Design and Evaluation of a Resource Selection Framework for Grid Applications

Chuang Liu* Lingyun Yang* Ian Foster*[#] Dave Angulo*

** Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*

[#] Math & Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, USA.

[chliu, lyang, foster, dangulo]@cs.uchicago.edu

Abstract

While distributed, heterogeneous collections of computers ("Grids") can in principle be used as a computing platform, in practice the problems of first discovering and then organizing resources to meet application requirements are difficult. We present a general-purpose resource selection framework that addresses these problems by defining a resource selection service for locating Grid resources that match application requirements. At the heart of this framework is a simple, but powerful, declarative language based on a technique called set matching, which extends the Condor matchmaking framework to support both single-resource and multiple-resource selection. This framework also provides an open interface for loading application-specific mapping modules to personalize the resource selector. We present results obtained when this framework is applied in the context of a computational astrophysics application, Cactus. These results demonstrate the effectiveness of our technique.

1 Introduction

The development of high-speed networks (10 Gb/s Ethernet, optical networking) makes it feasible, in principle, to execute even communication-intensive applications on distributed computation and storage resources [22]. In heterogeneous environments, however, the discovery and configuration of suitable resources for applications remain challenging problems. Like others [1, 6, 11, 21, 23, 28], we postulate the existence of a *resource selector service* responsible for selecting Grid resources appropriate for a particular problem run based on that run's characteristics; organizing those resources into a virtual machine with an appropriate topology; and potentially also assisting with the mapping of the application workload to virtual machine resources. These three steps—*selection*, *configuration*, and *mapping*—can be interrelated, as it is only after a mapping has been determined that the selector can determine whether one selection is better than another.

Many projects have addressed the resource selection problem. Systems such as NQE [8], PBS [19], LSF [33], I-SOFT [16], and Load Leveler [20] process user-submitted jobs by finding resources that have been identified either explicitly through a job control language or implicitly by submitting the job to a particular queue that is associated with a set of resources. This manually configured queue hinders the dynamic resource discovery. Globus [10] and Legion [7], on the other hand, present resource management architectures that support resource discovery, dynamical resource status monitor, resource allocation, and job control. These architectures make it easy to create high-level schedulers. Legion also provides a simple, generic default scheduler, but Dail et al. [12] show that this default scheduler can easily be outperformed by a scheduler with special knowledge of the application.

The AppLeS framework [6] guides the implementation of application-specific scheduler logic, which determines and actuates a schedule customized for the individual application and the target computational Grid at execution time. Petitet et al. developed a more modular resource selector for a ScaLAPACK application [23]. Since they embed application-specific details in the resource selection module, however, their tools cannot easily be used for other applications. MARS [18], SEA [29] and DOME [4] target particular classes of application. (MARS and SEA target applications that can

be represented by dataflow-style program graph, and DOME targets SIMD applications.) Furthermore, neither the user nor resource owner can control the resource selection process in these systems.

Condor [21] provides a general resource selection mechanism based on the *ClassAd language* [24], which allows users to describe arbitrary resource requests and resource owners to describe their resources. A *matchmaker* [25] is used to match user requests with appropriate resources. When multiple resources satisfy a request, a ranking mechanism sorts available resources based on user-supplied criteria and selects the best match. Because the ClassAd language and the matchmaker were designed for selecting a single machine on which to run a job, however, they cannot easily be applied when a job requires multiple resources.

To address these problems, we define a *set-extended ClassAds language* that allows users to specify aggregate resource properties (e.g., total memory, minimum bandwidth). We also present an extended *set-matching* matchmaking algorithm that supports one-to-many matching of set-extended ClassAds with resources. Based on these mechanisms, we present a general-purpose resource selection framework that can be used by different kinds of application.

Within this framework, both application resource requirements and application performance models are specified declaratively, in the ClassAd language, while mapping strategies can be determined by user-supplied code. (An open interface is provided that allows users to load the application-specific mapping module to customize the resource selector.) The resource selector locates sets of resources that meet user requirements, evaluates them based on specified performance model and mapping strategies, and returns a suitable collection of resources, if any are available. We also present results obtained when this technique was applied in the context of a nontrivial application, Cactus [2, 3].

The rest of this article is organized as follows. In Section 2, we present the set-extended ClassAd language and the set matching mechanism. In Section 3, we describe the resource selector framework. In Section 4, we describe a performance model and mapping strategy for the Cactus application used in our case study. In Section 5, we present experimental results. Finally, we summarize our work and discuss future directions.

2 Set-Extended ClassAds and Matching

We describe here our set-extended ClassAd language and set-matching algorithm.

2.1 Condor ClassAds and Matchmaking

A ClassAd (Classified Advertisement) [24] is a mapping from *attribute names* to *expressions*. Attribute expressions can be simple constants or a function of other attributes. A protocol is defined for *evaluating* an attribute expression of one ClassAd with respect to another ClassAd. For example, the expression "other.size > 3" in one ClassAd evaluates to **true** if the other ClassAd has an attribute named "size" and the value of that attribute is an integer greater than three. ClassAds can be used to describe arbitrary entities. In the current context, they are used to describe resources and user requests.

Condor matchmaking [25] takes two ClassAds and evaluates one with respect to the other. Two ClassAds *match* if each ClassAd has an attribute named "requirements" that evaluates to **true** in the context of the other ClassAd. (This symmetry distinguishes matchmaking from other commonly used selection mechanisms such as LDAP and relational queries. In this respect, matchmaking has similarities to unification [27].) A ClassAd can also include an attribute named "rank" that evaluates to a numeric value representing the quality of the match. When matchmaking is used for resource selection, the matchmaker evaluates a ClassAd request with respect to every available resource ClassAd and then selects a matching resource with highest rank.

2.2 Set-Extended ClassAd Syntax

In set matching, a successful match is defined as occurring between a single *set request* and a *ClassAd set*. The essential idea is as follows. The set request is expressed in set-extended ClassAd syntax, which is identical to that of a normal ClassAd except that it can indicate both *set expressions*, which place constraints on the collective properties of an entire ClassAd set (e.g., total memory size) and *individual expressions*, which apply individually to each ClassAd in the set (e.g., individual per-resource memory size). The set-matching algorithm attempts to construct a ClassAd set that satisfies both individual and set constraints. This set of ClassAds is returned if the set match is successful.

Set-extended ClassAds extend ClassAds as follows. (We emphasize that this syntax and other aspects of the set-matching framework continue to evolve.)

- A *Type* specifier is supplied for identifying set-extended ClassAds: the expression *Type="Set"* identifies a set-extended ClassAd.
- Three aggregation functions, *Max, Min*, and *Sum*, specify aggregate properties of ClassAd sets.

- A Boolean function *Suffix(V, L)* returns **true** if a member of list *L* is the suffix of scalar value V.
- The function *SetSize* returns the number of elements within the current ClassAd set.

The aggregation functions are as follows.

- *Max(expression)* returns the maximum value returned by *expression* when applied to each ClassAd in a set.
- *Min(expression)* returns the minimum value of *expression* in a ClassAd set.
- *Sum(expression)* returns the sum of the values returned by `expression` when it is applied to each ClassAd in a set. For example, *Sum(other.memory)>5GB* requires that the total memory of the resources selected be greater than 5 GB.

Aggregation functions might be used as follows. If a job consists of several independent subtasks that run in parallel on different machines, its execution time on a resource set is decided by the subtask that ends last. Thus, we might specify the rank of the resource set to be *Rank=1/Max(execution-time),* which means that the rank of the resource set is decided by the longest subtask execution time.

A user can use the *Suffix* function to constrain within particular domains the resources considered when performing set matching. For example, *Suffix(other.hostname, {"ucsd.edu", "utk.edu"})* returns **true** if *other.hostname="torc1.cs.utk.edu"* because *"utk.edu"* is the suffix of *"torc1.cs.utk.edu."*

2.3 Set-Matching Algorithm

The set-matching algorithm evaluates a set-extended ClassAd against a set of ClassAds and returns a ClassAd set that has highest rank. It comprises two phases.

In the *filtering* phase, individual ClassAd are removed from consideration based on individual expressions in the request. For example, the expression

"other.os==redhat6.1 && other.memory>=100M"

removes any ClassAd that specifies an operating system other than Linux Redhat v6.1 and/or with less than 100 Mb of memory. A *Suffix* expression can also be used in this phase, as discussed above. A set-matching implementation can index ClassAds to accelerate such filtering operations.

In the *set construction* phase, the algorithm seeks to identify a ClassAd set that best meets application requirements. As the number of possible ClassAd sets is large (exponential in the number of ClassAds to be matched), it is not typically feasible to evaluate all possible combinations. Instead, we use the following greedy heuristic algorithm to construct a set from the ClassAds remaining after Phase 1 filtering.

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

The algorithm repeatedly removes the "best" resource remaining in the ClassAd pool ("best" being determined by the rank of the resulting set) and adds it to the "candidate set." If the "candidate set" fulfills the specified requirements and has higher rank than the "best set" so far, the "candidate set" become the new "best set." This process stops when the ClassAd pool is empty. The algorithm returns the "best set" that satisfies the user's request, or failure if no such resource set is found.

This algorithm can adapt to different kinds of requests. It checks whether the candidate ClassAd fulfills the requirements expressed in the request and calculates the rank of the resource set based on the evaluation of the two expressions named as "requirements" and "rank" in the request ClassAd. Thus, by these two expressions, the user can instruct the matching algorithm to select a resource set with particular characteristics (as long as these characteristics can be described by expressions). This algorithm can also help the user choose the ClassAd set on which an application can get a preferred performance, for example, one on which the application can finish its work before a deadline.

The greedy nature of our algorithm means that it is not guaranteed to find a best solution if one exists. The set-matching problem can be modeled as an optimization problem under some constraints. Since this problem is NP-complete in some situations, it is difficult to find a general algorithm to solve the problem efficiently when the number of resources is large. Our work provides an efficient algorithm with complexity $O(N^2)$ with rank computation as the basic operation, where N is the number of ClassAds after the filtering phase.

3 Resource Selection Framework

We have implemented a general-purpose resource selection framework based on the set-matching

technique. It accepts user resource requests and finds a set of resources with highest rank based on resource information from a Grid information service. An open interface allows users to customize the resource selector by specifying an application-specific mapping module.

3.1 System Architecture

The architecture of our resource selection system is shown in Figure 1. Grid information service functionality is provided by the *Monitoring and Discovery Service* (MDS-2) component [9, 15] of the Globus Toolkit™ [17]. MDS provides a uniform framework for discovering and accessing system configuration and status information such as the compute server configuration and CPU load. The Network Weather Service (NWS) [30-32] is a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. Grid Index Information Service (GIIS) and Grid Resource Information Service (GRIS) [15] components of MDS provide resource availability and configuration information.

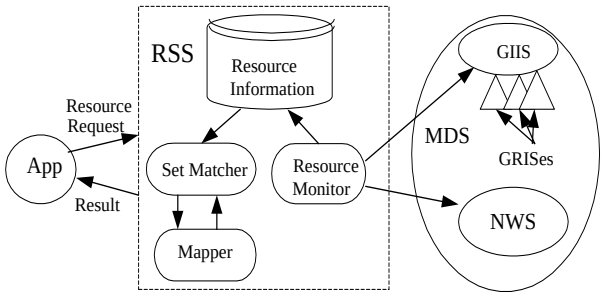


Figure 1: Resource selector architecture

The Resource Selector Service (RSS) comprises three modules. The *resource monitor* acts as a GRIS, in the terminology of [9]; it is responsible for querying MDS to obtain resource information and for caching this information in local memory, refreshing only when associated time-to-live values expire. The *set matcher* uses the set-matching algorithm to match incoming application requests with the best set of available resources. The performance of some applications, such as Cactus, is tightly related to resource topology and workload allocation. Thus, it is necessary to map the workload to the resources *before* judging whether those resources are good or bad. The *mapper* decides the resource topology and the allocation of application workload to resources.

Because the mapping strategy is tightly related to a particular application, it is difficult to find an efficient general mapping algorithm suitable for all applications. In addition, it is not yet clear how to express mapping constraints within ClassAds. Thus, we currently incorporate the mapper as a user-specified dynamic link library that communicates with the set matching process by instantiating certain ClassAd variables: e.g., Rlatency and Rbandwidth in the example in the next section.

3.2 Resource Request

The RSS accepts both synchronous and asynchronous requests described by set-extended ClassAds. It responds to a synchronous request with the best available resource set that satisfies this ClassAd, or "failure" if no such resources are available. An asynchronous request specifies a request lifetime value; the RSS responds if and only if a resource set that satisfies the specified ClassAd becomes available during the specified lifetime.

A request may include five types of element:

- *Type of Service*: Synchronous or asynchronous.
- *Job description*: The characteristics of the job to be run, for example, the performance model of the job.
- *Mapper*: The mapper program to be used.
- *Constraint*: User resource requirements, for example, memory capability, type of operating system, software packages installed.
- *Rank*: Criteria for ranking matched resources.

We can use these five elements to describe a variety of requests, as the following Cactus example shows.

```
1. [
2.  ServiceType = "Synchronous";
3.  Type="Set";
4.  iter=100; alpha=0; x=100; y=100; z=100;
5.  A=370; B=254; startup=30; C=0.0000138;
6.  computetime = x*y*alpha/other.cpuspeed*A;
7.  comtime = (other.RLatency+ y*x*B/other.RBandwidth
          +other.LLatency+y*x*B/other.Lbandwidth);
8.  exectime = (computetime+comtime)*iter+startup;
9.  Mapper = [type ="dll"; libname="cactus"; func="mapper"];
10. requirements =Suffix(other.machine, domains)
          && Sum(other.MemorySize) >= (1.757 + C*z*x*y);
11. domains={ cs.utk.edu, ucsd.edu};
12. rank=Min(1/exectime)
13. ]
```

Lines 2 and 3 specify that this is a synchronous set-matching request. Lines 4–8 are the job description including the problem size and the Cactus performance model. Line 8 models the execution time of every subtask on a machine. Line 9 gives the name and location of the mapping algorithm used for the application. Line 10 specifies the resource constraints, which state that (1) the total memory capability of the

resource set should be larger than the minimum size to keep the computation in memory that is described by a formula of the problem size, and (2) resources should be selected from machines in the "*cs.utk.edu*" or *"ucsd.edu"* domain that is described in Line 11. Line 12 indicates that the reciprocal of the execution time of the application is used as the criterion to rank candidate resources. Because the execution time of the application is decided by the subtask that finishes last, the rank of a resource set is equal to the minimum value of the reciprocal of the execution time of subtasks as specified in Line 12. If multiple resource sets fulfill the requirements, the resource set on which application gets smallest execution time has the highest rank.

3.3 Resource Selection Result

The result returned by the resource selector (expressed in XML) indicates the selected resources and mapping scheme. For example, the following is a result obtained for the Cactus application.

```
<virtualMachine>
 <result statusCode="200" statusMessage="OK"/>
 <machineList>
 <machine dns="torc2.cs.utk.edu"   processor= 2  x= 20>
 <machine dns="torc3.cs.utk.edu"   processor= 2  x= 15>
 <machine dns="torc6.cs.utk.edu"   processor= 2  x= 15>
 </machineList>
</virtualMachine>
```

This returned resource set includes three machines, each of which has two processors. These three machines have one-dimensional topology, and the workload is allocated to the machines according to the ratio 20:15:15.

4 Cactus Application

We applied our prototype in the context of a Cactus application that simulates the 3D scalar field produced by two orbiting sources [2, 3]. The solution is found by finite differencing a hyperbolic partial differential equation for the scalar field. This application decomposes the 3D scalar field over processors and places an overlap region on each processor. For each time step, each processor updates its local grid point and then synchronizes the boundary value.

4.1 Performance Model

In this Cactus experiment, we use the expected execution time as the criterion to rank all the sets of candidate resources. For a 3D space of X*Y*Z grid points, the performance model is specified by the following formulas, which describe the required memory and estimated execution time.

$$Memory\ size\ (MB) >= (1.757 + 0.0000138*X*Y*Z)$$

$$Execution\ time = startup\text{-}time + (computation\ (0) + communication\ (0)) * slowdown(CPU\ load)$$

Function *slowdown(CPU load)* presents the contention effect on the execution time of the application. CPU load is defined as the number of processes running on the machine. Figueira modeled the effect of contention on a single-processor machine [13, 14]. Assuming that the CPU load is caused by CPU-bounded processes and that the machine uses round-robin scheduling method, we extended her work by modeling the effect of contention on the dual-processor machine. We found that the execution time is smaller if we divide a job into two small subtasks than if we run this job as one task on dual-processor machines. We applied this allocation strategy to dual-processor machines and obtained the following contention model, which we validate in Section 5.1.

$$slowdown(CPUload) = (2*\ CPUcount - 1 + CPUload)/(\ 2*CPUcount - 1\ )$$

This formula is applicable when the CPU count is equal to one or two.

Computation(0) and *communication(0),* the computation time and communication time of the Cactus application in the absence of contention, can be calculated by formulas described in [26]. We incur a startup time when initiating computation on multiple processors in a Grid environment. In these experiments, this time was measured to be approximately 40 seconds when machines are from different clusters (sites) and 25 seconds when machines are in the same cluster.

4.2 Mapping Algorithm

We decompose the workload in the Z direction and decide the resource topology by using the following heuristic.

1. Pick the machine with the highest CPU speed as the first machine of the line.
2. Find the machine that has the highest communication speed with the last machine in the line, and add it to the end of the line.
3. Continue Step 2 to extend the line until all machines are in the line.

We thus attempt to minimize WAN communications by putting machines from the same cluster or domain in adjacent locations.

The mapper then allocates the workload to these resources. Our strategy is to allocate workload to each

processor in a fashion inversely proportional to the predicted execution time on that processor.

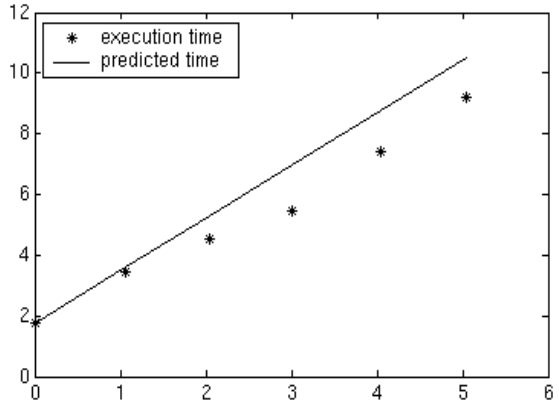
5 Experimental Results

To verify the validity of our RSS and the mapping algorithm developed for the Cactus application, we conducted experiments in the context of the Cactus application on the GrADS [5] test bed, which comprises workstation clusters at universities across the United States (including the University of Chicago, UIUC, UTK, UCSD, Rice University, and USC/ISI). We tested the execution time prediction function, the Cactus mapping strategy, and the set-matching algorithm.

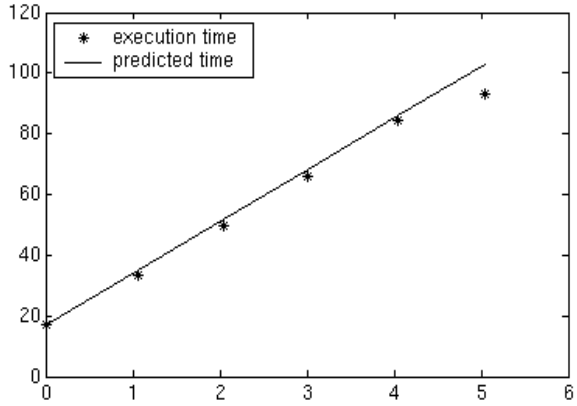
5.1 Execution Time Prediction Test

As noted above, our mapping strategy is based on the predicted Cactus execution time, which we also use for ranking the candidate resource sets (see Section 3.2). We tested the execution time prediction function both without and with communication time.

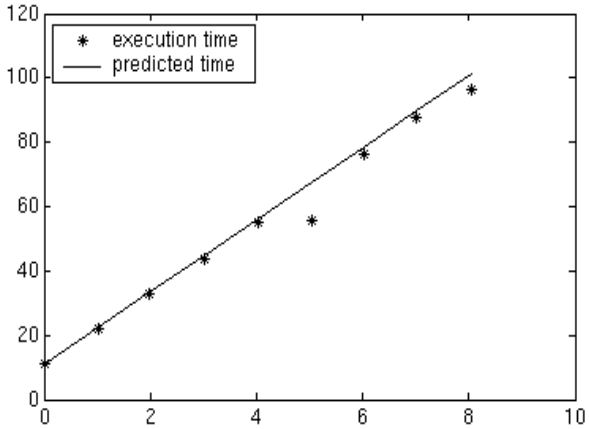
5.1.1. Computation Time Prediction Test. When the Cactus application runs on only one machine, there is no communication cost. To validate our computation time prediction function, we ran the Cactus application on one machine and compared the predicted computation time with the measured computation time. We did the experiments with diverse configurations, including (1) different problem sizes (20*20*20, 50*50*50, 100*100*100), (2) different clusters (UTK cluster, UIUC cluster, and UCSD cluster), (3) different CPU speeds (cmajor.cs.uiuc.edu 266 MHz, mystere.ucsd.edu 400 MHz, torc.cs.utk.edu 547 MHz), (4) different numbers of processors (UIUC and UCSD machines have 1 processor, UTK machines have 2 processors), and (5) different CPU loads.



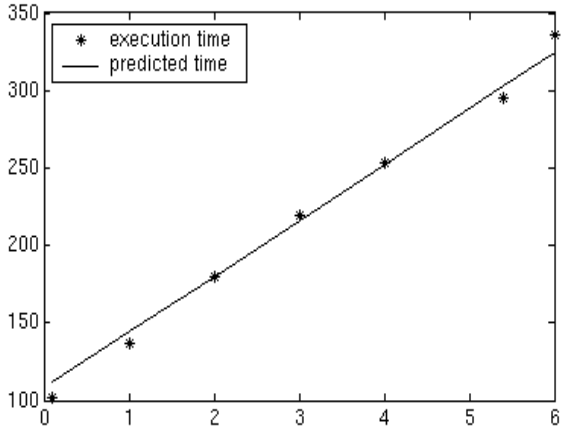
cmajor.cs.uiuc.edu (single-processor, PII 266 MHz)
Problem size: 20*20*20



cmajor.cs.uiuc.edu (single-processor, PII 266 MHz)
Problem size: 50*50*50



mystere.ucsd.edu (single-processor, PII 400 MHz)
Problem size: 50*50*50



torc1.cs.utk.edu (dual-processor, PIII 547 MHz)
Problem size: 100*100*100

Figure 2: Measured (points) vs. predicted (line) computation times (in seconds) for Cactus on a single node for different problem sizes and machines, as a function of CPU load

Figure 2 illustrates the predicted computation time and the measured computation time of the Cactus application with different CPU loads and various machine configurations. The results show that the computation time prediction function gives acceptable prediction in all cases. The error in this experiment was within 6.2% on average.

5.1.2. Computation and Communication Time. To test our execution time prediction function (which includes both computation time and communication time), we ran the Cactus application on various machine combinations and compared the measured execution time with the predicted execution time. We conducted experiments with various configurations, including (1) different problem sizes (100*100*100, 120*120*240, 140*140*280, 160*160*320, 200*200*400, and 220*220*420), (2) different clusters (UCSD cluster and UTK cluster), (3) different CPU speeds (o.ucsd.edu 400 MHz, torx.cs.utk.edu 547 MHz), (4) different numbers of processors (UCSD machines have 1 processor, UTK machines have 2 processors), and (5) different machine combinations.

Figure 3 shows both the predicted and measured execution time for the Cactus application when run in seven different configurations of problem size and machines:

1. 100*100*100 on torc
2. 120*120*120 on torc1 and torc5
3. 140*140*280 on torc1 and torc5
4. 160*160*320 on torc1 and torc5
5. 180*180*360 on torc1, torc3, and torc5
6. 200*200*400 on torc1, torc3, and torc5
7. 230*220*420 on torc1, torc3, torc5, o.ucsd.edu

The error is, on average, 13.13%. The time prediction formula works well except for problem size 160*160*320, where the predicted time is much greater than the execution time (error=59%). We monitored the CPU load of the machines on which the application had run during the experiments. We found that a competing application had been running on torc1 and torc5 when the resource selector collected system information to predict the execution time and this application terminated before our application run. We therefore believe that the reason for this large error rate is that the CPU load information we used to predict the performance of application does not reflect the real CPU load when the application ran.

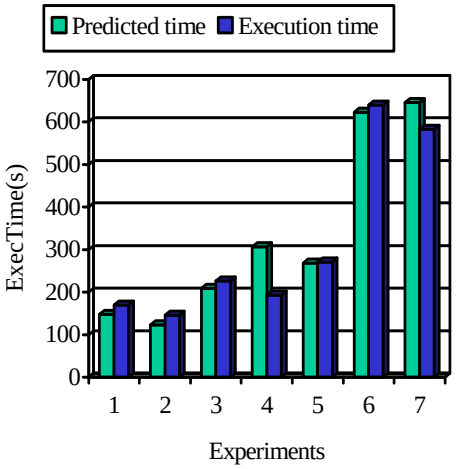


Figure 3. Predicted vs. real execution time for Cactus, in seven different configurations. See text for details.

5.2 Mapping Strategy Test

In the mapping strategy experiment, we evaluated the benefit gained from our mapping strategy. As mentioned in Section 4.2, the mapping strategy put machines with a high bandwidth connection into adjacent positions in the topological arrangement. Clearly, this one-dimensional arrangement minimizes the communication via WAN and thus reduces the total communication cost. In this section, we focused on how well the workload allocation strategy works. In particular, we tested whether the execution time of the Cactus application with allocation given by the mapper is shorter than its execution time with any other allocation strategy.

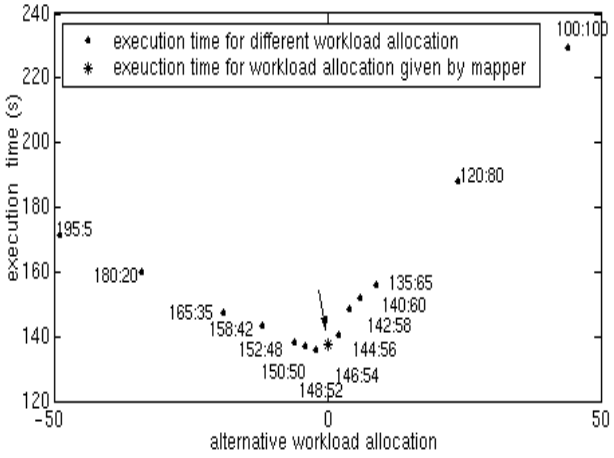


Figure 4. Execution times for different workload allocations on two machines.

We tested the workload allocation strategy on two machines (dralion.ucsd.edu and cirque.ucsd.edu). One machine (dralion) has a CPU speed of 450 MHz and no CPU load during the experiment. The other machine (cirque) has a CPU speed of 500 MHz and a CPU load of 2 during the experiment. We set up the Cactus application with a 3D space of 100*100*200 grid points and one-dimensional decomposition. According to our workload allocation strategy, the best performance was obtained when the workload was allocated on the two machines in the proportion of 146:54 (dralion:cirque) in the Z direction. We ran the Cactus application with this workload allocation and its variations (obtained by moving the division point to the right and left) and compared the execution time of the application with other workload allocation strategies.

The execution time for different workload allocations is shown in Figure 4. We can see that the execution time with the allocation given by the mapper is very close to optimal (only 1.2% higher than optimal). Moreover, the execution time increases when the deviation from our workload allocation scheme increases. Thus we can say that the workload allocation strategy works well.

5.3 Resource Selection Algorithm Test

To validate the resource selection algorithm, we asked the resource selector to select a set of machines for the Cactus application from a pool of three candidates. We also measured the execution time for the Cactus application on all possible machine combinations and used that information to determine whether the resource selector made the best choice. We performed this experiment both on machines from a single cluster and on machines from different clusters.

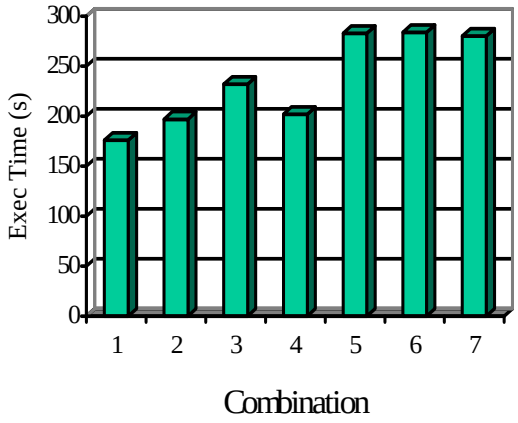


Figure 5. Execution time on all combinations of three candidate machines from a single cluster

In the single cluster experiment, the three machines are: {mystere, o, saltimbanco}.ucsd.edu. We number the seven possible machine combinations as follows:

1. mystere, o, saltimbanco
2. o, mystere
3. saltimbanco, mystere
4. o, saltimbanco
5. o
6. saltimbanco
7. mystere

These machines are connected via 100 Mbps Ethernet and thus communication costs between machines are relatively small. Our results, presented in Figure 5, show that the lowest execution time is for the first of the seven combinations: all three machines. Happily, our resource selector also identifies this combination as the best.

In the two-cluster experiment, the machines are torc6.cs.utk.edu and {o, saltimbanco}.ucsd.edu. We number the seven combinations as follows.

1. torc6, o, saltimbanco [BOTH]
2. torc6, saltimbanco [BOTH]
3. o. saltimbanco [UCSD]
4. torc6, o [BOTH]
5. saltimbanco [UCSD]
6. o [UCSD]
7. torc6 [UTK]

In this case, the high cost of inter-cluster communication resulted in the resource selector selecting the single (high-speed) UTK machine. The results in Figure 6 confirm that this was the right choice.

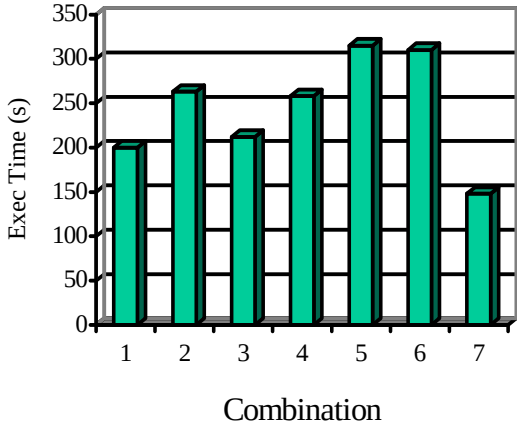


Figure 6. Execution time on all combinations of three candidate machines from two clusters.

6 Conclusion and Future Work

Grids enable the aggregation of computational resources to achieve higher performance and/or lower cost, than can be achieved on a single system. The heterogeneous and dynamic nature of Grids, however, leads to numerous technical problems, of which resource selection is one of the most challenging.

We have presented a general-purpose resource selection framework that provides a common resource selection service for different kinds of application. This framework combines application characteristics and real-time status information to identify a suitable resource set. A set-extended ClassAd language is used to express resource requests, and a new technique called set matching is used to identify suitable resources. We have used an application, Cactus, to validate the design and implementation of the resource selection framework, with promising results.

Our framework should adapt to different applications and computational environments. Further experiments on other kinds of application are needed to validate and improve our work. We also plan to provide more mapping algorithms for different kinds of application.

Acknowledgments

We are grateful to Alain Roy and Jennifer M. Schopf for many suggestions, and to our colleagues within the GrADS project for providing access to testbed resources. This work was supported in part by the Grid Application Development Software (GrADS) project of the NSF Next Generation Software program, under Grant No. 9975020.

References

1. Abramson, D., Giddy, J. and Kotler, L., High Performance Modeling with Nimrod/G: Killer Application for the Global Grid? In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2000
2. Allen, G., Benger, W., Dramlitsch, T., Goodale, T., Hege, H., Lanfermann, G., Merzky, A., Radke, T., Seidel, E. and Shalf, J. Cactus Tools for Grid Applications. *Journal of Cluster Computing* (4). 179-188. 2001.
3. Allen, G., Goodale, T., Lanfermann, G., Seidel, E., Benger, W., Hege, H.-C., Merzky, A., Mass'o, J., Radke, T. and Shalf, J. Solving Einstein's Equation on Supercomputers. *IEEE Computer* (32). 52-59. 1999.
4. Arabe, J.N.C., Beguelin, A., Lowekamp, B., Seligman, E., Starkey, M. and Stephan, P., DOME: Parallel Programming in a Heterogeneous Multi-User Environment. In *Proceedings of the 10th International Parallel Processing Symposium*, (Honolulu, Hawaii, 1996), IEEE Computer Society, 218-224
5. Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L. and Wolski, R. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, *15* (4). 327-344. 2001.
6. Berman, F. and Wolski, R., The AppLeS project: A Status Report. In *Proceedings of the 8th NEC Research Symposium*, (Berlin, Germany, 1997)
7. Chapin, S.J., Katramatos, D., Karpovish, J. and Grimshaw, A., Resource Management in Legion. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99)*, (San Juan, Puerto Rico, 1999)
8. Cray, R. Document number in-2153 2/97. Cray Research, 1997.
9. Czajkowski, K., Fitzgerald, S., Foster, I. and Kesselman, C., Grid Information Services for Distributed Resource Sharing. In *10th IEEE International Symposium on High Performance Distributed Computing*, (2001), IEEE Press, 181-184
10. Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W. and Tuecke, S., A Resource Management Architecture for Metacomputing Systems. In *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, (1998), 62-82
11. Dail, H. A Modular Framework for Adaptive Scheduling in Grid Application Development Environments *Computer Science*, University of California San Diego, 2002.http://grail.sdsc.edu
12. Dail, H., Obertelli, G. and Berman, F., Wolski, R., Grimshaw, A., Application-Aware Scheduling of a Magnetohydrodynamics Application in the Legion Metasystem. In *Proceedings of the 9th Heterogeneous Computing Workshop*, (Cancun, Mexico, 2000)
13. Figueira, S.M. and Berman, F., Modeling the Effects of Contention on the Performance of Heterogeneous Application. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing(HPDC 5)*, (Syracuse, NY, 1996), 392-401
14. Figueria, S.M. and Berman, F., Predicting Slowdown for Networked Workstations. In *Proceedings of the Sixth IEEE International Symposium on High Performance Distributed Computing(HPDC 6)*, (Portland, Oregan, 1997)
15. Fitzgerald, S., Foster, I., Kesselman, C., Laszewski, G.v., Smith, W. and Tuecke, S. A Directory Service for Configuring High-performance Distributed Computations. In *Proc. 6th IEEE Symp. on High Performance Distributed Computing*, 1997, 365-375.
16. Foster, I., Geisler, J., Nickless, B., Smith, W. and Tuecke, S., Software Infrastructure for the I-WAY High-

Performance Distributed Computing Experiment. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, (1996), IEEE Computer Society Press, 562-571

17. Foster, I. and Kesselman, C. Globus: A Toolkit-Based Grid Architecture. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 259-278.
18. Gehring, J. and Reinefeld, A. MARS-A Framework for Minimizing the Job Execution Time in a Metacomputing Environment. *Future Generation Computer Systems*, *12* (1). 87-99. 1996.
19. Henderson, R. and Tweten, D. Portable Batch System: External reference specification. Ames Research Center, 1996.
20. I.B.M., C. IBM Load Leveler: User's Guide. Document number SH26-7226_00, IBM Corporation. 1993.
21. Litzkow, M., Livny, M. and Mutka, M., Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, (1998), 104-111
22. Messina, P. Distributed Supercomputing Applications. In Foster, I. and Kesselman, C. eds. *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 55-73.
23. Petitet, A., Blackford, S., Dongarra, J., Ellis, B., Graham Fagg, Roche, K. and Vadhiyar, S. Numerical Libraries And The Grid: The GrADS Experiments With ScaLAPACK. *International Journal of High Performance Computing Applications*, *15* (4). 2001.
24. Raman, R. ClassAds Programming Tutorial (C++). 2000, http://www.cs.wisc.edu/condor/classad/c++tut.html.
25. Raman, R., Livny, M. and Solomon, M., Matchmaking: Distributed Resource Management for High Throughput Computing. In *IEEE International Symposium on High Performance Distributed Computing*, (1998), IEEE Press
26. Ripeanu, M., Iamnitchi, A. and Foster, I. Performance Predictions for a Numerical Relativity Package in Grid Environments. *International Journal of High Performance Computing Applications*, *15* (4). 2001.
27. Robinsin, J.A. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, *12* (1). 23-41. 1965.
28. Shao, G., Berman, F. and Wolski, R., Master/Slave Computing on the Grid. In *Proceedings of the 9th Heterogeneous Computing Workshop*, (Cancun, Mexico, 2000), 3-16
29. Sirbu, M.G. and Marinescu, D.C., A Scheduling Expert Advisor for Heterogeneous Environments. In *6th Heterogeneous Computing Workshop*, (1997), IEEE Press
30. Wolski, R. Dynamically Forecasting Network Performance Using the Network Weather Service. *Journal of Cluster Computing*. 1998.
31. Wolski, R., Spring, N. and Hayes, J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, *15* (5-6). 757-768. 1999.
32. Wolski, R., Spring, N. and Hayes, J., Predicting the CPU Availability of Time-shared Unix Systems on the Computational Grid. In *Proceedings of the 8th High-Performance Distributed Computing Conference, August, 1999*, (1999)
33. Zhou, S., LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems. In *Workshop on Cluster Computing*, (1992)