithms have been implemented in GRMS using its multicriteria selection framework of *MCEvaluator*.

For each job J_i from a head of the queue

For each resource R_j, at which this job can be executed

- Retrieve from the GPRES prediction system the estimated completion time of job C_{J_i,R_j} and err_{J_i,R_j}, $stdev_{J_i,R_j}$, max_{J_i,R_j}, min_{J_i,R_j}.
- Calculate the utility function F_{J_i,R_j}

Assign job J_i to resource R_{best} so that

$$F_{J_i,R_{best}} = \max_{R_j} \left(F_{J_i,R_j}\right)$$

Figure 14.7. Multicriteria MCT algorithm

14.6.1 Experiment

The system where the workload trace file was obtained from was a IBM SP2 System from Barcelona Supercomputing Center. The system, named Kadesh.cepba.upc.edu, was used with two different configurations: the IBM RS-6000 SP with 8*16 Nighthawk Power3 @375Mhz with 64 Gb RAM, and the IBM P630 9*4 p630 Power4 @1Ghz with 18 Gb RAM. A total of 336Gflops and 1.8TB of Hard Disk are available. All nodes are connected through an SP Switch2 operating at 500MB/sec. The operating system that they are running is an AIX 5.1 with the queue system Load Leveler. The workload was obtained from Load Leveler history files that contained around three years of job executions (178.183 jobs). Through the Load Leveler API, we converted the workload history files that were originally in a binary format. Analyzing the trace file we can see that total time for parallel jobs is approximately and order of magnitude bigger than the total time for sequential jobs, what means that in median they are consuming around 10 times more of CPU time. For both kind of jobs the dispersion of all the variables is considerable big, however in parallel jobs is also around an order of magnitude bigger. Parallel jobs are using around 72 times more memory than the sequential applications. In general these variables have significant amount of variability what may result in difficulties with predictions. In general users are not working with a big set of applications. In median, users submitted 9 different applications, and, also in median, they executed each application around 8 times. However, from the 98 users 22 of them had submitted in mean same applications more than 30 times. Taking into

account only these users, the presented median increases until 56.11 observations for user and application. Similar conclusions can be applied with user groups. Although same groups in general are submitting in median 22 different applications, they are still submitting few than 7 times the same application. However, there are some groups that are submitting in median more times same applications, from 22 groups, there are 6 groups that are submitting in median more than 42.2 times same applications.

We performed two major experiments. First, we compared results obtained by the MCT algorithm with a common approach based on the matchmaking technique (job was submitted to the first resource that met user?s requirements). In the second experiment, we studied improvement of results of the prediction-based resource evaluation after application of knowledge about possible prediction errors. For both experiments the following metrics were compared: mean, worst, and best job completion time. The worst and best job completion values were calculated in the following way. First, for each application the worst/best job completion times have been found. An average of these values was taken as the worst and best value for comparison. 5000 jobs from the workload were used to acquire knowledge by GPRES. Then 100 jobs from the workload were scheduled to Job queue ofr GRMS. The results of the comparison are presented in figure below. In general, it shows noticeable improvement of mean job completion times when the performance prediction method was used. The least enhancement was obtained for the best job completion times. The multi-criteria MCT algorithm turned out to be the most useful for improvement the worst completion times.

14.7 Conclusions

In this paper we elaborated on Grid job scheduling using Grid schedulers with resource reservation and prediction mechanisms. We also proposed the multi-criteria resource evaluation methods based on knowledge of job start- and run-times obtained from the prediction system. As a prediction system the GPRES tool was used. We exploited the method of multi-criteria evaluation of resources from GRMS. Resource reservation mechanisms were also used to make sure that resources are available at the moment of job staging. We presented how diverse end-users' requirements and preferences concerning time and cost can be modeled using multi-criteria decision support techniques. Thanks to this approach end-users can express both hard constraints that must be satisfied and soft constraints that help a resource broker to find the best offers of resource providers. Furthermore, we showed how preferences of multiple end-users can be aggregated in order to find a compromise schedule. The hypotheses assumed in the paper have been verified. Exploitation of the knowledge about performance prediction allowed a resource broker to make more

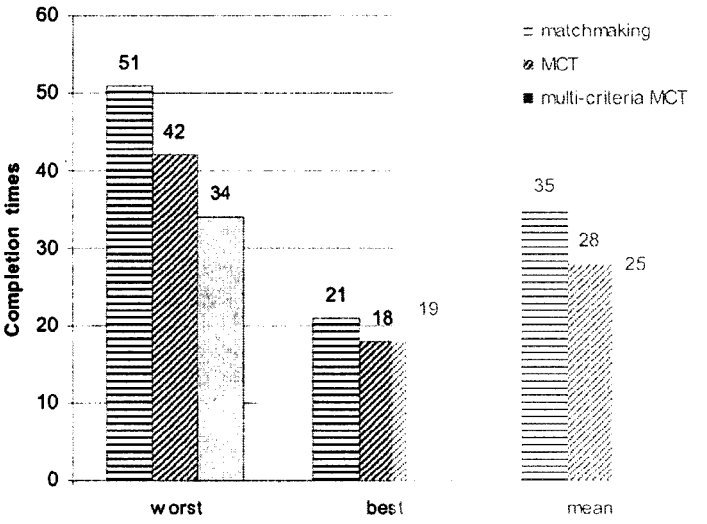

Figure 14.8. Comparison of job completion times for matchmaking, MCT, and multi-criteria MCT algorithms

efficient decisions. This was visible especially for mean values of job completion times. Exploitation of knowledge about possible prediction errors brought another improvement of results. As we had supposed it improved mainly the worst job completion times. Thus, taking the advantage of knowledge about prediction errors we can limit number of job completion times that are significantly worse than estimated values. Moreover, we can tune the system by setting appropriate criteria weights depending on how reliable results we need and how sensitive to delays the application are. For instance, certain users may accept 'risky' resources (i.e. only the mean job completion time is important for them) while others may expect certain reliability (i.e. low ratio of strongly delayed jobs). The preliminary results show that using prediction information and resource reservation can bring significant results, while scheduling jobs in Grid environments. Of course there are many limitations to apply the approach in open Grid systems, but, for many users and jobs, which run frequently in particular infrastructure the results may be impressive. In general, use of resource reservation and performance prediction mechanisms in Grids may help to improve performance by means of better Grid resource broker decisions (based on more accurate knowledge) and possibility of scheduling multiple jobs at once. Moreover, in certain scenarios in which QoS must be provided this approach is indispensable.

Nevertheless, there are some drawbacks and problems that must be taken into account when these mechanisms are used. First of all, an extensive use of resource reservation can deteriorate the overall job throughput. This unfavorable influence can be limited by use of accurate job execution time predictions (along with estimated imprecision) and resource providers policies that allow starting jobs earlier than their reserved start time. Convenient ratio of numbers of jobs with and without reservations is also a major factor that influences performance. The exact dependencies between these issues are a subject of further research. Another important issue is a need of additional steps to obtain offers of resource providers and estimations from a prediction system. These steps can also increase response time. Additionally, since we cannot assume a control of resource broker (and in consequence a prediction system) over local resources the prediction is more difficult due to limited information about resources. To solve this problem very advanced Grid monitoring systems need to be introduced.

Acknowledgments

The authors of this paper would like to thank Agnieszka Kwiecien, Maciej Dyczkowski and Marcin Wojtkiewicz from Wroclaw Networking and Supercomputing Center for integrating their GPRES systems with GRMS and making it available for the tests. We also wanted to thank Jesus Labarta from Barcelona Supercomputing Center for giving access to historical workloads of their machines.

References

Abramson, D., Buyya, R. and Giddy, J. (2002). A computational economy for Grid computing and its implementation in the Nimrod-G resource broker, *Future Generation Computer Systems*, 18(8):1061–1074.

Agrawal, R. and Srikant, R. (1994). Fast Algorithms for Mining Association Rules, in: *Proceedings of the Twentieth Intl. Conference on Very Large Databases*, Morgan Kaufmann, pp. 487–499.

Allen, G., Davis, K., Dolkas, K.N., Doulamis, N.D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J. and Taylor, I. (2003). Enabling Applications on the Grid - A GridLab Overview, *International Journal of High Performance Computing Applications*, 17(4):449–466.

Bode, B., Kendall, D.M. and Lei, Z. (2000). The Portable Batch Scheduler and the Maui scheduler on Linux clusters, in: *Proceedings of 4th Annual Linux Showcase and Conference*, October 2000.

Černy, V. (1985). Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm, *Journal of Optimization Theory and Applications*, 45:41–51.

Cheung, L.S. (2001). A Fuzzy Approach to Load Balancing in a Distributed Object Computing Network, in: *Proceedings of the First IEEE International Symposium of Cluster Computing and the Grid (CCGrid'01)*, pp. 694–699.

Condor Group, *Condor project*, http://www.cs.wisc.edu/condor.

Czajkowski, K., Foster, I., Kesselman, C., Martin, S., Smith, W. and Tuecke, S. (1997). A resource management architecture for metacomputing systems, *JSSPP Whorskshop, Lecture Notes on Computer Science*, 1459:62–68.

Dail, H. (2001). A Modular Framework for Adaptive Scheduling in Grid Application Development Environments, Technical report CS2002-0698, Computer Science Department, University of California, San Diego.

Darken, C. and Moody, J. (1990). Fast Adaptive k-means clustering: Some empirical results, in: *Proceedings of the International Joint Conference on Neural Networks*, vol. II, IEEE Neural Networks Council, pp. 233–238.

Dinda, P. (2001). Online prediction of the running time of tasks, in: *Proceedings of 10th IEEE Symp. on High Performance Distributed Computing*, pp. 336–337.

Downey, A. (1997). Predicting Queue Times on Space-Sharing Parallel Computers, in: *11th International Parallel Processing Symposium*, pp. 209–218.

Global Grid Forum DRMAA WG, DRMAA Web Site, http://www.drmaa.org.

European DataGrid Project, http://www.eu-datagrid.org.

El-Ghazawi, T., Gaj, K., Alexandridis, N., Vroman, F., Nguyen, N., Radzikowski, J.R., Samipagdi, P. and Suboh, S.A. (2004). A performance study of job management systems, *Concurrency and Computation: Practice and Experience*, 16(13):1229–1246.

Feitelson, D.G. and Mu'alem Weil, A. (1998). Utilization and predictability in sche-duling the IBM SP2 with backfilling, *Proceedings of 12th International Parallel Processing Symp.*, Orlando, pp. 542–546.

Feitelson, D.G., Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/work-load.

Figuiera, S.M. and Bermann, F. (2001). Mapping Parallel Applications to Distributed Heterogeneous Systems, Technical report CS2002-0698, Computer Science Department, University of California, San Diego.

Foster, I. and Kesselman, C. (1998). The Globus Project: A Status Report, in: *Proceedings of the Seventh Heterogeneous Computing Workshop*, pp. 4–18.

Foster, I. and Kesselman, C. (editors) (1999). *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kauffmann, San Francisco, California.

Foster, I. and Kesselman, C. (1999). Computational Grids, in: *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman, eds, Morgan Kaufmann, San Francisco, California, pp. 15–52.

Gibbons, R. (1997). A Historical Application Profiler for Use by Parallel Schedulers, *Lecture Notes on Computer Science*, 1297:58–75.

Globus Team, Globus Project, http://www.globus.org.

Glover, F. (1989). Tabu Search - part 1, *ORSA Journal of Computing*, 1(3):190–206.

Glover, F. (1990). Tabu Search - part 2, *ORSA Journal of Computing*, 2:4–32.

Glover, F. (1986). Future Path for Integer Programming and Links to Artificial Intelligence, *Computers & Operations Research*, 13:533–549.

Goldberg, D.E., (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading.

Greco, S., Matarazzo, B., Slowinski, R. and Tsoukias, A. (1998). Exploitation of a rough approximation of the outranking relation in multicriteria choice and ranking, in: *Trends in Multi-Criteria Decision Making*, T.J Stewart and R.C van der Honert, eds, Springer Verlag, Berlin, pp. 45–60.

Greco, S., Matarazzo, S. and Slowinski, R. (2001). Rough sets theory for multicriteria decision analysis, *European Journal of Operational Research*, 129(1):1–47.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press.

Ishibushi, H. and Murata, T. (1998). A Multi-Objective Genetic Local Search Algorithm and Its Application to Flowshop Scheduling, *IEEE Transactions on Systems, Man and Cybernetics*, 28(3):392–403.

Jackson, D.B., Maui Admin Guide, http://supercluster.org/maui/docs/mauiadmin.html.

Jaszkiewicz, A. (1998). Genetic local search for multiple objective combinatorial optimisation, Technical Report RA014 /98, Institute of Computing Science, Poznan University of Technology.

Kirkpatrick, S., Gelatt, C.D., Jr and Vecchi, M.P. (1983)., Optimization by Simulated Annealing, *Science*, 230:671–680.

Knowles, J.D. and Corne, D.W. (2000). A Comparison of Diverse Approaches to Memetic Multiobjective Combinatorial Optimization, in: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), Workshop On Memetic Algorithms*, pp. 103–108.

Knowles, J.D. and Corne, D.W. (2000). M-PAES: A Memetic Algorithm for Multiobjective Optimization, in: *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 325–332.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2000). Multicriteria Resource Management Architecture for Grid, in: *Proceedings of the 4th Globus Retreat*, Pittsburgh, PA, July 2000.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2000). Predicting Job Execution Times in the Grid, in: *Proceedings of the 1st SGI 2000 International User Conference*, Kraków, pp. 272–282.

Kurowski, K., Nabrzyski, J. and Pukacki, J. (2001). User preference driven multiobjective resource management in Grid environments, in: *Proceedings of the First IEEE International Symposium on Cluster Computing and the Grid (CCGrid'01)*, pp. 114-121.

Kurowski, K., Nabrzyski, J., Oleksiak, A. and Węglarz, J. (2003). Multicriteria Aspects of Grid Resource Management, in: *Grid Resource Management*, J. Nabrzyski, J. Schopf, and J. Węglarz, eds, Kluwer Academic Publishers, Boston/Dordrecht/London, pp. 271–294.

Kurowski, K., Ludwiczak, B., Nabrzyski, J., Oleksiak, A. and Pukacki, J. (2004). Improving Grid Level Throughput Using Job Migration and Rescheduling Techniques in GRMS, *Scientific Programming*, 12:(4)263-273.

Kurowski, K., Oleksiak, A., Nabrzyski, J., Guim, F., Corbalan, J., Labarta, J., Kwiecien, A., Wojtkiewicz, M. and Dyczkowski, M. (2005). Multicriteria Grid Resource Management using Performance Prediction Techniques, in: *Proceedings of the 2nd CoreGrid Workshop*, Springer Verlag (to appear).

Langley, P., Iba, W. and Thompson, K. (1992). in: *An Analysis of Bayesian Classifiers, Proceedings of AAAI-92*, pp. 223-228.

Lifka, D. (1995). The ANL/IBM SP scheduling system, in: *Job Scheduling Strategies for Parallel Processing*, D.G. Feitelson and L. Rudolph, eds, Springer-Verlag, Lecture Notes of Computer Science, 949:295–303.

Liu, C., Yang, L., Foster, I. and Angulo, D. (2002). Design and evaluation of a resource selection framework for Grid applications, in: *Proceedings if the Eleventh IEEE International Symposium on High-Performance Distributed Computing (HPDC–11)*, pp. 63–72.

Nabrzyski, J., Schopf, J. and Węglarz, J., editors, (2003). *Grid Resource Management - State of the Art and Future Trends*, Kluwer Academic Publishers.

Nabrzyski, J. (2000). User Preference Driven Expert System for Solving Multiobjective Project Scheduling Problems, Ph.D. Thesis, Poznan University of Technology.

Pawlak, Z. (1982). Rough Sets, *International Journal of Information & Computer Sciences*, 11(5):341–356.

Platform Computing Technical Docs, http://www.platform.com/services/support/docs/LSFDoc51.asp.

Quinlan, J.R. (1986), Induction of Decision Trees, *Machine Learning*, 1:81–106.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning Representations by Back Propagating Errors, *Nature*, 323:533–536.

Sandholm, T.W. (1999). Distributed Rational Decision Making, in: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, ed, MIT Press, pp. 201–258.

Schopf, J. and Berman, F. (1998). Performance prediction in production environments, in: *Proceedings of IPPS/SPDP*, pp. 647–653.

Shirazi, B.A., Husson, A.R. and Kavi, K.M. (1995). *Scheduling and Load Balancing in Parallel and Distributed Systems*, IEEE Computer Society Press.

Smith, W., Taylor, V. and Foster, I. (1999), Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance, *Proceedings of the IPPS/SPDP '99 Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 202–219.

Taylor V., Wu, X., Geisler, J., Li, X., Lan, Z., Hereld, M., Judson, R. and Stevens, R. (2001). Prophesy: Automating the modeling process, in: *Proceedings Of the Third International Workshop on Active Middleware Services.*

Veridian Inc. PBS: The Portable Batch System. http://www.openpbs.org/

Vazhkudai, S. and Schopf, J. (2003). Using Regression Techniques to Predict Large Data Transfers, *Journal of High Performance Computing Applications - Special Issue on Grid Computing: Infrastructure and Application*, 17: 249-268.

Węglarz, J., editor (1999). *Project Scheduling - Recent Models, Algorithms and Applications*, Kluwer Academic Publishers.

Wolski, R., Spring, N. and Hayes, J. (1999). The Network Weather Service: a distributed resource performance forecasting service for metacomputing, *Future Generation Computer Systems*, 15(5–6):757–768.

Wolski, R. (1997). Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service, *Cluster Computing*, 1(1):119-132.

Zadeh, L.A. (1965), Fuzzy Sets, *Information and Control*, 8(3):338–353.