**RC23597 (W0504-144) April 29, 2005**
**Computer Science**

# IBM Research Report

## On-line Resource Matching for Heterogeneous Grid Environments

**Vijay K. Naik**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

**Chuang Liu, Lingyun Yang**
Computer Science Department
University of Chicago
Chicago, IL 60637

**Jonathan Wagner**
IBM Software Group
Tivoli
Austin, TX 78758

**Research Division**
**Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich**

# On-line Resource Matching for Heterogeneous Grid Environments

Vijay K. Naik
*IBM T. J. Watson Res Center*
*Yorktown Heights, NY 10598*
*vkn@us.ibm.com*

Chuang Liu Lingyun Yang
*Computer Science Dept, University of Chicago*
*Chicago, IL 60637*
*{chliu, lyang}@cs.uchicago.edu*

Jonathan Wagner
*IBM Software Group, Tivoli*
*Austin, TX 78758*
*jonathaw@us.ibm.com*

## Abstract

*In this paper, we first present a linear programming based approach for modeling and solving the resource matching problem in grid environments with heterogeneous resources. The resource matching problem described here takes into account resource sharing, job priorities, dependencies on multiple resource types, and resource specific policies. We then propose a web service style architecture for on-line matching of independent jobs with resources in a grid environment and describe a prototype implementation. Our preliminary performance results indicate that the linear programming based approach for resource matching is efficient in speed and accuracy and can keep up with high job arrival rates – an important criterion for on-line resource matching systems. Also, the web service style architecture makes the system scalable and extendable. It can also be integrated with other existing grid services in a straightforward manner.*

## 1. Introduction

As grid environments become more popular, the size and the diversity in terms of resources and jobs are increasing as well. By their nature, individual resources are not under central control and they can enter and leave the grid system at any time. Similarly, job arrivals and job mix are much less predictable than in a dedicated parallel or cluster environment. Because of this, dynamically matching jobs with available resources plays an important role in the management of a grid environment. The resource matching mechanism must take into account dynamically changing conditions – both in terms of demand on resources and available supply of resource capacity. It must also accommodate resource specific and system wide policies.

In this paper, we consider the resource matching problem arising in grid environments where jobs require multiple heterogeneous resources and system managers expect resource sharing, load balancing, high resource utilization and/or throughput. The problem of matching independent jobs to heterogeneous resources is known to be NP-complete, if maximizing throughput is the optimization criteria [4]. Other optimization criteria such as load balancing, maximizing utilization, or minimizing resource cost also give rise to the same level of difficulty. While the problem is hard, for grid environments, fast efficient on-line resource matching algorithms that can quickly adapt to changes in the system are highly desirable. In this paper, we discuss a class of such algorithms that are well suited for this type of environment.

The contributions of this paper are as follows. We describe a novel approach to model the resource matching problem applicable to a grid environment consisting of heterogeneous resources. Using our approach, we can handle a rich class of job resource requirements, resource preferences or affinities, job priorities, capacity constraints, resource sharing, and a class of resource specific policies. Our approach allows resource sharing in the presence of on-line and asynchronous job arrivals. A second contribution of this paper is the architecture and a prototype implementation of a resource matching and allocation system suitable for large-scale grid environments.

We model resource matching as an Integer-Programming (IP) problem and apply integer-programming techniques to solve the resulting optimization problem. Previous work [12, 13] illustrates the benefits of using IP techniques to solve job-scheduling problems on dedicated resources of a single type (such as compute resources). However, grid environments present some unique challenges. These include:

- *Resources in a grid environment are often shared among multiple concurrent jobs.* Thus, instead of matching just one job with multiple resources all dedicated to run the job as in [8], general grid resource matching mechanisms must be able to match multiple jobs to a single resource, provided the resource has enough capacity to handle the matched jobs simultaneously. On the other hand, resource matching must take into account resource usage by each matched job so the capacity consumed by the matched jobs does not to exceed

the total resource capacity. We call this the *resource capacity-constraint* rule.

- *A grid job usually needs multiple types of resources to run.* A match is successful only if all required resources meeting the minimum capacity constraints are found. The match is unsuccessful even if a single resource cannot satisfy one of the constraints. Thus, a resource matcher should return either all required resources or none. We define this requirement as the *gang matching constraint.*
- *Jobs may arrive continuously and compete for resources.* Instead of matching jobs one by one to resources, we need to consider a batch of jobs, their requirements and provide a globally optimal matching scheme. For example, if resources are not enough to satisfy all jobs, we may want to match as many jobs as possible and give advantage to high priority jobs, or if the usage of multiple resources results in higher cost, it may be more desirable to match jobs with as few resources as possible. We call the metric used to measure the quality of a matching scheme as the *business-value objective function*. The matching process needs to maximize/minimize the objective function to achieve a globally optimal matching, in an on going continuous manner.
- *In a grid environment jobs arrive and depart asynchronously.* Thus, the degree of capacity sharing on a given resource varies continuously and it is not the same across multiple resources. Modeling the resource capacity constraints is more difficult because of the asynchronous resource sharing among multiple jobs.
- *Jobs and resources in a grid are heterogeneous*. Grid enables aggregation of different types of resources; e.g., a grid system may consist of compute servers, network resources, storage resources, application servers, etc. Similarly, a grid job may require more than one type of resource and these requirements vary from job to job. Because of this, typical approaches based on greedy algorithms do not work well. IP techniques provide means for expressing the global abstractions succinctly. However, modeling the resource-matching problem as an IP problem is much more challenging because of the heterogeneity across jobs and resources.

In the following sections, we describe how we handle these challenges. In Section 2, we discuss the scope of resource matching problem considered in this paper and its practical significance. In Section 3, we describe our approach to model the resource matching problem as a mixed integer linear program (MILP). In Section 4, we describe an architecture for on-line resource matching for jobs in a heterogeneous grid environment. We discuss a proof-of-concept implementation of a prototype and the preliminary performance results in Section 5. We discuss the related work in Section 6 and conclude the paper in Section 7.

## 2. Problem statement

In this paper, we consider the problem of matching jobs with available resources taking into account job resource requirements, job specific resource preferences, job priorities, resource capacities, resource specific policies, and system wide global objectives.

The workload considered here consists of a set of independent jobs with jobs arriving with some unknown arrival pattern. A job consists of a request, an executable, and some input data (or references to these). The request specifies the requirements and preferences to run the job. For simplicity, in this paper, a job means both the request and the actual executable. Priority is assigned to a job, when it enters the system.

Resource types are defined to classify all the resources in the system. Some common examples of resources types are computer system, network subsystem, file system, application, and database system. Each resource type is associated with one or more attributes with specific values. Examples of attributes of a computer system are CPU architecture, total and available memory, maximum and current degree of multi-programming, and so on. Thus, each resource instance in the system has a resource type and has specific values for the attributes of that resource type. Some attribute values are invariants while others change their values as jobs are run using the resource. In addition to attributes, a resource may also be associated with policies specifying preferences and usage permissions by job class. The policies can be dynamic as a function of time and/or jobs running on that resource. In this paper, we permit resource sharing by multiple jobs to the extent it is possible with the existing resource capacity and current policy associated with the resource.

The job requirements consist of a set of dependencies and constraints associated with each dependency. Each requirement is a dependency on a resource instance of a particular resource type. By associating constraints with each dependency, a job may specify the minimum criteria for selecting a resource instance of a particular resource type. For example, a request with a dependency on a computer system may specify minimum 400 MB available memory as a constraint. For the jobs considered in this paper, we restrict one dependency per resource type, but we do not have any restrictions on the number of different resource types a job may depend upon or the number of constraints a job may specify per resource type, as long as the constraints are independent.

In addition to the hard constraints for selecting resources, a job may also specify resource preferences

among the qualifying resource instances. For example, a job may request a computer system with minimum 400 MB available memory and preference to the one with the highest CPU speed.

Finally, system administrators may want to enforce some global objectives to achieve certain business goals. Some examples of global objectives are maximizing resource utilization using the smallest number of resources, maximizing throughput, allocating best resources to high priority jobs, balancing load among fixed number of resources, and so on.

In summary, in this paper, we are addressing a general resource-matching problem, which we define as the process of systematically selecting resources while conforming to job and resource specific requirements and constraints as well as optimizing site-wide objectives. However, there are some restrictions: (1) the workload considered here consists of independent jobs. Each job has at most one dependency per resource type. (2) Jobs are treated as non-preemptive. (3) Also not considered here is the issue of fairness. Even if necessary resources are available, the resource matcher may not match the resources with a job, but instead it may match the resources with one or more other jobs. This may repeat indefinitely depending on the workload and resource availability. We have extended our work to take into account jobs requiring multiple instances of a single type of resource and the problem of avoiding job starvation. We discuss this work elsewhere.

## 3. Resource matching as an IP problem

Integer programming is a technique for solving certain kinds of problems: maximizing the value of an objective function subject to constraints, where the objective function and constraints are all linear expressions. In the following, we describe models based on mixed-integer programming techniques and discuss how these models can be used to overcome the grid specific challenges, discussed in the introductory section, in solving the resource matching problem.

### 3.1. Basic model

An IP model includes input parameters, variables, a set of constraints on the value of variables, and an objective function. The goal of the model is to find values for every variable such that all constraints are satisfied and the value of the objective function is maximized/minimized.

Figure 1 describes our basic model for the resource matching problem. Given a set of jobs and resources, this model finds a best feasible match between the given set of jobs and resource.

**Input parameters:**

1. J is the set of jobs to be matched.
2. R is the set of resources available.
3. $P_j$ is the priority of job j, for each $j \in J$.
4. $N_{(r, t)}$ is the capacity of type t for resource r, $r \in R$. A resource may be associated with multiple types of resource capacities.
5. $U_{(j, t)}$ is the amount of resource capacity of type t consumed by job j, for each $j \in J$.

**Output variables:**

6. $X_{(j, r)}$ is 0/1 variable, for each resource $r \in R$ and job $j \in J$. $X_{(j, r)}$ is equal to 1 if job j is matched to resource r, and is equal to 0 otherwise.
7. $Z_j$ is 0/1 variable, for each job $j \in J$. $Z_j$ is equal to 1 if job j is matched to its required resources, and is equal to 0 otherwise.

**Maximize/Minimize:**

8. An objective function such as the number of matched jobs.

**Subject to:**

9. Resource capacity constraints on resource r, for each $r \in R$
10. Gang matching constraints on job j, for each $j \in J$
11. Jobs' requirements on satisfying resources.

**Figure 1. Basic model for resource matching**

More specifically, we describe a job j by its priority (line 3) and the amount of resource capacity it is going to consume (line 5). We describe every resource r by its available capacity (line 4). A resource may have multiple capacity limitations, for example, a machine may allow at most 6 jobs to run at the same time and provide at most 1 GBytes memory in total. Thus, we describe resource capacity of resource r by $N_{(r, t)}$ where t indicates the type of capacity. In the same way, we describe the resource consumption of a job j by $U_{(j, t)}$.

We define a set of variables to represent a matching scheme in lines 6-7. The value of variable $X_{(j, r)}$ indicates if a job j is matched to a resource r, and the value of $Z_j$ indicates if a job j is matched to required resources. The search for a matching scheme is converted to the search for feasible variable values such that the objective function is maximized or minimized.

Constraints in line 9-11 define a feasible matching scheme. Because there may exist multiple feasible matching solutions, we use the objective function in line 8 to measure the quality of a scheme.

The model in Figure 1 can be used to represent a wide range of grid resource matching problems. If we instantiate the values of job and resource parameters, this model becomes a particular resource-matching problem. In order to solve it using IP techniques, we need to formalize

the basic model in Figure 1 by linear expressions and constraints.

### 3.2. Dependency initialization

To run successfully, a job may require multiple resources each with particular characteristics and capacity. We refer to the job requirement on a resource type as a *dependency*. Thus, a job may have multiple dependencies.

Many resources may satisfy the requirements of a dependency. We define these resources as *equivalence set* for this dependency. For convenience, we represent the equivalence set for dependency d of job j as $E_{(j,\ d)}$. Resources in an equivalence set are feasible resources and are candidate matches for that resource type.

As the first step of the IP modeling, we classify available resources into equivalence sets based on job requirements. However, we cannot pick a random resource and assign it to this job because we also need to satisfy the gang match constraints (described in Section 3.3) and resource capacity constraints (described in Section 3.4) for a batch of jobs, and find a globally optimal solution (described in Section 3.5).

### 3.3. Gang matching constraints

Gang match constraints ensure that, for a job requiring multiple resources, we match all required resources to this job, or none. We formalize the gang match constraint for each job j as a set of linear equations:

$$\sum_{r \in E_{(j,d)}} X_{(j,r)} = Z_j ,$$

for each dependency d of each j ∈ J

Note that $Z_j$, is equal to 1 if job j is matched with all required resources, and is equal to 0 otherwise. As defined in Figure 1, $X_{(j,\ r)}$ is equal to 1 if resource r is matched to job j, and is equal to 0, otherwise. Thus, the left side of the equation, the sum of $X_{(j,\ r)}$ over resources in an equivalence $E_{(j,\ d)}$, is the number of resource matched to the dependency d. Thus, these equations guarantee that for every dependency of job j, exactly one resource is matched when this job is matched with all required resources and none of the dependencies is matched if any one of the dependencies is unsatisfied.

### 3.4. Resource capacity constraints

Resource capacity constraints ensure that the sum of the fraction of the capacity consumed by all supported jobs by a resource does not exceed the total available capacity.

Given a resource r and a job j, job j will consume $U_{(j,\ t)}$ unit of capacity type t of resource r if this job is matched to this resource; and consume 0 otherwise. Thus, the resource consumption of a job on a resource can be expressed by a linear expression $U_{(j,\ t)} * X_{(j,\ r)}$. The resource capacity constraints can be expressed by the following set of inequalities:

$$\sum_{j \in J} U_{(j,t)} * X_{(j,r)} \le N_{(r,t)} ,$$

for each r ∈ R and its capacity t.

This set of inequalities express resource capacity constraint on all resources.

### 3.5. Objective functions

The objective function defines the quality of a matching scheme when multiple feasible solutions exist. The matching algorithm uses objective function to select the best matching scheme. We model four types of objective functions.

**Throughput**. This objective function is defined as the number of jobs matched. When maximizing the value of this objection function, the matching process will find a matching scheme that matches as many jobs as possible. We formalize the throughput as a linear expression $\sum_{j \in J} Z_j$ .

When jobs have different priority, we use a weighted throughput, defined as $\sum_{j \in J} Z_j P_j$ . In this way, we give more advantage to jobs with higher priority.

**Number of resources used.** This objective function is defined as the number of resources used to match jobs. In some cases, users may want to run a batch of jobs on as few resources as possible. For example, if resource owners charge users for the use of resources, the user may want to limit jobs to a small set of resources to reduce the cost. Minimizing the value of this objective function, the matching process will find scheme using as few resources as possible.

In order to calculate the number of resources used, we define a new 0/1 variable $S_r$, which is equal to 0 if resource r is not matched to any one of jobs, and is equal to 1 otherwise, for each resource r∈R by the following two sets of linear inequalities:

$$\sum_{j \in J} X_{(j,r)} \ge S_r$$

and

$$\sum_{j \in J} X_{(j,r)} \le S_r * Card(J),$$

for each resource r∈ R, where Card(J) is the number of jobs considered.

These inequalities ensure that $S_r$ behaves as defined.. For a resource r, the sum of $X_{(j,\ r)}$ over jobs is equal to the number of jobs matched to this resource. If no job is matched with this resource, its value is equal to 0. In this case, based on the first inequality, $S_r$ is equal to 0 because $S_r$ must be less than or equal to 0 to make the inequality stand. On the other hand, if some jobs get matched to this

resource, the sum of $X_{(j, r)}$ over all jobs is greater than or equal to 1. Based on the second inequality, $S_r$ is equal to 1 because $S_r$ must be greater than 0 to satisfy the inequality.

Then, we can formalize the number of used resource as a linear expression, $\sum_{r \in R} S_r$ .

**Load balance.** This objective function is defined as the percentage of unused capacity on the most heavily loaded resources. Trying to maximize the value of this objective function, the matching process will move jobs from the most heavily loaded resources to less loaded resources so as to achieve a load distribution as even as possible.

We define a new variable $G_{(r, t)}$ for each resource $r \in R$ to represent the percentage of used capacity type t in this recourse. We modify the linear inequalities for resource capacity constraint in Section 3.4 to calculate the value of variable $G_{(r, t)}$.

$$\sum_{j \in J} U_{(j,t)} * X_{(j,r)} = N_{(r,t)} * G_{(r,t)},$$

$$G_{(r,t)} \geq 0,\ G_{(r,t)} \leq 1$$

for each resource $r \in R$ and its capacity type t

As described in Section 3.4, the left side of the equation is the amount of capacity used by all jobs, and $N_{(r, t)}$ is the total available capacity of the resource. Thus, $G_{(r, t)}$ is the used percentage.

We then define another new variable F to represent the percentage of unused capacity on the most heavily loaded resources by the following set of inequalities:

$$F \leq 1 - G_{(r,t)}$$

for each resource $r \in R$ and its capacity type t.

The right side of the inequality is the percentage of unused capacity of a resource r and its capacity type t. Since the most heavily loaded resources have the least percentage of unused capacity, F is less than or equal to the right side.

**Job preferences on resources.** This objective function is defined as the sum of ranks of all resources matched to jobs. A job may provide a criterion to rank all satisfying resources. For example, a job may prefer a machine with higher CPU speed and use CPU speed as a ranking criterion. We can formalize this objective function by a linear expression:

$$\sum_{\substack{j \in J, \\ r \in R}} Q_{(j,r)} * X_{(j,r)},$$

where $Q_{(j, r)}$ is the given rank of resource r by job j.

Maximizing this objective function, the matching process will find a matching scheme containing as many preferred resources as possible.

By using these four objective functions, users can configure the matching process to find different optimal matching scheme. Using MILP techniques, we show that it is possible to realize different resource matching processes each configured to optimize a specific business objective. System administrators can easily modify these objective functions or create their own objective function to adapt to their specific needs.

## 4. Architecture of an on-line resource matching system

We now describe an on-line resource matching system that accepts multiple job requests in a batch, models the matching problem as a MILP problem, and then generates a matching scheme that is optimal based on a given objective function. The architecture of the matching system is shown in Figure 2.

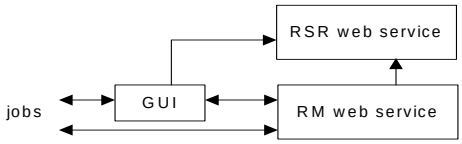


**Figure 2. Architecture of the on-line resource matching system**

The on-line resource matching system consists of three functional modules, with each implemented as a web service.

**Resource State Repository (RSR)**. Resource State Repository stores soft real time status of all resources. Our current implementation of RSR is shown in Figure 3. It uses IBM DB2 to store the up-to-date status of resources and organizes resource data based on resource types. RSR provides a web service interface to allow resources to update their status, clients to query resource status, and administrators to manage the resource data, such as defining new resource types, adding/deleting resources, etc.

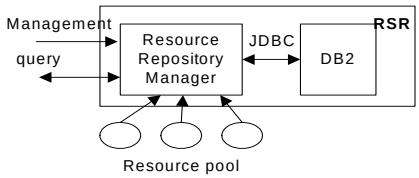


**Figure 3. Structure of Resource State Repository**

**Resource Matcher (RM)**. Resource Matcher is the decision making part of the matching system. Its structure is shown in Figure 4. RM performs following functions: (1) it accepts job matching requests through it web service interface, (2) queries the Resource State Repository to obtain current resource status, (3) instantiates the model created in Section 3 with the actual job and resource data, (4) submits the model to a MILP solver to solve for the variables in the model, (5) maps the variable values to a matching scheme, and (6) sends the matching results back to clients.

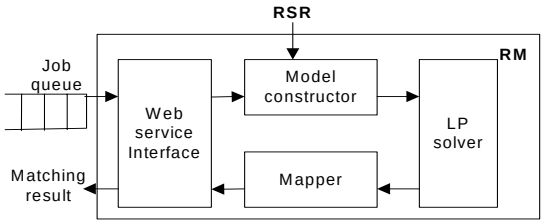


**Figure 4. Structure of Resource Matcher**

In our implementation, we use a standard modeling language AMPL [3] to present the model we propose in Section 3, and use CPLEX solver [1] to solve it. Because the output of the solver is a set of 0/1 variables, we use a mapper to translate the variables to more meaningful values of jobs and resources. For example, if $X_{(j, r)}$ is equal to 1, we translate it as resource r is matched with job j.

**GUI interface.** The GUI interface provides a friendly web interface to help users to submit jobs, configure the objective function in Resource Matcher, and manage the Resource State Repository.

Since the three functional modules are implemented as web services, they can work as standalone services or can easily be integrated with other services. Also, the architecture is not limited to the current implementation. For example, we can replace our RSR by other information systems such as MDS [10] and Ganglia [11] etc. by simply wrapping their interface in a web service defined by Resource State Repository.

## 5. Performance results and discussion

To validate our architecture and design, we have implemented a prototype version of the Resource Matcher, Resource State Repository, and the GUI. In this section, we describe our prototype implementation and present some preliminary performance results. The experimental set up is shown in Figure 5.

Resource Matcher and Resource State Repository are deployed as web services on an IBM WebSphere Application Server (Version 5.1) [14]. The application server itself is installed on a Windows 2000 server. GUI interfaces for administrative tasks and for job submission purposes are defined using java server pages (JSPs) and are deployed on the same application server. Users can submit job requests using a web browser, or using a standalone application client that interacts directly with the Resource Matcher using its web services interface. The database for the repository is installed on an IBM DB2 Server (DB2 UDB version 8.1.4) running on a separate Windows 2000 server [15]. The Resource State Repository web service interacts with the repository on the DB2 using JDBC protocol.

In our experiments, Resource Matcher is configured to use throughput as its objective function. The client is a java program that can run on any system supporting JDK 1.4 and that has network access. In our experiments, we used an IBM Thinkpad T23 to submit jobs to Resource Matcher using the RM client.

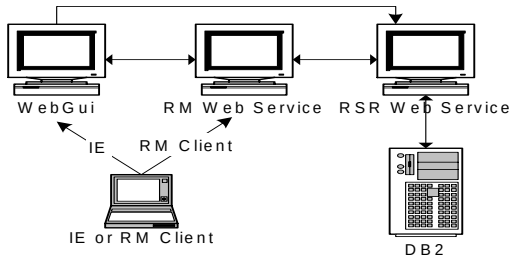


**Figure 5. Deployment of the online matching system.**

We simulated several resources in the Resource State Repository. For the experiments reported in the following, we defined four resource types: (i) compute server (attributes: network location, OS version, CPU type, CPU speed, memory, maximum and available application capacity), (ii) file system (attributes: network location, type, size, maximum and available capacity), (iii) database (attributes: network location, type, maximum and available number of connections), and (iv) network subsystems (attributes: network domain, type, maximum and available bandwidth). Further, we defined five instances of compute servers, five instances of file systems, four database instances, and five network subsystems. The resource instances belonging to the same resource type differ in their total and available capacities that can be consumed by a user submitted job.

For workload generation and job submissions, we used a standalone client. This client generates a batch of jobs, submits these in a batch to Resource Matcher, receives the matched results, and then repeats this cycle. The workload generator was configured to generate jobs with random resource requirements with the following constraints: each job has a dependency on one computer system and zero or more dependencies on other types of resources (i.e., database, network, or file system). For each dependency, the capacity requirements were determined randomly within a specified range.

In the client, we varied the number of jobs submitted at a time in a batch and measured the time between the submission of a batch of jobs and the return of matched results, which we call Resource Matcher *response time*. For a given number of jobs N, the experiment was repeated 20 times (by submitting 20 consecutive batches) and the Resource Matcher response time was determined by taking the average over the 20 runs. Figure 6 shows the average response times for a batch of 10, 20, and 30 jobs, respectively. The response time shown on the Y-axis is in milliseconds. In addition to the total response time, we show the breakdown of the response time into three parts: (i) time spent in querying the repository for resource instances meeting the minimum criteria (shown as RSR

query), (ii) time spent in the optimizer that computes the actual matches, and (iii) the rest of the time which includes the cost of marshalling the individual components, web service overhead, and networking delays.

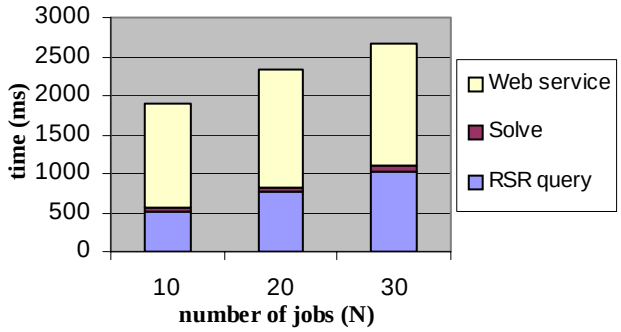


**Figure 6. Measurement of time to match N jobs**

Note that since the jobs in the workload have random job requirements, the resulting workload has a mix of jobs with varying resource requirements and job mix changes randomly from batch to batch. Because of this, the average response time for a given batch size should be viewed as an estimate of the expected performance of the Resource Matcher in matching N jobs with corresponding resources. We also note that the performance also depends on the number of resources in the repository and the number of resources identified as feasible resources meeting the minimum criteria for running a job. The number of resources in the repository affects the query time and number of feasible resources affects the optimization time. In our experiments, we had a fixed number of resources in the repository and therefore, the performance results shown in Figure 6 are a function of the repository we used.

Our preliminary performance results indicate that current implementation can handle resource matching for tens of jobs in several seconds. We break down the service time into three parts: *web service*, which is the time spent on communications among client and Resource Matcher web service; *solve*, which is the time used to solve the MILP problem for the resource matching, and *RSR query*, which is the time spent on querying Resource State Repository to get resource status.

From Figure 6, we can see that most of time is spent on communications among client and Resource Matcher web service. It is almost the same for different number of jobs because all job submissions make the same web service call with different parameters. The second largest part of the time is spent on querying Resource State Repository. It increases linearly with number of jobs. This is because in current implementation, Resource Matcher makes one query for each dependency of each job request. Currently, we are optimizing the query process by caching query results in Resource Matcher such that Resource Matcher does not have to query Resource State Repository every time. The smallest part of time, tens milliseconds, is spent on finding an optimal matching scheme. This result indicates that modeling and solving resource matching as a MILP problem is time efficient.

Our preliminary results shows that, using MILP, we can model the resource matching problem for a grid environment in a flexible and general way, and solve the problem efficiently. In our current implementation, we have used the trial version of CPLEX [1]. It only supports solving small size MILP problem (up to 300 constraints). However, this is not a major limitation even for realistic grid environments. This is because by grouping jobs into appropriate size batches, the size of the optimization problem can be managed. The problem size becomes large only if a job is associated with a large number of dependencies and constraints. In the future, we plan to use the full version of CPLEX to measure the performance of our system with jobs with complex dependencies.

## 6. Related work

Parallel job scheduling and resource allocation are well studied areas of research [2]. However, in most cases the resources considered are of only of one type -- processors and these are typically homogeneous. Moreover, on dedicated high performance systems, resource sharing is not considered. In our work, we consider jobs requiring multiple types of resources and we allow resource sharing, which is more common to grid environments. Maheswaran et al. [8] have considered the problem of mapping jobs to heterogeneous computing systems. In their work, they compare performance of several matching heuristics. Again they consider only one type of resource and they do not take into consider simultaneous resource sharing by multiple jobs.

Raman et al. [9], Cipriano et al.[13] and Liu et al. [6,7] have proposed modeling methods and algorithms to solve the resource matching problem. Their work focuses on finding optimal resources for one job with complex resource co-selection requirements. Instead of considering one job at a time, our work considers multiple jobs to archive global optimal matching scheme. Although not discussed in this paper, we have extended the models described in this paper to take into account pair-wise dependencies among multiple types of resources. This work will be discussed in a future paper.

In [5], Kumar and Naik have modeled the problem of resource allocation and matching of highly available services supported by resources that are subject to random failures. Similar to the jobs requirements considered in this paper, the highly available services have dependencies on multiple types of resources and these need to be satisfied at all times. The on-line resource matching mechanism monitors resources and whenever a failure is detected it computes new matches and allocations using currently

available components. The resulting problem is modeled as a linear optimization problem using an approach similar to the one describe in this paper. In this paper, the jobs arrive dynamically and their requirements change from job to job. Because of this, the actual models used are different.

## 7. Conclusions

In this paper, we have described a resource matching mechanism that is suitable for grid environments where resources are shared and jobs require multiple types of resources. The on-line resource matching problem is modeled and solved as an optimization problem using mixed integer linear programming methods. We have described how we can model the resource dependencies, capacity requirements and constraints, resource preferences and job priorities. We have also described modeling of four types global objective functions: maximizing throughput and priority weighted throughput, minimizing resource cost, balancing load among a set of resources, and maximizing job preferences for resources.

We also have described an architecture suitable for on-line resource matching in grid environments. Based on this architecture we have designed and implemented a job scheduling and resource management system using the resource matching mechanisms described in this paper. Such a system is suitable for managing datacenter resources and workloads consisting of independent jobs. By allowing system administrators to dynamically set resource usage policies and to change global objectives to suit business goals, the resource matching system described here can be adapted to dynamically changing conditions that are typical of large scale grid environments.

Performance measurements from our prototype implementation indicate that the core resource matching algorithms described in this paper are efficient in terms of speed and accuracy. In our experimental setting, the matching system is capable handling (without becoming a bottleneck) a workload consisting of jobs with an arrival rate of slightly more than 500 jobs per minute, which is a high arrival rate even for a busy compute/datacenter.

We note here that the prototype discussed in this paper was developed as a proof of concept and it was not tuned for performance. We have already extended this work to include jobs requiring multiple instances of a resource with the same type and jobs with dependencies across resource types. We have also extended this work to incorporate backfilling mechanisms to avoid job starvations. In the future, we will be studying the scalability of the system by increasing the number of resources and requests. We are also planning on incorporating other types of resource matching mechanisms in our system and comparing their performance on similar workloads.

## References


1. CPLEX. http://www.ilog.com as of Nov. 30, 2004.
2. D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling – a status report. Presented at 10th Workshop on Job Scheduling Strategies for Parallel Processing, NY 2004.
3. R. Fourer, D. Gay, and B. Kernighan. AMPL: A modeling language for mathematical programming. 2nd Ed., Publisher Thomson Brooks/Cole, 2003.
4. O. Ibarra and C. Kim. Heuristic algorithm for scheduling independent tasks on nonidentical processors. J. of the ACM, vol. 24, pp. 280-289, 1977.
5. V. Kumar and V. Naik. Modeling the global optimization problem in highly available distributed environments. Presented at the 4th Applied Mathematical Programming and Modeling Conference (APMOD) 2000, Brunel University, London, April 2000.
6. C. Liu, L. Yang, I. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In Proceedings of HPDC-11, pp. 63-81, 2002.
7. C. Liu and I. Foster. A Constraint Language Approach to Grid Resource Selection, Technical Report TR-2003-07, Department of Computer Science, University of Chicago, March, 2003.
8. M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In Proceedings of the 8th Heterogeneous Computing Workshop, pp. 30-44, 1999.
9. R. Raman, M. Livny, and M. Solomon. Policy-Driven Heterogeneous Resource Co-allocation with Gangmatching. In Proceedings of HPDC-12, pp. 80-89, 2003.
10. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, Grid Information Services for Distributed Resource Sharing. Presented at 10th IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, August 2001.
11. M. L. Massie, B. Chun, and D. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. Parallel Computing, vol. 30, 2004.
12. L. A. Wolsey. Integer Programming for Production Planning and Scheduling. Lecture notes available at http://www.core.ucl.ac.be/wolsey/chorin.ps
13. C. Santos, X. Zhu, H. Crowder. A mathematical optimization approach for resource allocation in large scale data centers. HP Labs Tech Rep 2002-64, 2002.
14. IBM WebSphere. http://www-306.ibm.com/software/websphere/
15. IBM DB2. http://www-306.ibm.com/software/data