Improving Grid Resource Allocation via Integrated Selection and Binding

Yang-Suk Kee, Ken Yocum, Andrew A. Chien
Computer Science and Engineering, University of California at San Diego
yskee@csag.ucsd.edu, kyocum@cs.ucsd.edu, achien@ucsd.edu

Henri Casanova
Information and Computer Sciences Department, University of Hawai`i at Manoa
henric@hawaii.edu

Abstract

Discovering and acquiring appropriate, complex resource collections in large-scale distributed computing environments is a fundamental challenge and is critical to application performance. This paper presents a new formulation of the resource selection problem and a new solution to the resource selection and binding problem called integrated selection and binding. Composition operators in our resource description language and efficient data organization enable our approach to allocate complex resource collections efficiently and effectively even in the presence of competition for resources. Our empirical evaluation shows that the integrated approach can produce solutions of significantly higher quality at higher success rate and lower cost than the traditional separate approach. The success rate of the integrated approach can tolerate as much as 15%-60% lower resource availability than the separate approach. Moreover, most requests have at least the 98th percentile rank and can be served in 6 seconds with a population of 1 million hosts.

1. Introduction

Over the last decade we have witnessed an explosion in the scale of distributed computing systems in areas such as Utility computing, Grid computing, Internet computing, P2P network, etc. A common research challenge in these areas is the efficient and effective resource allocation at large scale, which is critical to application performance.

Conceptually, resource allocation can be divided into resource selection (discovery) and resource binding (acquiring). Accordingly, the common architecture of traditional resource brokering systems [1-4] separates resource selection from resource binding. In fact, these systems mostly focus on resource selection and provide sophisticated resource specification languages and resource selection algorithms. Given a resource specification, a resource selection algorithm first discovers a matching set of resources using available resource information, and then the application attempts to acquire the resources by negotiating with individual local resource managers.

However, this separate approach cannot deal with binding failures efficiently. In the real world, resource binding can fail because of inaccurate resource information, authentication failure, and most importantly competition between applications. The key issue here is that these binding failures cannot be resolved until the system attempts to bind the resources after the resources have been selected. To address this issue, some systems [4] consider multiple solution candidates to a whole specification. Nevertheless, this approach still does not work well in shared resource environments because binding will succeed only when all the resources in a solution are available.

This paper proposes a novel approach to solving the resource selection and binding problem called **integrated selection and binding**. The key idea is to make resource selection lightweight and aware of binding. Our approach is effective for competitive access to resources in terms of binding success rate, resource quality, and cost. The specific contributions of this paper include:

- A new formulation of the resource selection problem, by which we convert the traditional NP-hard constraint satisfaction problem into a simple search problem - we perform off-line resource classification and express resource constraints via a compositional (and hierarchical) resource description language, the Virtual Grid Description Language (vgDL) [5]; the two techniques together enable fast on-line resource selection.
- A novel algorithm for solving the resource selection and binding problem, called integrated selection and binding - leveraging our resource classification and the compositional features in vgDL, this algorithm can bind appropriate resource collections successfully in the presence of binding failures.
- An empirical evaluation of the integrated approach comparing to the aforementioned separate approach - the integrated approach can satisfy complex resource

SC2006 November 2006, Tampa, Florida, USA
0-7695-2700-0/06 $20.00 ©2006 IEEE

requests against much higher binding failure rates and acquire resources of higher quality at lower cost.

This paper is organized as follows. Section 2 describes the resource allocation problem in shared resource environments. Then, we present the separate approach and our integrated approach to this problem in Section 3. In Section 4, we evaluate the two approaches using diverse resource specifications and probabilistic models of binding failures. Sections 5 and 6 discuss related work and summarize the results of our experiments.

2. The Problem

Via resource selection and binding, applications should be able to quickly acquire appropriate resources within large resource universes. Unfortunately, competition for resources is a fundamental challenge to resource selection and binding in shared resource environments [6-9], including Grid platforms [10-12]. While the selection process may discover an appropriate set of resources that match the application needs, the resources are useless if they cannot be bound.

The likelihood of binding success is contingent upon resource management policy of individual resource management systems. This problem is less serious in the systems where access is less controlled [6] although those systems suffer from poor performance under high degree of competition for resources. For example, the best-effort network service offered by the Internet trades resource guarantees for immediate access. In contrast, the binding failure problem is significant in environments where resource access is managed [13,14], and denial or delay of access is routine. For instance, Grid systems composed of batch-scheduled resources allow strong performance guarantees only after the resources are made available, which can be days or even weeks later.

Prior work in resource selection treats binding as a second-class constraint. The user submits a resource specification to the resource broker, which returns a set of physical resources matching the specification. Then, the user is responsible for binding these resources by interacting with individual resource managers. Since this approach was mainly built for time-sharing or dedicated resources, the resources are assumed available at the binding stage. However, many large computing platforms employ resource management systems to efficiently utilize the resources, multiplexing the resources among multiple users. Therefore, contention for resources leads to unavailability, and seemingly high-quality resources discovered by the resource broker may not be available or usable. To address this shortcoming, we propose to integrate resource selection and resource binding, enabling binding-aware selection.

A specification language, selection algorithm, and binding mechanism are the core elements that enable applications to obtain resources. They are tied together intimately, and the design of each affects each other. For example, if a resource specification is overly specific, it may not be possible to find enough solutions to satisfy the specification. Likewise, if selection deterministically chooses the same resources for all applications, most applications are likely to fail to bind appropriate resources. The following discussion reflects this critical inter-dependence.

3. Grid Resource Allocation

3.1 Language and Representation

We develop hierarchical resource description features, as part of a new language (vgDL) [5]. Our language builds upon the concepts in several existing constraint-based resource description languages that provide constructs for expressing constraints on static and dynamic attributes for computing resources and relationships between them. Prominent examples include Globus RSL [1], RedLine [3], Condor ClassAd [2,15,16], and SWORD [4]. These languages support ranking functions that sort candidates by some user-specified or default functions.

The key advances of vgDL are the capability to specify hierarchical resource aggregates and qualitative notions of network proximity between them. Leveraging these features, users can compose and decompose a specification in a hierarchical fashion. We call one or more closely related aggregates or single hosts within a complete vgDL search statement, i.e., an independent resource allocation unit, a **component**. We may then examine the selections for each component until binding succeeds, and compose a solution to the entire specification, aggregating the resources in a bottom-up fashion.

vgDL provides three resource aggregates, distinguished based on homogeneity and network connectivity:

- LooseBag: a collection of heterogeneous nodes with possibly poor connectivity.
- TightBag: a collection of heterogeneous nodes with good connectivity.
- Cluster: a set of well-connected nodes with near identical resource attributes.

As illustrated in Figure 1, aggregates themselves can be composed so that an application can request a LooseBag that contains Clusters each of which contains Nodes. Each Node must meet a set of static or dynamic resource constraints. In this example, Cluster is the only component in the specification.

In addition, vgDL provides four operators that define network connectivity between entities: close, far, highBW, and lowBW. These operators indicate coarse notions of network proximity in terms of latency and bandwidth. We define good connectivity in terms of thresholds set per search, or set by examining the current set of dynamic inter-

```
vgdl = LooseBagOf (clusters) [2:4] {
  clusters = ClusterOf (hosts) [64] {
    hosts = [ (Processor == "Pentium 4") &&
      (Clock >= 3000) && (Memory >= 1024) &&
      (Disk >= 20480) ] [Rank = Clock] } }
```

Figure 1. A vgDL Description for a Set of Clusters

node attributes. This means the vgDL specifications may be written oblivious to current network conditions or technologies. (See [5] for all details)

3.2 Separate Approach with N Selections

In systems such as Condor [17], Globus [18], SWORD, application users can use a structured constraint language to specify resource requirements on single entities and relationships amongst them. Mostly an entity is a host while some of them provide a concept of group. These systems construct a collection of resources for an entire specification and return it to the application. Then, the application is responsible for binding and accessing the resources of interest. Of course, if the resources are shared, such binding attempts may fail. SWORD averts the need to make repeated (potentially expensive) selections by returning multiple solution candidates.

Typically, the separate approach builds a collection of resources for a specification with a bottom-up approach. The algorithm collects a set of resources (typically the highly-ranked ones) [1-4] for each host and assembles the resources to construct a solution candidate that satisfies all the constraints in the specification. Then, it returns one or more solution candidates to the application for binding. The following describes this separate selection and binding algorithm.

Separate Selection and Binding Algorithm

1. *For each entity of a specification, collect a set of selections*
2. *Take one from the selections for each entity, and construct a solution candidate by assembling each selection, considering relationship between entities. Repeat this until N solution candidates are constructed*
3. *Remove the best one from the solution candidates, and attempt to bind it.*
4. *If any entity fails to bind, the entire solution fails. Try binding with next best candidates. If all N solution candidates are exhausted, return failure.*
5. *If every entity in a solution candidate is bound successfully, return the solution to the application.*

Since the separate approach commonly returns the solution candidates that are globally optimized, binding failure of an entity results in binding failure of the entire request. Furthermore, rebinding the failed entity to other resources satisfying the relationship constraints between entities could be prohibitively expensive.

3.3 Integrated Approach with N Selections

To realize efficient and effective resource allocation, we propose the integrated selection and binding approach. The key idea is binding-aware selection. First, we exploit the design of vgDL to make the compositional structure of resource requests explicit and malleable to identify independent resource allocation components. Second, we select multiple sets of resources for each component and then bind the resources until sufficient resources to compose a solution have been collected. The following is a brief description of the algorithm.

Integrated Selection and Binding Algorithm

1. *Identify components in a specification, clustering closely related entities together into one*
2. *For each component, select N best selections.*
3. *Remove the best one from the selections for each component, and attempt to bind them in parallel.*
4. *If a component fails to bind its selection, only the component fails. Try binding with next best ones for the component. If any component exhausts all N selections, return failure.*
5. *If every component succeeds to bind its selection, compose the bound resources into a solution to the entire specification, exploiting the compositional structure, and return the solution to the application*

One critical aspect of this approach is the ability to decompose specifications flexibly, which is an intentional design element of vgDL where both LooseBag and resources far away (or those with no relationship) form natural bases of decomposition. Successful decomposition enables pre-selection of composable resource collections and accordingly makes it possible to compose the bound resources into a solution. The binding process is straightforward as it simply tries to bind one of the selections for each component, without backtracking. Therefore, this enables N selections for each component to correspond to $N^{components}$ solution candidates for a whole request. Without composition flexibility, N selections for each entity in the separate approach would correspond to only N solution candidates. Our hypothesis is that this flexibility will prove critical to succeed in meeting large, complex resource requests in highly-contested resource environments.

3.4 Virtual Grid and Execution System

A set of resources allocated for a vgDL specification is presented to the application through a Virtual Grid (VG) [19]. A VG provides a high-level hierarchical abstraction of the resource collection. Corresponding to the vgDL structure, the VG has a resource tree, of which leaves are decorated with allocated hosts and intermediate nodes represent aggregate and connectivity between child nodes. A VG is an active entity that is the instantiation of the desired resource collection. The application can then use the VG to find specific information about the allocated physical resources, to deploy application tasks, and to modify or evolve the resource collection. Note that a VG does not define how the application uses resources, nor does it provide a functional virtualization as in a virtual machine.

The Virtual Grid Execution System (vgES) realizes the VG abstraction. The key components of vgES are vgFAB, vgAgent, vgLaunch, and vgMon. The application provides a specification of its resource needs in vgDL to vgES, and then vgFAB realizes the integrated selection and binding algorithm to allocate appropriate resources. To achieve this, vgFAB uses resource information presented by vgAgent, which provides a single interface to extent resource information services (e.g., MDS [20], NWS [21], Ganglia [22], and R-GMA [23]). vgFAB also requests binding services to vgLaunch, which interfaces to individual local resource managers [13,14,17] to acquire resources on which it initializes the application. Once all necessary resources have been allocated, vgFAB builds a VG to represent the collected resources and returns it to the application. With this VG, the application can utilize the resources and use sophisticated, application-specific scheduling, load-balancing, and other optimization techniques as well as evolve the resource collection (e.g., modify the VG). During the lifetime of the VG, vgMon monitors the resources to ensure that performance expectations are met throughout application execution.

3.5 Implementation

3.5.1 Data Representation

Retrieving resource information directly and interactively from deployed information services to satisfy resource requests is prohibitively expensive and not scalable in general. Instead, our system proactively retrieves available resource information periodically and caches it in a local information store. The challenge in this approach is to identify a data representation used by the information store that allows fast and effective resource selection.

Our data representation is based on resource classification: we classify resources into three categories corresponding to three aggregates in the vgDL language using the information from resource information services. Tightly coupled homogeneous hosts compose a Cluster, and the well connected Clusters form a TightBag. All the TightBags together compose a large LooseBag. Both homogeneity and network proximity can be defined according to the characteristics of the system and the broad needs of applications, which will evolve as technology advances. For instance, hosts with the same system architecture, the same instruction set architecture, and similar clock rate can be regarded homogeneous, and if latency between two hosts is less than a certain threshold value, they are close to each other.

To use this resource classification for fast resource selection, we exploit relational database technology. Note that the notion of using relational databases for resource information has been explored in [23,24]. Our hierarchical resource classification is directly implemented as a database schema, which consists of three tables: Host, Cluster, and TightBag. The Host table contains static and dynamic attributes of all individual hosts. Dynamic resource attribute includes available memory and disk space, average CPU loads, etc. while static attributes include processor type, clock speed, cache size, installed memory and disk capacity, IP address, host name, etc. To keep track of the hierarchy, each host entry contains one of more "pointers'' to the Cluster(s) to which the host belongs. The Cluster table contains information about the resource properties that define homogeneity as well as aggregate resource properties such as total amount of memory and disk space, and number of hosts. The Cluster table also contains pointer(s) to the TightBag(s) to which the Cluster belongs. Similarly, the TightBag table contains an entry for each TightBag, with the total number of Clusters in the TightBag, network information that defines Cluster closeness, and aggregate resource properties such as total amount of memory and disk space. Finally, hosts in the Host table compose a large LooseBag.

3.5.2 Selection Algorithm

Different from the traditional selection algorithms that solve the online constraint satisfaction problem, our selection algorithm reduces the complexity of the online process by exploiting the hierarchical composition features and resource classification. The resource classification together with the hierarchical composition enables all searches to reduce to four levels of LooseBag, TightBag, Cluster, and Node, and simplifies selection process as simple pair-wise comparisons. This makes resource selection fast even with a large resource set. Since the classification process is offline, done while the information store is populated, and can be distributed, classification cost is not critical.

The resource selection algorithm is implemented as a sequence of SQL queries against a database structured using the aforementioned schema. vgDL specifications can have arbitrary levels of nesting, with ranking functions associated with each level. Rather than incurring the prohibitive cost of finding the optimal resource allocation, we opt for a fast algorithm that only evaluates each ranking function for each level once. Although this heuristic algorithm does not guarantee to return the optimal selection, we can quantify the quality of selection, and we will show in Section 4.4 that selections are close to the optimal.

In brief, the resource selection algorithm operates as follows. First, it builds a parse tree corresponding to the vgDL specification and identifies its components. Next, for each component, the algorithm synthesizes a sequence of SQL queries as follows. Infeasible resources that do not match the constraints on attributes of individual hosts or aggregates are eliminated using a WHERE clause. The ranking function is evaluated by declaring a dynamic attribute via an AS clause, and matching entries are sorted in descending order with respect to this dynamic attribute via an ORDER BY clause. Since the set of (intermediate or final) matching entries can be large, the algorithm decimates the entries of poor quality via a LIMIT clause. Once each component identifies selections, it is annotated with physical resource information for the selections.

4. Evaluation

We evaluate the integrated selection and binding comparing to the separate approach. Specifically, we focus on the following key questions:

- With what frequency do the resource selection and binding algorithms succeed for given specification complexity and resource availability? Does using multiple solution candidates improve success rate?
- What quality of results can each approach achieve and at what cost?

In the next sections, we describe our experimental methodologies and metrics for evaluation, and then present our experimental results.

4.1 Methodology

4.1.1 Simulated Environment

For our evaluation, we use a synthetic distributed resource environment generated by state-of-the-art tools for synthesizing representative cluster and network environments. We used BRITE [25] to generate a wide-area network topology and a statistical cluster configuration generator [26] to annotate the network topology with compute resources. These generators enable us to perform experiments for resource environments far larger than those for which actual information is available today. We use a resource environment with about 30,000 clusters, totaling about 1 million hosts while the size of cluster follows a lognormal distribution.

4.1.2 Resource Binding Model

Once we have generated the resource environment, we need a model of how the resources are managed and how they can be bound by an application. In the real world, resource binding can fail for many reasons, including competitive use by other applications, inaccurate resource information, authentication failure, etc. Of these, the dynamic behaviors due to competition for resources are of greatest impact.

As a first step, we use an analytic model to simulate resource availability with a probability of binding failure per resource manager. The advantage of this simplistic model is that it makes it easier to understand the significance of simulation results. Specifically, we use a Bernoulli distribution to model resource availability: for each binding request, a resource manager produces two outcomes, failure with probability p or success with probability $1-p$. For this model, we assume that a component is served by a resource manager, the only owner of resources, and that the behaviors of all resource managers are independent of each other. That is, each attempt to bind a resource for a component constitutes an independent random trial. If a request to a resource manager fails, however, successive attempts to the same resource manager will fail as well for some time to model the fact that the resource is busy with normal workloads, not with binding attempts (e.g., tentative binding). We do not claim that this model matches behaviors of resource managers in the real world, and such a model is an open question, which is beyond the scope of this paper. However, we believe that our simple models make it possible to identify meaningful trends when resource competition increases.

To cope with binding failures, both approaches can attempt to bind multiple solution candidates to increase the success probability. The separate approach can discover multiple selections to a whole specification. If all of these selections fail to bind, the whole request fails. By contrast, the integrated approach collects multiple selections for each component, and the bound resources together form a successful response to the request only if there is at least one successful binding for each component.

4.2 Metrics

4.2.1 Success Rate

Given a vgDL specification, our experiments attempt to find and bind resources that satisfy the specification in our simulated environment. The success rate is defined as the fraction of those attempts that would return a satisfying set of bound resources to the user.

4.2.2 Resource Quality

While it is important for selection and binding to succeed, a critical concern for users is the quality of the resources obtained. We defined resource quality based on the ranking functions specified in a vgDL request. Since a vgDL specification typically defines multiple ranking functions generating multiple dimensions of ranking, we define a way to combine them into a single metric. For instance, consider a specification for a Cluster with M to N processors, of which clock rates are higher than X. When Clusters are distributed as depicted on the left side of Figure 2, we average the percentile ranks in each dimension as an overall quality of a Cluster over the population of possible selections. Specifically, we define quality as $Q(R) = \sum_{i}^{n} p_i / n$, where p_i is the percentile of resource R when the resources are sorted according to the ith rank function. In Figure 2, the quality of Cluster 2 is (90+80)/2=85, and that of Cluster 1 is (100+60)/2=80. The overall quality of resources selected for a specification is then defined as the average of the resource qualities for all the components in the specification.

4.2.3 *Request Classification*

The quality of the resources chosen by a selection algorithm depends on the nature of the resource environment. Therefore, we would ideally need a metric that quantifies the difficulty of a request in a given resource environment for fair comparisons. Intuitively, one feels that the difficulty of a request depends on the number of possible selections and the distribution of the quality of these selections. However, it seems challenging to provide a single, general metric. Instead, we categorize requests in four groups and observe request difficulty empirically; we compute the total number of possible selections for each request and the standard deviation of the quality of these selections, and then characterize each as low or high based on the mean values of both parameters. We use the following acronyms to represent request classes.

- SC-SV: Queries with small number of solutions and small variance in their quality
- SC-LV: Queries with small number of solutions and large variance in their quality
- LC-SV: Queries with large number of solutions and small variance in their quality
- LC-LV: Queries with large number of solutions and large variance in their quality

4.2.4 *Selection Cost*

We define the cost of resource selection as the time spent by the resource selection algorithm to identify resources matching a vgDL specification.

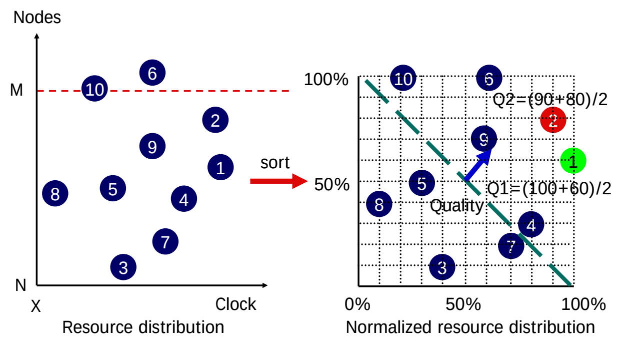


Figure 2. An Example of Resource Quality Evaluation: vgdl = ClusterOf (host) [N:M] { host = [Clock >= X] [Rank = Clock] }

4.2.5 *Binding Cost*

Resource allocation algorithms may bind resources partially before binding the entire set of needed resources. Moreover, resources may be bound tentatively but then unbound if other needed resources are not available. Some resources can be bound in parallel while others may be bound incrementally. We account for this hold time for each resource, and define the binding cost as the sum of all hold times until selection and binding for a vgDL request completes successfully or until it fails.

Hold times are measured in algorithm steps, where each step corresponds to the time for attempting to bind a resource. For example, an algorithm that attempts n bindings sequentially would consist of n steps, and if all bindings are successful, the binding cost would be $n(n+1)/2 \quad (= \sum_{i=1}^{n} i)$. On the contrary, if all bindings can be done in parallel and are successful, the cost would be 1.

4.3 Success Rate

We evaluated the binding success rate for both approaches with respect to specification complexity, resource availability, and number of selections. First, we vary the complexity of the vgDL specification from 1 to 16, defined as the number of components. Second, we vary the resource availability by changing the probability of binding failure of individual resource managers from 0.1 to 0.9. Third, we vary the number of selections used by the approaches from 1 to 16, for the whole request for the separate approach and for each component for the integrated approach. We consider a request with N-selections as a trial only when all N selections are available for the request. For each instantiation of these parameters, we perform 1000 random trials and calculate the success rate.

Results are presented in Figure 3. Each graph plots success rate versus binding failure probability and shows one curve per level of specification complexity. The graphs on the left

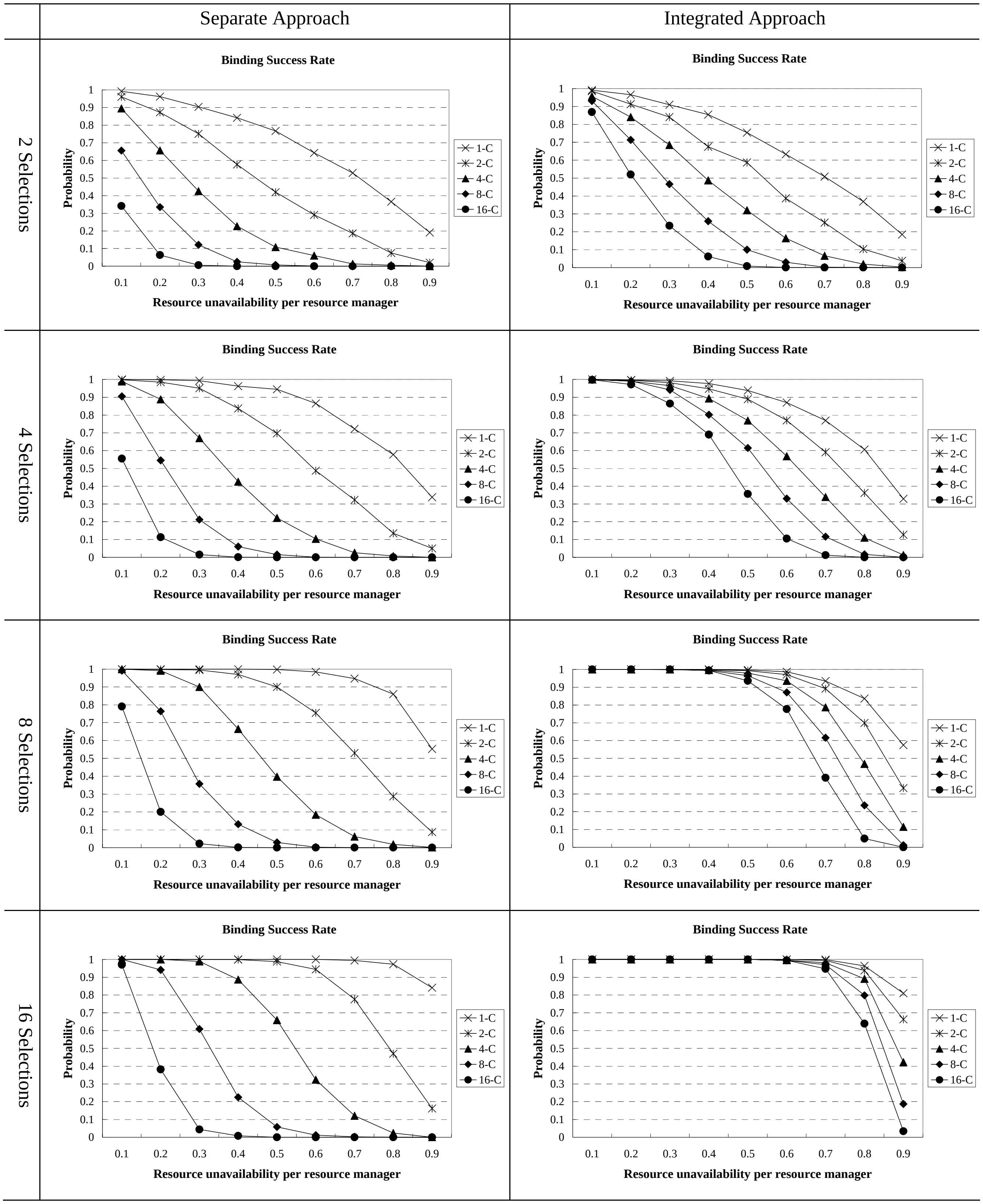


Figure 3. Binding Success Rate for the Integrated and the Separate Approaches, Varying Resource Availability, Specification Complexity (*x-C*: Number of Components), and the Number of Selections

side are for the separate approach while those on the right side are for the integrated approach. Each row of graphs is for a different number of selections pre-selected by both approaches. Overall, the success rate expectedly decreases as binding failure probability increases (i.e., when resource competition become higher) and as the specification complexity increases (i.e., when more resource managers are involved). In all cases, the integrated approach has an equal or greater success rate than the separate approach.

The major benefit of the integrated approach appears at low levels of resource availability and high levels of specification complexity. For instance, when both approaches use two selections, the integrated approach has a 30% higher binding success rate on average even with simple specifications that consist of only two components. This gap becomes wider with more complex specifications. Meanwhile, for specifications with 16 components, the success rate of the integrated approach can tolerate as much as 15%-60% lower resource availability than the separate approach, making it practical in realistic competitive resource environments. In contrast, the success rates of the separate approach drop quickly as specification complexity increases and/or resource availability decreases. Regardless of the number of selections, the separate approach never successfully binds resources for specifications with more than 8 components and binding failure probability above 60%, which would correspond to moderate resource unavailability in a production environment.

Another observation is that larger numbers of selections contribute to higher binding success rates. The separate approach can get higher success rate with more selections when the specifications have a few components (e.g., 2). However, when the specifications become more complex, the separate approach cannot take significant advantage of using many selections. In contrast, when using the integrated approach, the success rate for specifications with 16 components at 50% resource availability is 0% for 2 selections but the rate increases steadily to close to 100% for 16 selections. A remarkable difference in performance between two approaches is for large numbers of selections. With 16 selections, for instance, the integrated approach can always satisfy complex specifications (8 to 16 components) at moderate binding failure probability of 60% while the separate approach never succeeds for these complex specifications. In fact, the integrated approach has some chances to bind resources for the specifications with 8 and 16 components even with 90% binding failure probability. This difference is a crucial factor for utilizing resources from multiple resource managers synchronously in practice.

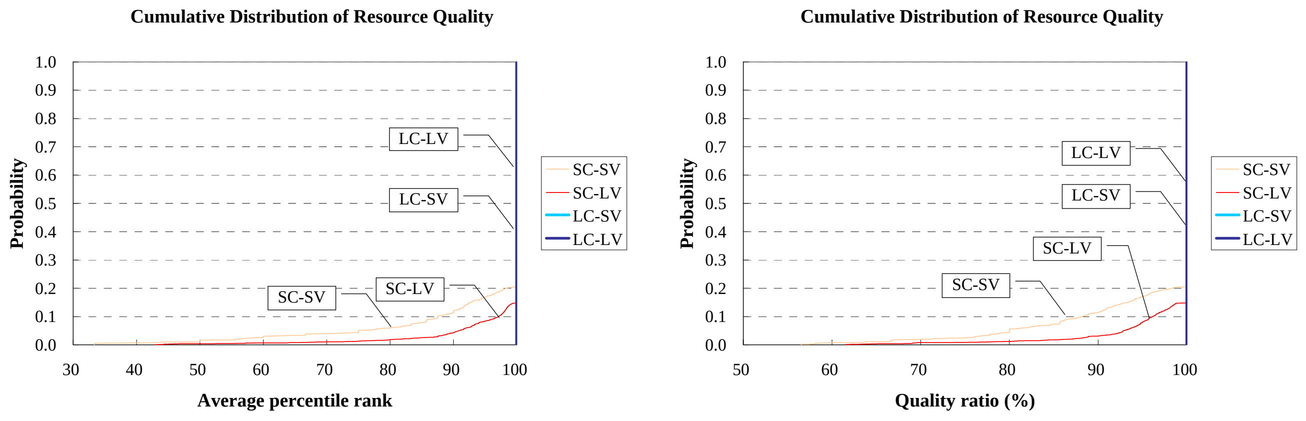
4.4 Selection Quality and Cost

A fundamental question for resource selection is how the quality of the selected resources compares to both the optimal and the population of possible selections that meet the minimum requirements. Note that most resource selection systems [1,2,24] do not provide any assurance of quality to users. For each resource selection attempt, we can compute the quality metric for all matching sets of resources in the environment and for the set of resources returned by the two approaches. Although the two approaches use different algorithms, both lead to the same quality of selected resources because they use the same underlying selection mechanisms.

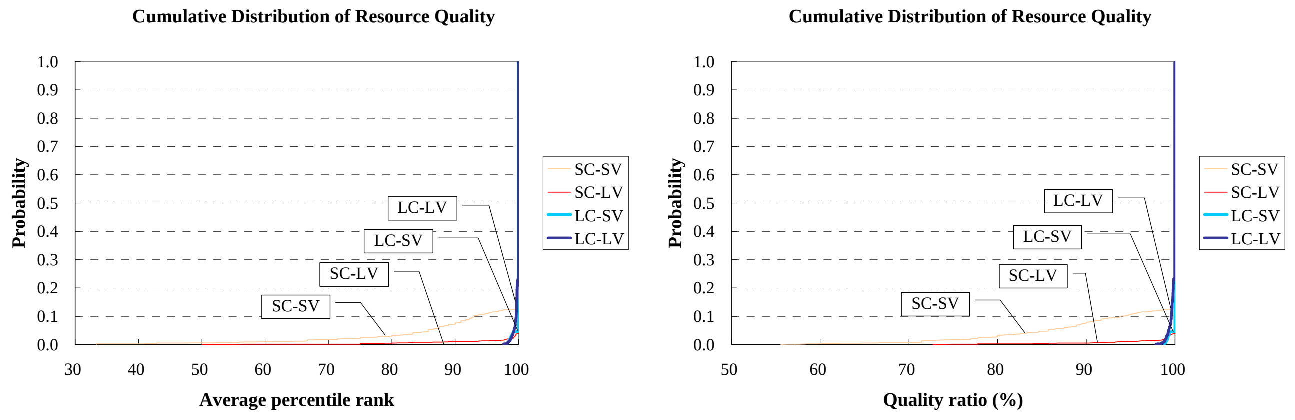
Critical determinants of the selection quality are the properties of the resource requests, the resource structure, and the quality of the heuristics. We evaluate the quality of selected resources for random Cluster or TightBag specifications, which are classified into four groups according to their difficulty. For each aggregate, about 4000 random specifications with different constraints on processor architecture, clock rate, memory capacity, and aggregate size were generated. We evaluate Clusters and TightBags separately as they are the basic components that form vgDL requests. Note that we do not present the quality results for LooseBags because the selection algorithms trivially pick the best LooseBag in all possible cases.

Figure 4 (a) and (b) show resource quality for Clusters and TightBags, respectively. The graphs on the left side plot the percentile ranks of resources versus the cumulative distribution of requests while those on the right side plot the ratio of quality of selected resources to that of the best one versus the cumulative distribution of requests. These graphs show that regardless of request types, requests with many possible selections (LC) have very close to and often optimal quality. All of them have at least the 98th percentile rank. 100% requests for Clusters and about 98% for TightBags have the optimal. These results show that the variance does not affect quality significantly for requests with many possible selections. Meanwhile, for requests with few possible selections, the majority of requests have quality close to optimal. About 80% of requests for Clusters with few possible selections and small variance (SC-SV) have optimal quality and about 89% of them have the 90th percentile rank. Only 1% of requests have poor quality below the 50th percentile. In these cases, the resource selection algorithm selects the biggest aggregate without considering host constraints because the current implementation arbitrarily prioritizes aggregate size over host constraints. Requests for TightBags show similar trends. Among the four request classes, requests with few possible selections and small variance in their quality are deemed to be the most difficult to find good solutions.

Note that poor percentile ranks do not always mean absolutely poor resource quality when only few possible selections are available because a slight difference in quality can result in a large difference in percentile value. The two graphs on the right side support this argument. When we evaluate the ratio of the selected resource quality to the best one, the resources have at least 57% quality.



(a) Selection Quality of Clusters



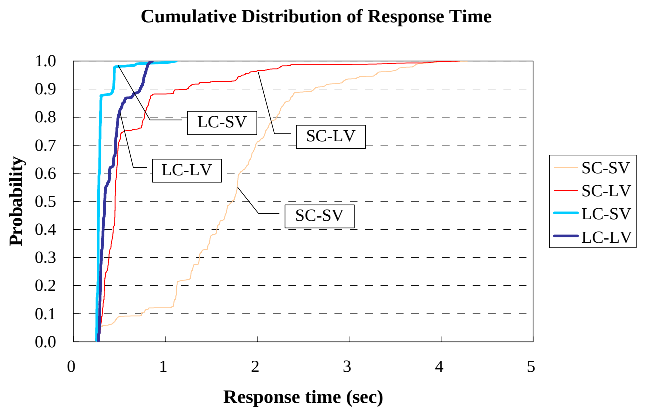
(b) Selection Quality of TightBags

Figure 4. Quality of Selections According to Request Classification; the Graphs on the Left Side Show the Cumulative Distribution of the Percentile Ranks and Those on the Right Side Show the Cumulative Distribution of the Ratio of the Quality of the Selected Resources to That of the Best One (=*Q(solution)/Q(best)*100*).

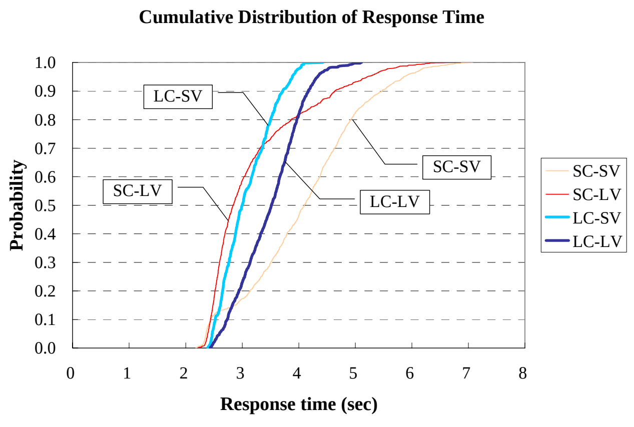
Our resource selection algorithm selects resources of high quality at low cost. Given the size of the resource population and a number of selections used by the algorithm, we measured the execution time of selection according to the request difficulty. Figure 5 (a) through (c) show the spectrum of response time for the three aggregates on a desktop machine with a Pentium4 3.2 Ghz processor with 1 GB main memory. The graphs plot the response time versus the cumulative distribution of requests. We use the same workload as for our evaluation of quality. No LooseBag request was categorized as LC-LV (many possible selections with large variance).

All the graphs show that the less available the possible selections, the longer the execution time, while the influence of the variance depends on aggregates. Nonetheless, overall selection cost is quite reasonable even in the worst cases. More than 90% of the requests can be served in 2.52 seconds for Cluster, 5.47 seconds for TightBag, and 5.56 seconds for LooseBag. Most cases in the other 10% exhibit longer response time because only few resources can satisfy the requests due to their scarce constraints, resulting in the entire database being scanned to find possible selections. Overall, requests for Clusters have the lowest cost because the algorithm exclusively uses the Cluster table and the Host table. Most requests with a large number of possible selections take less than 1 second. By contrast, since the requests for TightBags involve complex combinations and use all three tables (Host, Cluster, TightBag) to test if resources satisfy constraints, their selection cost is the highest (considering the worst cases). Finally, LooseBag requests use only the Host table. The reason why the requests for Cluster are less expensive than those for LooseBag is that the use of the Cluster table narrows the scope of searching the Host table.

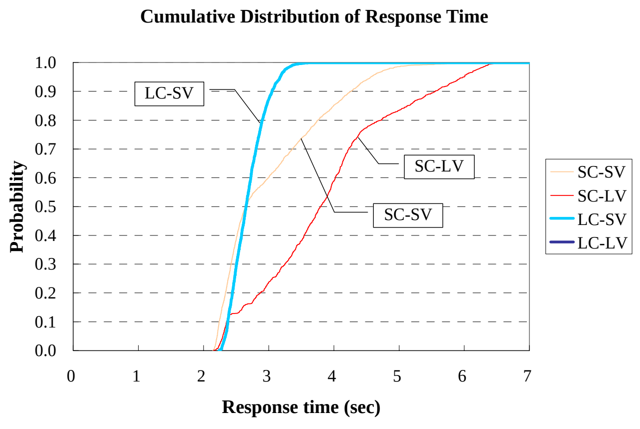
Even with a large population of 1 million hosts, the response time is only a few seconds. The resource classification and component-based search mainly contribute to fast selection. Moreover, we use multiple simple queries against single tables rather than join operations against multiple tables. Since resource selection is known to be an NP-hard



(a) Response Time of Clusters



(c) Response Time of TightBags



(c) Response Time of LooseBags

Figure 5. Cumulative Distribution of Response Time According to the Request Classification; We Use about 4000 Different Requests per Aggregate Varying the Processor Architecture, Clock Rate, and Memory Size.

problem [3], we deem these selection costs reasonable considering the quality of the resources obtained.

4.5 Binding Quality and Cost

Although selection quality is fundamental, what ultimately matters for the application is the quality of the bound resources. In Section 4.3, we have already shown that the integrated approach gives better chances to bind resources than the separate approach. In this section, we focus on the quality of bound resources and on binding cost when both the approaches succeed for the entire request.

Figure 6 shows the quality of bound resources for a LooseBag with 16 Clusters. We use the quality metric and the same experimental setups but with resource binding simulation using our Bernoulli trial model. We compute the average percentile rank of bound resources for requests with about 4000 different combinations of constraints, repeating 1000 random trials per resource availability.

The overall trend is that the integrated approach leads to equal or better binding quality than the separate one. The major benefit of the integrated approach appears at high levels of resource competition. By contrast, the binding quality of the separate approach drops quickly as resource competition increase and is close to zero when resource unavailability is above 40%. Expectedly, the separate approach cannot take advantage of multiple selections as it tries to bind resources for the whole request at once (16 solution candidates), as opposed to binding each of the 16 components one by one (16x16 solution candidates).

For both approaches, quality of requests with many possible selections is very close to and often optimal as long as binding succeeds. The binding quality of a request is highly related to the quality of its selections. That is, the top 16 selections out of many possible selections have better percentile ranks than those out of a few possible selections. In contrast, the quality of the requests with fewer possible selections worsens quickly as resource availability decreases. Another reason for this phenomenon is that some requests have less than 16 selections, i.e., less than 16 solution can-

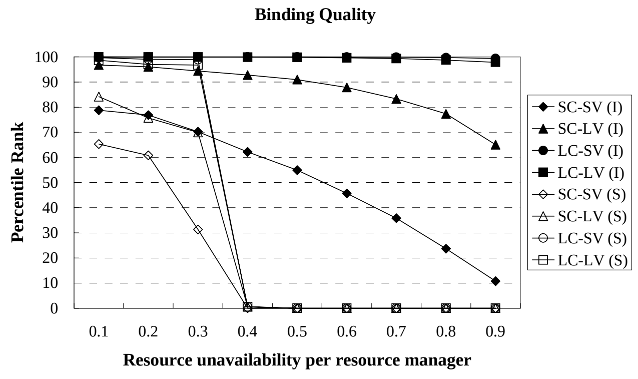


Figure 6. Quality of Bound Resources for Specifications with 16 Components Using 16 Selections (Acronyms: I: Integrated Approach, S: Separate Approach.)

didates, which gives the algorithm less chance to bind good resources and to succeed in binding.

To compare the two approaches in terms of binding costs, we performed the following experiment. For each approach, we compute the hold time until success using a maximum number of retries of 16, for the whole request for the separate approach and for each component for the integrated approach. Resources are bound for each retry, and end up being unbound if the retry is not successful. The hold times thus include hold time for resources that end up not being bound. Figure 7 (a) shows the average binding costs for both approaches versus resource availability. We show results only when 90% or more of the requests lead to successful binding, which explains why some of the curves for the separate approach are truncated.

The overall observation is that the separate approach ends up costing significantly more than the integrated approach in more competitive environments and with more complex requests as it fails more often (lower success rate) and holds resources more often (more binding trials). This can be understood plainly by examining the average number of binding attempts in this experiment, which we show in Figure 7 (b). Since the separate approach has low success rate, it tends to try binding more resources as the resource competition becomes higher. When resource unavailability is higher than 0.4, the total number of binding attempts reaches the maximum number of trials, which sums up to 256 (=16x16) attempts in total.

5. Discussion and Related Work

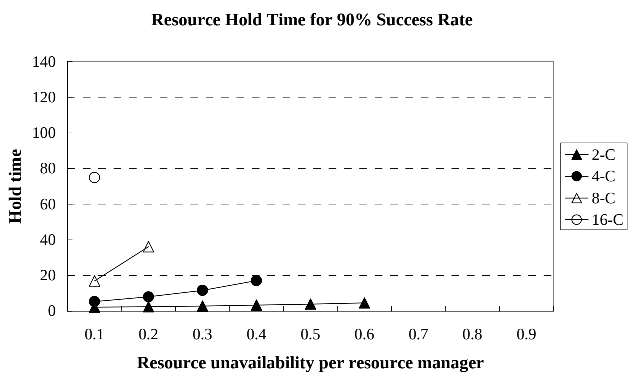
Traditional resource selection approaches formulate the problem as selecting a good set of resources to satisfy an application resource specification. For example, Globus's RSL [1] and the later RedLine [3] system focus only on resource description and selection. Resource binding is cleanly separated into a resource access protocol. In the Condor system and its evolutionary descendants (i.e., Condor-G [27]), the matchmaking system uses an ad-hoc constraint language to pair applications with suitable resources. However, the product of matchmaking is only an introduction of the application to the resource, and binding and access are negotiated subsequently by the Condor Resource Manager [17]. The Condor system was originally designed to deal with single resources, but has since been generalized to deal with sets (homogeneous resources) [15] and gangs (heterogeneous resources) [16]. However, the matchmaking model remains the same. Many other systems for resource description and selection [28-30] similarly separate selection from binding. There are a few distributed ad-hoc approaches such as used in SpiderNet [31] that perform resource selection and binding in an integrated fashion. However, these systems cannot provide any strong characterization of the quality and cost of the achieved results.

Our work supports the synchronous use of distributed resources across multiple resource managers – a capability traditionally tied to the notion of high-level resource selection and then co-scheduling (binding). Systems such as Globus DUROC [32] implement the co-scheduling or binding of resources that have already been selected. Higher-level systems such as GARA [33] and WS-Agreement [34] implement resource description and selection (or negotiation), lowering the description until it is tied to specific resources and resource managers and can be co-allocated, using lower-level DUROC or GRAM mechanisms. Our integrated selection and binding approach is strikingly different from these architectures that separate binding from selection.

Finally, one might combine high-level approaches for selection and negotiation (e.g. GARA and WS-agreement) with advance reservation in resource managers. We are aware that this approach is being pursued [35,36], and believe it is also promising. However, it depends on the widespread adoption of advance reservation in resource managers, which is not yet realized in the configuration of production resources.

6. Summary and Future Work

Understanding interplay between resource description, selection algorithm, and data representation is critical to efficient and effective resource allocation in distributed resource environments. We have described a new hierarchical resource description language (vgDL), the integrated selection and binding algorithm, and resource classification to allocate complex resource collections efficiently in the presence of opaque resource constraints. We have realized this idea in the framework of the Virtual Grid and its execution system (vgES). Through an efficient implementation using relational database technologies, we can provide a robust resource allocation service to the applications.



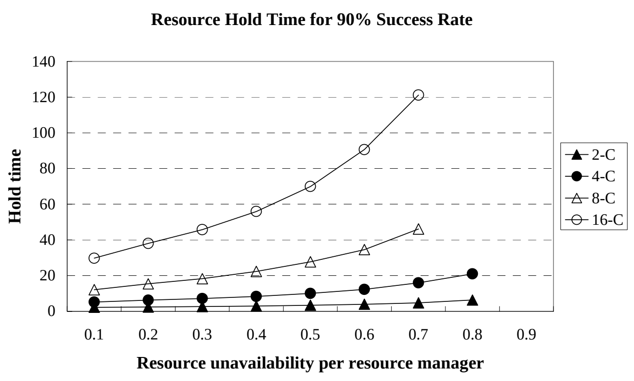
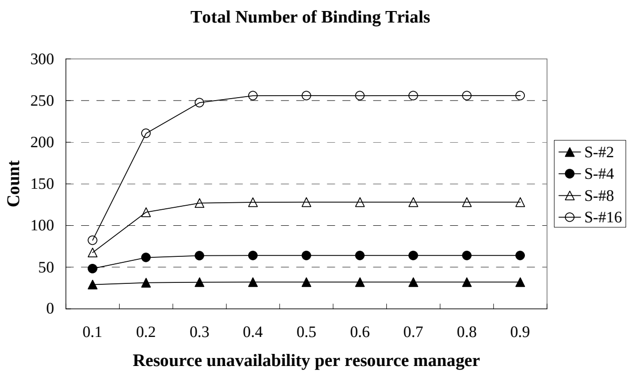


Figure 7 (a) Resource Hold Time for the Requests with More than 90% Success Rate Using 16 Solutions; the Left Graph is for the Separate Approach and the Right One is for the Integrated Approach (*x*–*C*: the Number of Components of Requests)



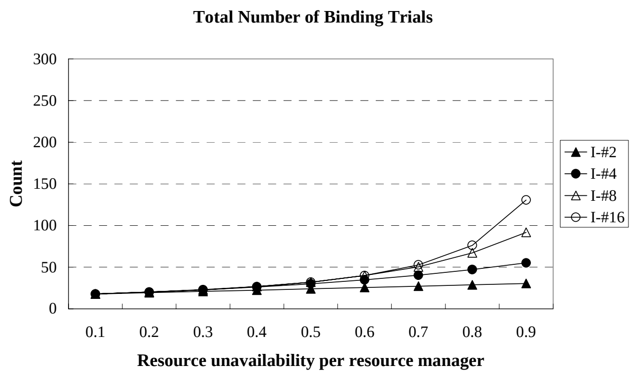


Figure 7 (b) Number of Binding Trials for the Requests with 16 Components until a Global Decision (Success or Failure) is Made; the Left Graph is for the Separate Approach and the Right One is for the Integrated Approach (-#*x*: the Number of Solutions)

We evaluated the integrated selection and binding approach in terms of success rate, selection quality and cost, and binding quality and cost. These experiments explore a range of request complexity and levels of resource competition. The empirical evaluation shows that the integrated approach with modest numbers of selections can provide significantly higher success rate and higher quality resources at lower cost than the traditional separate approach. The integrated approach takes the major benefit at low levels of resource availability and high levels of request complexity while the separate approach never succeeds even at moderate levels of resource availability and request complexity. The composable feature of the integrated approach fully exploits the extra selections with small overhead.

To exploit the advantages of simplistic model, this paper used the Bernoulli model for binding behaviors of resource managers. However, we understand that more realistic models would give more comprehensive evaluations of the algorithms. We are currently working on evaluating the binding algorithms using a behavioral model of the most popular resource manager i.e., batch system. Furthermore, it would be interesting to see how the algorithms affect application performance (e.g., with respect to makespan).

Although the current implementation is centralized, the algorithm does not preclude distributed implementations. The composable features of the integrated approach allow the system to be distributed. A distributed implementation is promising in the real world because selection and binding can be expensive operations. For instance, the resource information database can be replicated or distributed so that it is not a performance bottleneck or a single point of failure. In addition, collecting resource information could be distributed like web crawlers. Finally, multiple systems could cooperate to distribute requests from users. In future work, we will evaluate these design choices to see which one leads to the best performance.

Acknowledgements

The authors and research described here are supported in part by the National Science Foundation under NSF Cooperative Agreement NSF CCR-0331645, NSF NGS-0305390, and NSF Research Infrastructure Grant EIA-0303622. Support from Hewlett-Packard, BigBangwidth, Microsoft, and Intel is also gratefully acknowledged, as well as support from the Ministry of Information and Communication, Republic of Korea.

References

[1] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," Lecture Notes in Computer Science, 1459 ed., Workshop on Job Scheduling Strategies for Parallel Processing , Mar. 1998, pp. 62-82.

[2] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," IEEE Symposium on High Performance Distributed Computing (HPDC-8), Jul. 1998, pp. 140-147.

[3] C. Liu and I. Foster, "A Constraint Language Approach to Matchmaking," IEEE International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04), Mar. 2004, pp. 7-14.

[4] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Design and Implementation Tradeoffs for Wide-Area Resource Discovery," IEEE Symposium on High Performance Distributed Computing (HPDC-14), Jul. 2005, pp. 113-124.

[5] A. Chien, H. Casanova, Y.-S. Kee, and R. Huang, "The Virtual Grid Description Language: vgDL," Technical Report CS2005-0817, University of California, San Diego.

[6] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," ACM SIGCOMM Computer Communication Review, vol. 33, no. 1, pp. 59-64, Jan. 2003.

[7] B. White, S. Guruprasad, M. Newbold, J. Lepreau, L. Stoller, R. Ricci, C. Barb, M. Hibler, and A. Joglekar, "Netbed: An Integrated Experimental Environment," ACM SIGCOMM Computer Communication Review, vol. 32, no. 3, pp. 27-27, Jul. 2002.

[8] J. Touch and S. Hotz, "The X-Bone," Global Internet Mini-Conference in conjunction with Globecom '98, Nov. 1998, pp. 59-68.

[9] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Lebofsky, "SETI@home - Massively Distributed Computing for SETI," Computing in Science and Engineering, vol. 3, no. 1, pp. 78-83, Jan./Feb. 2001.

[10] F. Berman, "Viewpoint: From TeraGrid to Knowledge Grid," Communications of the ACM, vol. 44, no. 11, pp. 27-28, Nov. 2001.

[11] P. Avery and I. Foster, iVDGL Annual Report for 2002-2003, Aug. 2003.

[12] I. Foster, et al, "Grid2003 - The Grid2003 Production Grid: Principles and Practice," IEEE International Symposium on High performance Distributed Computing (HPDC-13), Jun. 2004, pp. 236-245.

[13] R. L. Henderson, "Job Scheduling Under the Portable Batch System," Lecture Notes in Computer Science, 949 ed., Workshop on Job Scheduling Strategies for Parallel Processing, 1995, pp. 279-294.

[14] IBM Corporation, IBM Load Leveler: User's Guide, Sep. 1993.

[15] C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and Evaluation of a Resource Selection Framework," IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Jul. 2002, pp. 63-72.

[16] R. Raman, M. Livny, and M. Solomon, "Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching," IEEE Symposium on High Performance Distributed Computing (HPDC-13), Jun. 2003, pp. 80-89.

[17] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," IEEE International Conference on Distributed Computing Systems (ICDCS-8), Jun. 1988, pp. 104-111.

[18] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputer Applications, vol. 11, no. 2, pp. 115-128, 1997.

[19] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. A. Chien, "Efficient Resource Description and High Quality Selection for Virtual Grids," ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID-5), May 2005, pp. 598 - 606.

[20] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," IEEE Symposium on High Performance Distributed Computing (HPDC-10), Aug. 2001, pp. 181-194.

[21] R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," Journal of Future Generation Computing Systems, vol. 15, no. 5-6, pp. 757-768, Oct. 1999.

[22] F. Sacerdoti, M. Katz, M. Massie, and D. Culler, "Wide Area Cluster Monitoring with Ganglia," IEEE International Conference on Cluster Computing, Dec. 2003, pp. 289-298.

[23] A. W. Cooke1, A. J. G. Gray, W. Nutt1, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. M. Fisher , S. Hicks, J. Leake, R. Middleton, A. Wilson, X. Zhu, N. Podhorszki, B. Coghlan, S. Kenny, D. O Callaghan, and J. Ryan, "The Relational Grid Monitoring Architecture: Mediating Information about the Grid," Journal of Grid Computing, vol. 2, no. 4, pp. 323-339, Dec. 2004.

[24] P. A. Dinda and D. Lu, "Nondeterministic Queries in a Relational Grid Information Service," ACM/IEEE Conference on High Performance Networking and Computing (SC'03), Nov. 2003.

[25] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An Approach to Universal Topology Generation," International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS-9), Aug. 2001, pp. 346-353.

[26] Y.-S. Kee, H. Casanova, and A. A. Chien, "Realistic Modeling and Synthesis of Resources for Computational Grids," ACM/IEEE International Conference for High Performance Computing and Communications (SC'04), Nov. 2004.

[27] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," IEEE International Symposium on High Performance Distributed Computing (HPDC-10), Aug. 2001, pp. 55-63.

[28] T. Kichkaylo, A. Ivan, and V. Karamcheti, "Constrained Component Deployment in Wide-area Networks Using AI Planning Techniques," IEEE International Parallel and Distributed Processing Symposium (IPDPS'03), Apr. 2003.

[29] A. Huang, and P. Steenkiste, "Building Self-configuring Services Using Service-specific Knowledge," IEEE International Symposium on High Performance Distributed Computing (HPDC-13), Jun. 2004, pp. 45-54.

[30] D. Spence and T. Harris, "XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform," IEEE International Symposium on High Performance Distributed Computing (HPDC-12), Jun. 2003, pp. 216-225.

[31] X. Gu, K. Nahrstedt, and B. Yu, "Spidernet: An Integrated Peer-to-peer Service Composition Framework," IEEE International Symposium on High Performance Distributed Computing (HPDC-13), Jun. 2004, pp. 110-119.

[32] K. Czajkowski, I. Foster, and C. Kesselman, "Resource Co-Allocation in Computational Grids," IEEE International Symposium on High Performance Distributed Computing (HPDC-8), Aug. 1999, pp. 219-228.

[33] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, "A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation," IEEE International Workshop on Quality of Service (IWQoS'99), Jun. 1999, pp. 27-36.

[34] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, Web Services Agreement Specification (WS-Agreement), version 1.1, May 2004.

[35] H. H. Mohamed and D. H. J. Epema, "Experiences with the KOALA Co-Allocating Scheduler in Multi-clusters," IEEE/ACM International Symposium on Cluster Computing and the GRID (CCGrid'05), May 2005, pp. 784- 791.

[36] K. Yoshimoto, P. Kovatch, and P. Andrews, "Co-Scheduling with User-Settable Reservations," Lecture Notes in Computer Science, 3834 ed., Workshop on Job Scheduling Strategies for Parallel Processing, Jun. 2005, pp. 146-156.