Combined Selection and Binding for Competitive Resource Environments

Yang-Suk Kee, Henri Casanova[1], Andrew A. Chien
Computer Science & Engineering and Center for Networked Systems, University of California, San Diego
yskee@csag.ucsd.edu, casanova@sdsc.edu, achien@ucsd.edu

ABSTRACT

A critical technology for Grid computing is the ability to describe, select, and bind appropriate resources for synchronous use by applications. Our Virtual Grid Description Language (vgDL) allows applications to describe and manage resources conveniently via application-level abstractions and enables a novel approach to application-driven resource management called "finding and binding". We explore the viability of this strategy to enables applications to obtain complex resource collections in competitive resource environments.

Our evaluation shows that combined resource "selection and binding" can scale well to millions of hosts, identifying good matches in a few seconds for the most complex resource requests. The combined selection and binding improves success rates for complex requests and the advantage increases for more complex requests and higher resource competition. Combined Selection and Binding can double the resource utilization (from 30% to 70%) at which synchronous resource allocation and use across as many as sixteen resource managers is possible.

1. INTRODUCTION

Over the past eight years, the Grid has moved from an ambitious vision pursued by a small number of academic researchers into a large-scale research and production activity involving hundreds of researchers, nearly all of the major computing technology vendors, and dozens of national and international initiatives. The grid vision of flexible, large-scale resource and data sharing across multiple organizations is the galvanizing shared goal for this large community. Remarkably, these efforts have spawned not only a wealth of technical research papers, but also major commercial products and large-scale production use for both large-scale science and commerce [1-3].

The broad, ambitious vision for the Grid has widespread appeal for the distributed and high performance computing communities [1-3]. However, as with most new technologies, only a small part of the Grid's potential is currently a reality. Many large-scale Grid applications in use on production Grid environments [4-6], perhaps the predominant use, involve asynchronous sharing of resources. These applications typically involve massive task-parallelism and may include processing of large-scale data [7-11]. Perhaps because it is fundamentally more difficult, a major element of the vision for Grid computing, synchronous use of shared resources distributed across multiple organization and administrative domains, is largely unrealized. Such synchronous use is challenging because it requires a form of co-scheduling [12], as well as a framework for managing a set of grid resources, including performance disappointments, resource failures, or even security compromises. In this paper, we focus on providing support for synchronous use of distributed grid resources.

A fundamental challenge for Grid applications is to describe and obtain appropriate resources to enable efficient, high performance execution; for synchronous applications, this challenge is further complicated by the need to deal with autonomous resource managers (generally at least one for each organization participating in the Grid). In the traditional approach [7,13-15], applications provide a resource description (i.e., RSL), resources are selected, a resource list is returned to the application, and finally the application attempts to "bind" (initiate execution) on those resources [14-21]. For the purposes of our discussion, the key elements of this approach are the separation of selection and binding, which means for resources across multiple sites, binding depends on co-scheduling of those resources. Efficient co-scheduling has turned out to be remarkably difficult to reconcile with local site resource management and the optimization of local performance metrics, so the practical result in this model is that synchronous applications are not well supported.

We propose a new approach for Grid applications to describe and obtain resources for synchronous use called "finding and binding". This is a new formulation of the problem with significant consequences for the achievable capabilities. By integrating the selection and resource binding process, our approach allows application's preference for resource choices to be flexibly combined with resource availability for which definitive information is only available from the resource managers. The result is that large complex resource needs involving multiple resources can be satisfied, even if resource availability is low. Specific contributions of this paper include:

1. Design of an integrated resource selection and binding framework, which includes an information organization, algorithms, and heuristics for adjusting to increased resource contention. This framework is realized in a working implementation called vgFAB (the Virtual Grid Finder and Binder)
2. Evaluation of Combined resource Selection and Binding across grid resource environments ranging from 100 to 1 million distinguishable resources. By using an information agent to tap Grid information services, a novel information schema, and powerful relational database technology, vgFAB performance scales well, and it returns good matches

[1] Also affiliated with the San Diego Supercomputing Center

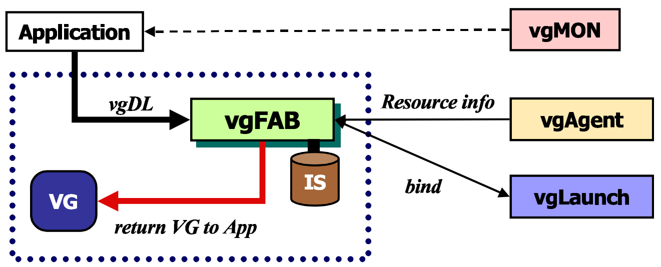


Figure 1. A Scenario of interactions between the vgES components

in less than two seconds for the largest resource environments and most complex resource requests.

3. Evaluation of Combined Selection and Binding against traditional separate selection approaches. The results show that Combined Selection and Binding not only dramatically increases success rates at high resource utilizations (binding failure rates), it does so with modest numbers of candidates. In short, Combined Selection and Binding can double the resource utilization (from 30% to 70%) at which synchronous resource allocation and use across as many as sixteen resource managers is possible.

These results represent a significant advance on our recent paper [22]; specifically, this manuscript includes the first description of the multi-candidate selection algorithm, and first in-depth evaluation of the effectiveness of integrated selection and binding approach.

This work is a part of the Virtual Grid Application Development Software (VGrADS) project (led by Ken Kennedy, see http://vgrads.rice.edu), which is developing the concept of a Virtual Grid as a flexible, high-level resource abstraction for Grid applications. Novel aspects of the Virtual Grid include: a high-level application-oriented resource description language (vgDL), integrated selection and binding scheme (described in this paper), and an explicit "reified" Virtual Grid resource abstraction that supports flexible application management of resources for fault-tolerance, performance, or other criteria.

The remainder of the paper is organized as follows. In Section 2 we provide background on the Virtual Grid, the Virtual Grid resource description language, and other relevant VGrADS context. In Section 3, we describe the core selection and binding algorithms, and their implementation in the vgFAB. In Section 4, we evaluate the algorithms and vgFAB implementation in terms of scalability, resource quality, and robustness. In Section 5, we discuss the implications of this evaluation and survey related work relevant to this work. Finally, we summarize our results and describing possible directions for future work in Section 6.

2. VIRTUAL GRIDS

One of the key innovations in the aforementioned Virtual Grid Application Development Software (VGrADS) project is the concept of a *Virtual Grid (VG)*. A VG provides a high-level, hierarchical abstraction of the resource collection that are needed and used by an application. This abstraction provides a clean separation of concerns between grid applications and the inherent complexity of the grid infrastructure: the application specifies its resource needs using a high-level language, vgDL, and the Virtual Grid Execution System (vgES) finds and allocates appropriate resources for the application. A virtual grid (VG), an active entity that is the instantiation of the desired resource collection, is returned to the application. The application can then use the VG to find specific information about the allocated physical resources, to deploy application components, 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 VG is a wholly high-level abstraction for managing resources.

The concept of a VG is realized in the vgES as shown in Figure 1. This paper focuses on the vgFAB component (inside the dotted rectangle) and the vgDL language, but we briefly describe the vgES components here to illuminate the larger context in which this work is being performance. The application provides a description of its resource needs, written in the vgDL language that we have designed and that we describe below, to the vgFAB ("Finder And Binder") component, which selects appropriate resources and allocates them for the application. To achieve this, vgFAB uses resource information gathered by the vgAgent component which provides a single interface to extent grid resource information services (e.g., MDS[23], NWS[24], Ganglia[25]), extracting and presenting both static and dynamic information to vgFAB (for incorporation into its information store). A novel aspect of vgFAB is that it performs integrated resource selection and resource binding. Resource binding is achieved via the vgLaunch component, which interfaces to local resource managers [26] to acquire resources on which it initializes the application (e.g. Bind the resources). Once all necessary resources have been found and bound, vgFAB builds a VG instance to represent the collected resources and returns it to the application. With this VG instance 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, the vgMON component monitors resources to ensure that performance expectations are met throughout application execution.

A central element of the VG approach is the virtual grid description language (vgDL), which provides resource abstractions that applications can use specify their resource needs and utilize allocated resources. We have described vgDL in a previous paper [22]. For the sake of completeness, we briefly review the major elements of the vgDL design here. vgDL is designed for direct use by application developers (human-readable), and provides high-level resource abstractions which correspond to what application programmers often use to organize their applications portably – across many different resource environments. The vgDL language was designed based on a detailed study of half a dozen real-world grid applications, and the full description and rationale is available in [27]. vgDL not only uses simple resources abstractions, supporting simple specifications, but is also a rich, expressive language that enables experts to control resource specification with precision. Figures 2 show the BNF for the vgDL language. We acknowledge borrowing from the design of RedLine [14] BNF for the description of resource attribute constraints.

Our study of scientific and grid computing applications showed that in order to design for performance (and to manage complexity) portably, application developers often use three simple resource abstractions to aggregate individual resources. Consequently, vgDL contains three resource aggregates, distinguished based on homogeneity and network connectivity.

```
Vgrid ::= VgDefineExpr ["at" time/event ]
VgDefineExpr ::= Identifier "=" VgExpr
Identifier ::= String
VgExpr ::= VgSubExpr | VgDefineExpr ("close" | "far" | "highBW" | "lowBW") VgDefineExpr
VgSubExpr ::= VgAssociatorExpr | VgNodeExpr | "{" VgExpr "}"
VgAssociatorExpr ::= VgBagExpr | VgClusterExpr
VgBagExpr ::= ("LooseBagof" | "TightBagof") "(" Identifier ")" "[" MinNode ":" MaxNode "]" [ "[" Number ("su" | "sec" ) "]" ]
[" [" Rank "=" ArithmeticExpr "]" ] "{" VgDefineExpr"}" |
MinNode ::= Integer
MaxNode ::= Integer
Number ::= Integer
VgClusterExpr ::= "Clusterof" "(" Identifier ")" "[" MinNode ":" MaxNode ["," MinTime ":" MaxTime] "]"
[" [" Rank "=" ArithmeticExpr "]" ] "{" VgDefineExpr "}"
MinTime ::= Integer
MaxTime ::= Integer
VgNodeExpr ::= "[" RedlineExpr "]" [" [""Rank" "=" ArithmeticExpr "]" ]
ArithmeticExpr ::= Arithmetic expression in Redline for ranking function

RedlineExpr ::= CondAndExpr [ "||" CondAndExpr ]* [ "," Predicate ]
CondAndExpr ::= EqualExpr [ "&&" EqualExpr ]*
EqualExpr ::= RelationalExpr [ ("==" | "!=") RelationalExpr ]*
RelationalExpr ::= AddExpr [ (">=" | "<=" | ">" | "<") AddExpr ]*
AddExpr ::= MultExpr [ ("+"|"-") MultExpr ]*
MultExpr ::= UnaryExpr [ ("*" | "/")  UnaryExpr ]*
UnaryExpr ::= Integer | Float | Attribute | "(" RedlineExpr ")" | ("Cluster" |"LooseBag" |"TightBag") "." Attribute)
Predicate ::= "Required" "(" Attribute ["," Attribute ]* ")"
Attribute ::= String
```

Figure 2. BNF description of vgDL language

- `LooseBag`: a collection of heterogeneous nodes with poor connectivity; users only care about number of nodes but node architecture and connectivity between nodes are not major concerns
- `TightBag`: a collection of heterogeneous nodes with good connectivity
- `Cluster`: adding homogeneity, a well-connected set of nodes with identical (or nearly so) individual resource attributes

Each aggregate specifies a range for its size (i.e., number of elements). User can specify constraints on attributes of individual elements within the aggregate (e.g., clock rate, processor architecture, memory, etc.), or constraints on aggregate attributes (e.g., total aggregate memory, total aggregate disk space). The user can define can new attributes and many attributes published by grid information services can be used in a vgDL specification. Note that aggregates can be nested (e.g., a LooseBag of Clusters) to arbitrary depth. The vgDL language has been used to express resource abstractions for over half a dozen major applications, and appears to be sufficient for all of the grid applications we have studied to date.

Aggregator properties and many of the characteristics of vgDL descriptions are *qualitative*. In our analysis of application needs, we found that detailed quantitative specifications are often a distraction, and as such cause resource specifications to be fragile when moving to new resource environments. In view of this, vgDL provides four operators that define network connectivity: `close`, `far`, `highBW`, and `lowBW`. These composers indicate coarse notions of network proximity in terms of latency and bandwidth. Particular implementation will use specific quantitative values as definitions for these operators, as appropriate for distribution of grid resources, and changing as technology advances. Learning how to set these values most productively for a particular resource pool distribution of resources, their connectivity, and application workload is a topic of active research. Note that applications that require detailed quantitative resource information for the resources they obtain can query grid information services for the resources in a VG once it has been instantiated. With these aggregates and network operators, an application can structure the specification of its resource environment in top-down fashion and decorate components with constraints when desired.

In addition to constraints, applications can also express resource preference by using a scalar rank function: a user-defined expression of basic arithmetic operators, resource attribute and resource aggregate attribute values which defines a scalar value which represents the quality of that resource set for the application's request. Because the vgDL requests are hierarchical, a specification may include multiple ranking functions (one at each level in each subcomponent). These rank functions are used by vgFAB to compare resource candidates and select those most desirable to the application (see Section 5). Compared to other systems [14,16-19], the vgDL rank functions provide significant flexibility, allowing combination of multiple attributes in complex fashion easily. vgDL provides many other capabilities, as described in [22,27].

Given a vgDL description, the vgFAB builds a corresponding parse tree, and then uses the structure of that tree to generates identify candidate resources. The translation of the vgDL request, and the resource information schema against which queries are made are described in detail in [22]. Section 3.1 explains how the structure of the resource information schema enables efficient resource selection and binding. Once appropriate resources have been found and bound, vgFAB constructs a VG with the bound resources, and returns it to the application. The VG has a hierarchical structure which corresponds to the hierarchical vgDL description. Each leaf in the hierarchy corresponds to a concrete resources. Each higher node in the structure corresponds to a vgDL relationship amongst resources. All of the virtual grid nodes are decorated with a wide range of attribute values; some defined by the system, others by users or information providers. vgES provides API's for applications to navigate the VG, read and modify attributes, and even modify the virtual grid. Applications may also discover attributes, retrieve attribute values (e.g., hostname of particular nodes). Additionally, the VG is a dynamic entity whose attribute values are periodically updated by the vgAgent. In addition, the application can evolve the VG to meet changing resource needs. In the next section we describe the algorithms and techniques that allow vgFAB to perform efficient and high-quality coordinated resource finding and binding in large-scale resource environments that exhibit contention for resources.

3. RESOURCE SELECTION & BINDING

In traditional systems [7,13,14], resource description and selection are separated from resource binding. That is, applications request resources, and receive a list of candidates. If those candidates, perhaps a single response to the resource description request, or in some cases a list of responses to the request, are deemed suitable, then the application proceeds to attempt to acquire (bind) them. We term this traditional approach **Separate Selection**. A parameter for this approach is the number of distinct candidates – each response to the original resource request – returned. In order to satisfy large, complex resource requests in grids which have competition for resources, we are pursuing an approach that integrates the algorithm for selection with the process for binding. The integration is used to preserve a large degree of choice freedom in composing responses to the request subject to the success of binding. We term this approach **Combined Selection and Binding** or "finding and binding".

In the following section, we describe the information store and algorithmic elements needed to make a clear comparison between these approaches to enabling grid applications to make synchronous use of complex resource structures in competitive

Host	Cluster	TightBag
Host1	Host1, Host2	Host1, Host2, Host3
Host2		
...		
Host5		

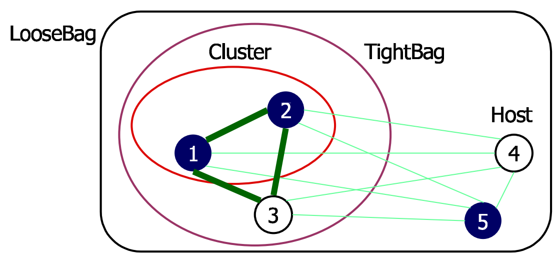


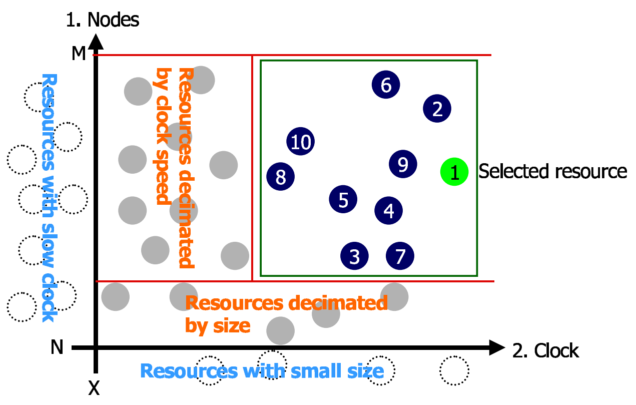
Figure 3. Example of Resource Classification: Host1, Host2, and Host5 share the same type. Host3 and Host4 share the same type. Thick lines represent fast communication connections and thin lines represent poor communication connections.

resource environments. Naturally, the best solution would be a system that responds quickly (even in very large grids) with high quality solutions, and never fails to satisfy a request. While this ideal is probably unachievable, it does identify the desirable attributes of scalability, quality in solution, and tolerance of resource competition.

In the following sections, we take the following steps to lay the groundwork for a reasoned comparison. First, we describe an information store structure which allows efficient implementation of the vgDL language and several algorithms which span the range from separate selection to combined selection and binding. This information store organization is critical to achieving scalability for the Virtual Grid system to grid resource environments of the future which are likely to contain millions if not billions of distinct resources. Second, we describe an algorithm which uses the information store to implement **separate selection**. This algorithm exploits the highly developed relational database technology to return responses to requests that are highly ranked, according to the application-supplied ranking function. Third, we describe an algorithm for **combined selection and binding.** This algorithm integrates these two traditionally separate activities, preserving choice freedom in the final selection subject to binding by maintaining multiple candidates for each resource request subcomponent which can then be flexibly composed into multiple solutions.

3.1 Organizing Resource Information

Because retrieving resource information from information services such as MDS [23] interactively to satisfy requests is prohibitively expensive, we take a proactive approach, retrieving and caching a copy of the available grid resource information locally in a relational database, and exploiting the power of relational database technology (indexing, data organization, query optimization, etc.) to compose high quality responses to resource requests quickly. To support this model, vgES uses vgAgent to periodically retrieve resource information from information services and stores it in a local vgFAB database. It turns out that even for large grids, large numbers of requests can be supported by a single database, and if needed multiple database systems can be deployed.



1. Nodes
M
N
X
2. Clock
Resources decimated by clock speed
Resources with slow clock
Resources decimated by size
Resources with small size
Selected resource
1
2
3
4
5
6
7
8
9
10

Figure 4. Example of resource selection: vgDL = ClusterOf (host) [N:M] { host = [Clock >= X][Rank = Clock] }

Our information organization is designed to support rapid identification of a good solution for each vgDL resource request, and to allow the quality of that solution to have "quantifiable quality". That is, we can describe quantitatively how compromises for scalability and speed affect the number of better solutions that might be missed.

The critical organization for resource information is a *classification* for the resources that relates to the resource aggregates that occur in vgDL (see Section 2). This classification accelerates the selection process because it captures the critical relationships of resources to each other (communication performance). Specifically, the schema includes three tables; Host, Cluster, and TightBag. The Host table contains static and dynamic attributes of all of the individual hosts. Dynamic resource attribute includes available memory and disk size, average CPU loads, etc., while static attributes include processor type, clock speed, cache size, installed memory and disk size, IP address, host name, etc. In addition, each host entry contains information about the clusters and tightbags to which the host belongs. This Host table can be used to select loose bags. The Cluster table contains information about aggregate resource properties of a cluster, such as total amount of memory and disk, number of hosts, which hosts, etc. At present, for clusters we define homogeneity to simply be the same instruction set processor architecture. In future vgFAB implementations, broader definitions of homogeneity may well be used. The Cluster table also contains information about tightbags that the cluster may belong to. Finally, the TightBag table contains aggregate resource properties of collections of hosts that might be used to form a TightBag. In general, these are sets of resources that are tightly-coupled to each other. This information is similarly to that in the Cluster table, except that it does not have the processor architecture information as an attribute of homogeneity.

Figure 3 illustrates an example of resource information in the vgES information store (thicker lines in the figure indicate better network connectivity, and different node colors indicate heterogenity). For instance, host1 and host2 are grouped as a cluster because they are homogeneous and network performance between them is good. Meanwhile, host1, host2, and host3 are grouped as a tight bag because host3 has good network connectivity to the cluster even though they are heterogeneous. Singleton hosts either are part of a tight bag or exist as independent entities in the implied loose bag that includes all resources.

Classification of resources in a production grid can be achieved by independent (localized or distributed) agents that identify resource classes using the network and host information services systems such as the Network Weather Service (NWS) [24], the (Monitoring and Discovery Service) MDS [23], or from de-facto resource configurations such as a cluster managed by a batch scheduler. Our current implementation of the vgAgent performs the classification from these information services, and is localized.

3.2 Separate Selection

To implement a separate selection that identifies good solutions according to the user-defined ranking function, we can use a simple algorithm.

Separate Selection Algorithm

1. Filter out infeasible candidates using the constraints on hosts and aggregates
2. Evaluate the rank function and default preferences based on resource and aggregate attributes for each candidate, and compose them into overall request solutions
3. Sort the overall solution candidates from "best" to "worst" according to the rank function computed on the composed solutions, and return either the best, or the best "k" solutions, as appropriate. For efficiency, since only a few responses are needed, many intermediate results can be truncated.

This algorithm can be implemented quite efficiently with the information store organization indicated in Section 3.1 for requests expressed in the vgDL language.

Figure 4 illustrates an example of how to select candidates for a cluster of hosts, with clock rate on the x-axis, and cluster size on the y-axis. The cluster size must be in between N and M, and as high as possible. The clock rate must be greater than X, and as high as possible. First, vgFAB filters out candidates using the range of size [N:M], sorts the candidates by their sizes, and decimates a portion of feasible candidates with small size (at the bottom of the graph). vgFAB repeats the same process on the remaining candidates using the clock speed for filtering, and decimating a portion of the remaining candidates (on the left side of the graph). In this example, vgFAB finishes with ten candidates. If only a single solution is desired, it would be the highest rank solution, #1 in Figure 4.

The above selection algorithm is implemented directly using the information store schema described in Section 3.1 and queries against that relational database. First, vgFAB builds a parse tree corresponding to the vgDL description and simplifies the parse tree using simple rules that we described in [22] (e.g., a Cluster of Clusters is a Cluster). Next, vgFAB implements the resource selection algorithm by synthesizing a sequence of SQL queries to identify a set of candidates. vgFAB filters out infeasible candidates using a `WHERE` clause with the constraints of individual hosts and aggregators. It then evaluates the ranking function and default preferences declaring a dynamic attribute by an `AS` clause and sorting the records in a descending order with respect to this dynamic attribute. Truncation to reduce the number of solutions is done with a `LIMIT` clause. When a candidate has been found vgFAB annotates the vgDL parse tree with resource information, and binds the resources as described in the next section. Finally, vgFAB returns a VG instance.

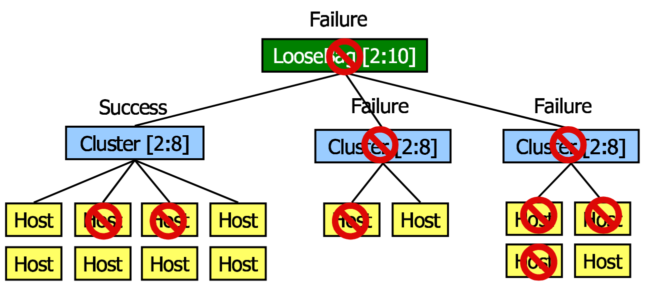


Figure 5. Combined Selection and Binding: Initial Failure

vgDL = LooseBagOfCluster = LooseBagOf (Clusters) [2:10] { Clusters = ClusterOf (host) [2:8] { host = […] } }

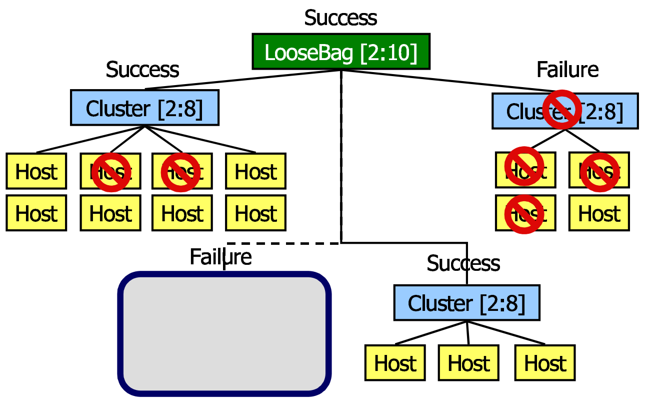


Figure 6. Combined Selection and Binding: Success

3.3 Combined Selection and Binding

Combining selection and binding requires a slightly different algorithm. Here the approach is to integrate these two steps, preserving selection flexibility into the binding phase to tolerate binding failures which may arise from resource competition, change in access, etc.

Combined Selection and Binding Algorithm

1. Filter out infeasible candidates using the constraints on hosts and aggregates
2. Evaluate the rank function and default preferences based on resource and aggregate attributes for each candidate, and collect a set of highly ranked candidates for each component of the vgDL request
3. For each component in the vgDL request, attempt to bind the highest ranked candidate. If succeed, go to next step. If not, continue with the next highest ranked candidate until we succeed. If the candidates are exhausted, this vgDL request has failed.
4. Take the bound resource for each component of the vgDL request, and combine them into a Virtual Grid (VG). Build the corresponding data structures to represent their role and relationship in the VG and return them to the application.

Note that by keeping track of multiple candidates per component of a VG specification (i.e., Cluster, TightBag, LooseBag, and Host) this algorithm preserves selection flexibility into the binding phase. It is easy to preserve such flexibility for requests expressed in vgDL, but not for resource requests from all resource specification languages. This "sub-candidate" diversity

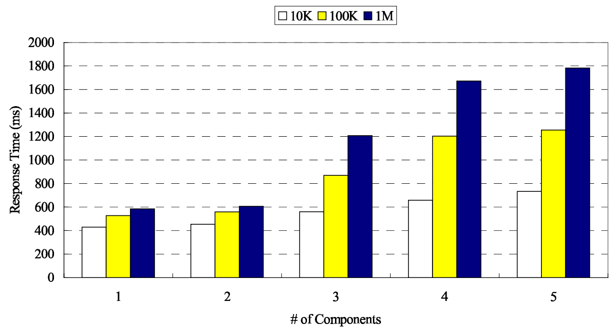


Figure 7. vgFAB Selection time vs. Request complexity and Grid size.

is a key to high binding success: binding a complex resource set at once is difficult in a competitive resource environment, while incremental binding component-by-component is significantly easier.

Figure 5 and 6 depict a resource selection and binding scenario in which an application has requested a LooseBag of Cluster. vgFAB found a number of candidates for the clusters and the highest ranked candidates has 8 hosts, 2 hosts, and 4 hosts, respectively as shown in Figure 5, satisfying the user requirements. However, binding failed on the two right-most candidates. If a second candidate exists for the 2-host cluster, the next highest ranked candidate can be selected, with vgFAB successfully binding it and satisfying the overall vgDL request.

4. EVALUATION

In this section, we study the combined resource selection and binding scheme, evaluating it for scalability, quality of results, and success rate in competitive resource environments. In the following subsections, we describe the experimental methodology used, followed by detailed discussion of the evaluation results in each dimension.

4.1 Methodology

All of our experiments are performed using resource information generated by state-of-the-art tools for generating representative grid resource and network environments (a statistical grid resource generator [28] and the BRITE network topology generator [29]). Both of these generators represent the current state of research knowledge and the use of synthetic resource environments enables us to perform experiments that explore behavior in resource environments far larger (1,000,000 resources, 100x larger than any enterprise server grid of which we are aware) than those for which actual information in available. In the following experiments, we consider grid environments of 10,000, 100,000 and 1M resources.

Experiments with vgFAB are run on a modest machine, an IBM Thinkpad X31 with a single 1.4 Ghz Pentium III processor and only 512MB of memory. All the test application that submits the vgDL requests and the vgFAB including the database system run on this single machine.

An important element of all of the experiments is the workload of vgDL requests used. We generated a range of vgDL expressions, varying the terms used, the depth and breadth of the expressions, and then classify their use in each of the experiments based on the most relevant characteristics.

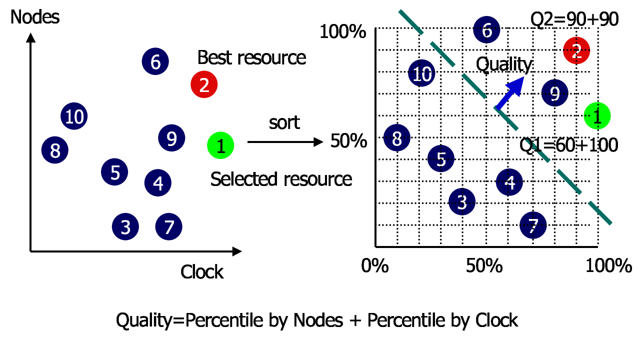


Figure 8. Example of quality evaluation: vgDL = ClusterOf (host) [N:M] { host = [Clock >= X][Rank = Clock] }

For the scalability experiments, we measure selection time from when the request is submitted to when the response is received. For each vgDL complexity (# of components), we used 100 different vgDL requests.

For the selection quality experiments, we compute a quantitative metric, a multi-dimensional percentile score, which indicates how the response compares according to the user-defined rank function to the larger population of responses that meet the requirements of the request.

For the comparison of separate selection and combined selection and binding, we use 100 trials of the same vgDL request to generate each result. Because the binding success is a random trial, each of these 100 trials could potentially produce a different result.

4.2 Selection in Large Grids (Scalability)

We characterized the scalability of selection algorithm and our vgFAB prototype system by studying response time required to perform selection as a function of the grid resource population and the vgDL request complexity. Grid resource population was varied from 10,000 hosts to 100,000 hosts and then to 1,000,000 hosts. vgDL requests varied in depth of nesting of aggregates and in breadth. We characterize the complexity of a request by its number of components, varying from one to five in these experiments. The number of components is a good measure of complexity because it corresponds both to the number of entities the application developer thinks about and the separate resources that must be identified by the vgFAB. The vgDL requests range from simple descriptions with single entities to complex ones with nested components as well as network connectivity and are described in detail in Appendix A. In each case, we computed average response times over 100 trials.

Our experiments show that selection times increase with grid resource population and vgDL request complexity (see Figure 7). In all cases, the selection times are short – no more than two seconds --, due to the appropriate data organization and mature database technology. We have also designed vgFAB to avoid expensive join operations. Based on analysis of the vgFAB implementation, we would expect selection time to increase roughly linearly with vgDL request complexity and nlog(n) in grid resource population. However, because this only a first implementation, we leave a more detailed analysis for future work.

4.3 Selection Result Quality

A fundamental question for resource selection is how the quality of the results compares to both the ideal (optimal) and the population of responses that meet the minimum requirements of the request. Most selection systems [14,16] provide no assurances of quality to users. In our system, each vgDL request contains a rank function, defining a near total order on the possible request responses. In this section, we evaluate our vgFAB implementation by comparing how the returned solution fares by rank function, when compared to the population of allowable responses in the vgFAB grid resource population. When no rank function is defined, we use our default ranking (i.e. faster CPU's are better, more memory is better, more CPU's are better, etc.) as defined in [22].

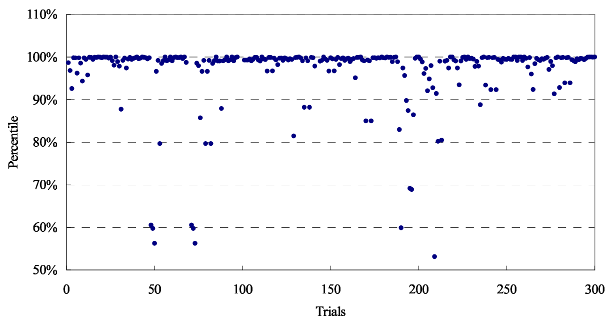


Figure 9. Quality of selected resources compared to the optimal one

These rank functions and defaults generate multiple dimensions of ranking (each induces a dimension), so we need a regular means to combine them into a single metric. For instance, assume that the user is interested in clusters that have processors with clock speed higher than X, and with a number of processors between N and M. With the resource selection algorithm described in Section 3.2, vgFAB selects candidates as shown in the left side of Figure 8. From the user's perspective, a cluster with higher clock speed and bigger size is the better.

We define the combination of percentile ranks in each dimension as an overall ranking on the population of possible solutions (a definition of quality). Specifically, we define quality as $Q(R) = \frac{\sum_i^n p_i}{n}$, where p_i is the percentile of resource R when the resources are sorted according to the ith rank function. For example, in Figure 8, the quality value of resource 2 is (90+90)/2 = 90 (since resource 2 is in the 90th percentile in both dimensions) and that of resource 1 is (60+100)/2 = 80. Note that because vgFAB uses simpler approximate combination functions, and often truncates intermediate results for efficiency, it may select resource 1 although resource 2 might ultimately have a higher quality value.

Our evaluation of the vgFAB for quality of results shows the quality function values (see Figure 9). The results show that in general, vgFAB does a good job of returning high quality results. In fact, in most cases, vgFAB returns a solution which is 90th percentile quality in the solution population. In all cases, the returned value is above the 50th percentile for quality. When vgFAB returns a lower-ranked solution, it is typically because the vgFAB heuristic of considering only one ranking functions at a time and then composing the results finds a local optimum, not a global one. In some cases, when the results fall below 80th percentile, this is because there are only a small number of candidate solutions in the grid resource population.

4.4 Selection and Binding in Competitive Resource Environments

We study a basic comparison between the two algorithmic approaches outlined in Section 3 – **Separate Selection** and **Combined Selection and Binding** in a simulated competitive resource environment. In general, binding can fail for any number of reasons, including competitive use by other applications, inaccurate information, authentication failure, etc. Of these, the dynamic properties due to competition for resources are of greatest interest here. As binding success rates fall due to increased competition, both approaches can use multiple selection to increase success rates for entire vgDL requests. As described in Section 3, the Separate Selection approach can collect multiple candidate solutions (for the entire request). The Combined Selection and Binding approach collects multiple candidates for each subpart of the solution, composing the successfully bound subparts to create an overall successful response.

We simulate resource competition with a probability of binding failure (and success). The same probability is used for all resources in the grid, reflecting a level of competition for resources. Each attempt to bind a resource constitutes an independent random trial (and if you fail to bind, successive trials will not succeed – the resource is busy). If any of the parts of a Separate Selection candidate have a binding failure, that entire candidate fails. If all of the candidates developed by the Separate Selection algorithm fail, then the Separate Selection algorithm fails for this request. For the Combined Selection and Binding algorithm, if there is at least one binding success for each of the vgDL request components, the bound resources can be tied together into a successful response to the request. We vary vgDL request complexity, increasing the number of components across a range from one to sixteen (see Appendix A for more information). The vgDL request complexity is a critical factor in the comparison between Separate Selection and Combined Selection and Binding.

Our first set of results explores resource request success rates for relatively simple vgDL requests with one to four components (see Figures 10-13). As indicated above, the trials are based on grid resource populations of one million hosts and 100 repeats of each vgDL request. The graphs in turn consider 2, 4, 8, and 16 candidates selected by both algorithms. In all cases, the Combined Selection and Binding approach has an equal or greater success rate than the Separate Selection approach. The major benefit of the Combined approach appears at high levels of resource competition and more complex requests. For example, for the simple descriptions, Combined tolerates a 10% higher binding rate, allowing a much higher practical resource utilization while supporting synchronous use of grid resources. For complex requests of three to four components, Combined tolerates as much as a 20-30% higher binding failure rate, allowing dramatically higher resource competition to be compatible with synchronous grid-style use of resources even at resource utilizations of 60% and 70%.

In contrast, the success rates of Separate Selection drop quickly as query complexity or resource competition increase. For example, Separate Selection never successfully binds resources for requests with > four components and binding failure rates above 40%, a moderate resource utilization.

Our second set of experiments compare the performance of Separate Selection and Combined Selection and Binding for more complex vgDL descriptions, with the number of components ranging from one to sixteen (see Figures 14-17). In these experiments, we see similar trends, and a remarkable increasing difference in performance between the approaches is as the number of candidates is increased. For instance, Figure 16 shows that with eight candidates, Combined Selection and Binding can satisfy very complex requests (eight to sixteen components) at resource utilization rates of 60 – 70%. In contrast, the Separate Selection approach never succeeds for these complex requests for resource utilizations greater than 30%. Even more extreme, in Figure 17, Combined Selection and Binding is able to more than double the resource utilization (from 30% to 70%) at which complex vgDL descriptions with eight and sixteen components have some chance of being satisfied. This difference will make a major difference in views of the compatibility of efficient local resource management, a synchronous grid resource use across multiple resource managers.

Another observation is that the number of candidates contributes to high success rates of binding. We increase the number of candidates from 2 to 16 in Figures 10-17. Larger numbers of candidates provide better success rates. Overall, per-component binding scheme takes advantage of multiple candidates much more than Separate Selection binding scheme because the overall success rate of Separate Selection scheme decreases by the factor of number of components.

5. DISCUSSION and RELATED WORK

Our work on Virtual Grids builds on the four-year GrADS effort [30] to research development tools for adaptive grid applications. The GrADS framework coupled performance models and monitoring on a per-application basis to support intelligent adaptation. The Virtual Grid (VGrADS) approach separates the application and runtime system concerns with an explicit, high-level resource abstraction. The Virtual Grid's high-level application-oriented approach differs from traditional low-level resource description and selection systems [16,26] that focus on individual machine characteristics. For more information on this contrast, the interested reader is directed to Section 2.

Our problem focus reflects the evolution of Grids from controlled environments to increasingly diverse, complex, multi-organizational entities [3,6]. In such systems, if synchronous use is to be achieved, the tension between good resource selection for applications and efficient resource management needs a resolution that departs from the common approach today in which resource efficiency is the dominating concern.

In traditional systems, resource description and selection are decoupled from resource binding. This differs significantly from the combined finding and binding approach explored in this paper. For example, Globus' RSL [26] and the later Redline [14] system focus only on description and selection. Resource binding is cleanly (and explicitly) separated into a resource access protocol. Systems such as Condor and its many evolutionary descendants (Condor-G [32], DAGMAN [33], etc.) explicitly chose to separate single resource selection (matchmaking [16]) and multi-resource selection (gangmatching [18]) from the act of resource binding that is separately controlled by the Condor Resource Manager. Many other systems for resource description and selection [19,21] similarly separate selection from resource binding. There are a few distributed "ad hoc" approaches such as used in SpiderNet

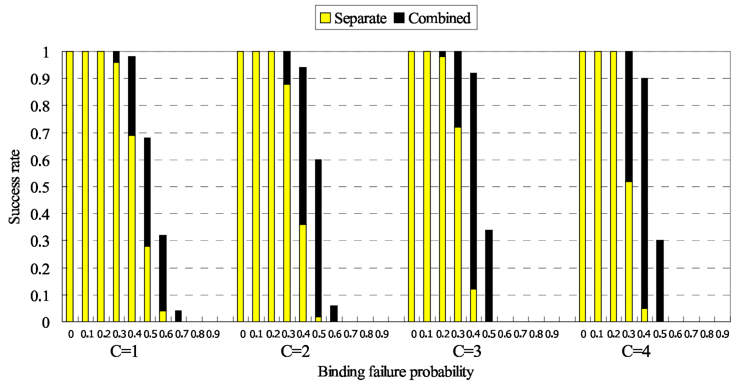


Figure 10. Success rates of binding resources (C: the number of components, # of candidates = 2)

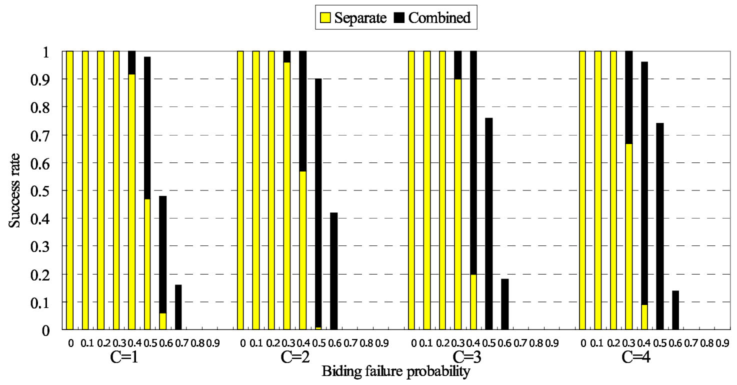


Figure 11. Success rates of binding resources (C: the number of components, # of candidates = 4)

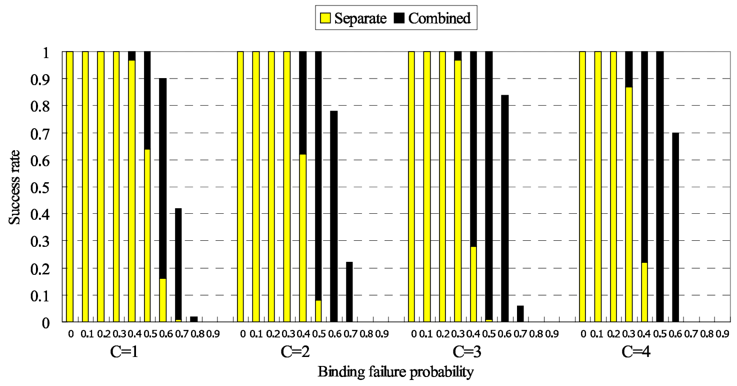


Figure 12. Success rates of binding resources (C: the number of components, # of candidates = 8)

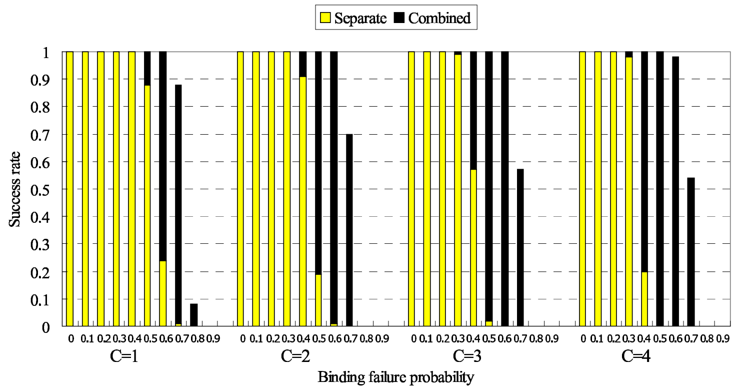


Figure 13. Success rates of binding resources (C: the number of components, # of candidates = 16)

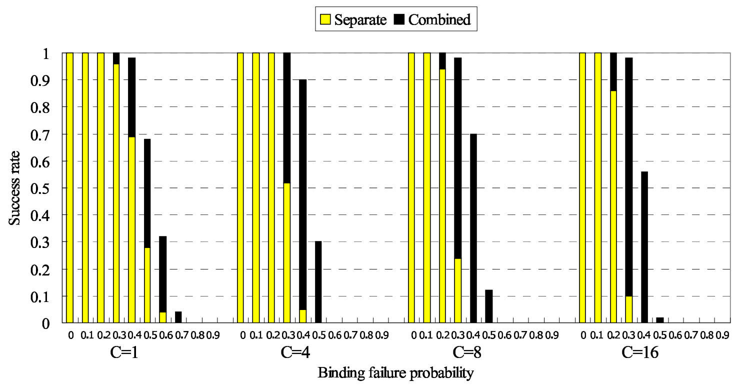


Figure 14. Success rates of binding resources (C: the number of components, # of candidates = 2)

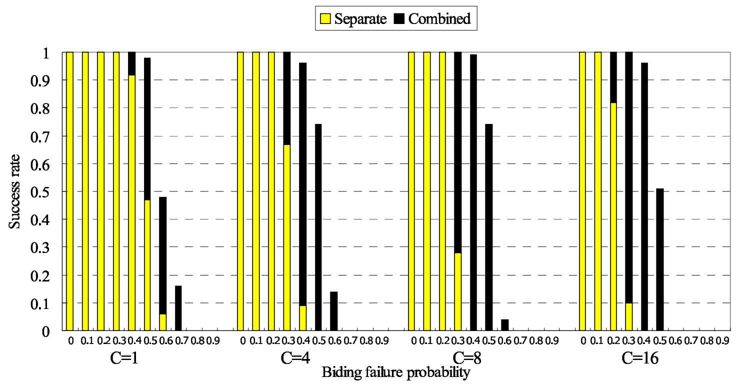


Figure 15. Success rates of binding resources (C: the number of components, # of candidates = 4)

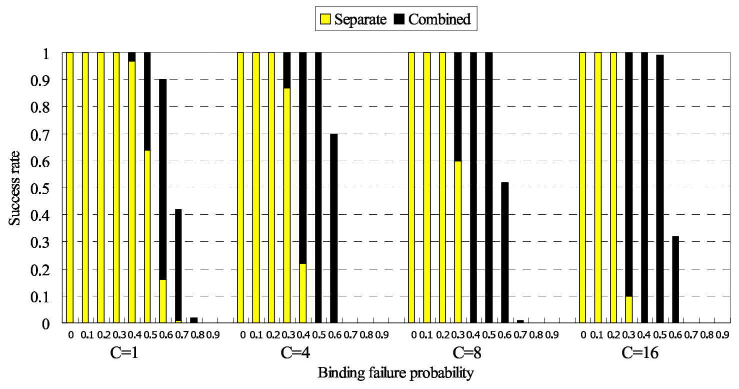


Figure 16. Success rates of binding resources (C: the number of components, # of candidates = 8)

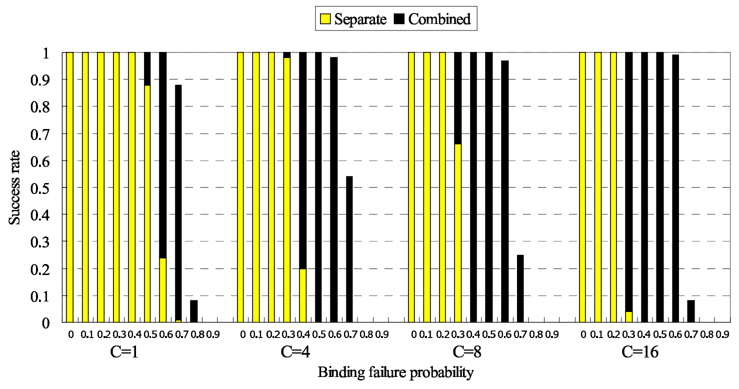


Figure 17. Success rates of binding resources (C: the number of components, # of candidates = 16)

[20] that perform resource selection and binding in an integrated fashion). However, these systems cannot provide any strong characterization of the quality of the achieved results.

Our work supports the synchronous use of grid 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 [12] implement the co-scheduling or binding of resources (and the specific resource managers) 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. The vgFAB approach of integrated selection and allocation is strikingly different from these architectures, which separate selection from binding.

Finally, on 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, and believe it is also promising. However, it depends on the widespread adoption of advance reservation in resource managers. This is an eventuality long desired by the Grid research community, but not yet realized in the configuration of production resources.

6. SUMMARY & FUTURE WORK

We have described the Virtual Grid system, as realized in the vgES, which enables applications to describe and manage resources conveniently via application-level abstractions. A key element of the vgES system is the vgFAB, a component that performs resource selection and binding. A key element of this approach is Combined Selection and Binding which exploits the flexibility of vgDL descriptions to produce high quality solutions to complex application requests, and to reliable produce solutions even in environments with high levels of resource competition.

We perform a range of experimental evaluations of vgFAB and the Combined Selection and Binding approach. This integrated "finding and binding" is demonstrated to scale to grid resource populations of millions of hosts, identifying good matches in few seconds. This good performance is achieved by use of a sophisticated resource classification system and efficient relational database technology in vgFAB. We also evaluated the quality of solutions returned by the vgFAB prototype. Our results show that in general, vgFAB returns solutions that are in the top 10% of satisfying solutions.

We also perform extensive comparisons of Combined Selection and Binding to the traditional Separate Selection. These experiments explore a range of request complexity and levels of resource competition. The results show that Combined Selection and Binding not only dramatically increases success rates at high resource utilizations (binding failure rates), it does so with modest numbers of candidates. In short, Combined Selection and Binding can double the resource utilization (from 30% to 70%) at which synchronous resource allocation and use across as many as sixteen resource managers is possible.

Although the current vgFAB prototype is localized, the vgES architecture does not preclude distributed implementation, and are promising. 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, agents that collect resource information could be distributed like web crawlers. Finally, multiple vgFABs could cooperate to distribute requests from users. Since several design choices are available, we need to explore them for better performance. In the meantime, we have implemented a vgES prototype, which is being used to exploit various research issues in the areas of scheduling, fault tolerance, and application deployment on large-scale Grid platforms.

7. 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. Support from the Ministry of Information and Communication, Republic of Korea is also gratefully acknowledged.

The authors acknowledge the efforts of Richard Huang, Dionysios Logothetis, Jerry Chou, and Ken Yocum to build the vgES system used in these experiments, and also the members of the VGrADS team whose research insights and experience have provided invaluable help.

8. REFERENCES

[1] Geoffrey Fox and David Walker, e-Science Gap Analysis, UK e-Science Technical Report, UKeS-2003-0, http://www.nesc.ac.uk/technical_papers/uk.html, Jun. 2003.

[2] Japan's National Research Grid Initiative (NAREGI), Asian Technology Information Program Report, ATIP03.016, http://www.atip.org/REPORTSMATRIX/public/year2003_total.html, Dec. 2003.

[3] Butterfly.net Inc., Butterfly Grid Solution for Online Games, http://www.butterfly.net, 2003.

[4] Paul Avery and Ian Foster, iVDGL Annual Report for 2002-2003, Aug. 2003, Available from http://www.ivdgl.org.

[5] Grid2003 - The Grid2003 Production Grid: Principles and Practice. IEEE HPDC'04, June 2004.

[6] Charlie Catlett, The TeraGrid: A Primer, http://www.teragrid.org/about, Sep. 2002.

[7] M. J. Litzkow, M. Livny, and M. W. Mutka. *Condor - a hunter of idle workstations*. In Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104--111, 1988.

[8] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams, S. Koranda, GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists, High Performance Distributed Computing, 2002. HPDC-11 2002. Page(s): 225 - 234

[9] W. Li, R. Byrnes, J. Hayes, V. Reyes, A. Birnbaum, A. Shahab, C. Mosley, D. Pekurovsky, G. Quinn, I. Shindyalov, H. Casanova, L. Ang, F. Berman, M. Miller, P. Bourne. The Encyclopedia of Life Project: Grid Software and Deployment. Special Issue on Grid Systems for Life Sciences. New Generation Computing.

[10] Available from http://www.cs.wisc.edu/vdt//index.html

[11] Oracle Grid Computing, White Paper, Feb. 2005, Available from http://www.oracle.com/technologies/grid/index.html

[12] Resource Co-Allocation in Computational Grids. K. Czajkowski, I. Foster, and C. Kesselman. Proceedings of the

Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.
[13] Globus: A Metacomputing Infrastructure Toolkit. I. Foster, C. Kesselman. Intl J. Supercomputer Applications, 11(2):115-128, 1997.
[14] C. Liu and I. Foster, A Constraint Language Approach to Matchmaking. RIDE'04, Boston, 2004.
[15] A. Huang, and P. Steenkiste, Building Self-configuring Services Using Service-specific Knowledge, IEEE HPDC'04
[16] R. Raman, M. Livny, and M. Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, IEEE HPDC'98, pp. 140-147, July 1998.
[17] C. Liu, L. Yang, I. Foster and D. Angulo, Design and Evaluation of a Resource Selection Framework, IEEE HPDC'02, pp. 63-72, July 2002.
[18] R. Raman, M. Livny, M. Solomon, Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching, IEEE HPDC'03, pp. 80-89, June 2003.
[19] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. UC Berkeley Technical Report UCB//CSD-04-1334, July 2004.
[20] X. Gu, K. Nahrstedt, and B. Yu, Spidernet: An Integrated Peer-to-peer Service Composition Framework, IEEE HPDC'04
[21] T. Kichkaylo, A. Ivan, and V. Karamcheti, Constrained Component Deployment in Wide-area Networks Using AI Planning Techniques, IEEE IPDPS'03, Apr 2003.
[22] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, A. A. Chien, Efficient Resource Description and High Quality Selection for Virtual Grids, ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID'05), May 2005, Cardiff, United Kingdom.
[23] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid Information Services for Distributed Resource Sharing, IEEE HPDC'01, Aug. 2001.
[24] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, FGCS 15(5-6): 757-768, Oct. 1999.
[25] F. Sacerdoti, M. Katz, M. Massie, D. Culler, Wide Area Cluster Monitoring with Ganglia, IEEE Cluster, Dec 2003.
[26] A Resource Management Architecture for Metacomputing Systems. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.
[27] A. Chien, H. Casanova, Y.-S. Kee, and R. Huang. The Virtual Grid Description Language: vgDL. University of California, San Diego, Department of Computer Science and Engineering Technical Report CS2005-0817. Available from http://www.cs.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2005-0817
[28] Y.-S. Kee, H. Casanova, and A. A. Chien, Realistic Modeling and Synthesis of Resources for Computational Grids, ACM/IEEE SC'04, Nov. 2004
[29] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. International Workshop on Modeling, MASCOTS'01, Aug. 2001.
[30] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. International Journal of High-Performance Computing Applications, 15(4), 327-344
[31] Condor-G: A Computation Management Agent for Multi-Institutional Grids. J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
[32] DAGMan Metascheduler, Available from http://www.cs.wisc.edu/condor/dagman
[33] Alain Roy and Volker Sander, Grid Resource Management: State of the Art and Future Trends, pages 377-394, Fall 2003
[34] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Jim Pruyne, John Rofrano, Steve Tuecke, Ming Xu, Web Services Agreement Specification (WS-Agreement), version 1.1, May 2004.

Appendix A

In our experiments, we have used about 50 unique vgDL resource descriptions, generated from a variety of activities, including application design, system test, and general brainstorming. Because there isn't space to include all of the descriptions, we include one sample of each level of complexity (number of components) for illustrative purposes.

- one_component = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [(Processor == Pentium4) && (Memory>=4096)] } }

- two_component = rsc1 = LooseBagOf(c)[2:4] { c = ClusterOf(node)[4:8] { node = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(tb)[2:4] { tb = TightBagOf(node)[4:8] { node = [Clock>=2.048] } }

- three_component = g1 = { rsc1 = LooseBagOf(c1)[2:4] { c1 = ClusterOf(node1)[4:8] { node1 = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(tb1)[2:4] { tb1 = TightBagOf(node2)[4:8] { node2 = [Clock>=2.048] } } } **FAR** rsc3 = LooseBagOf(c2)[2:4] { c2 = ClusterOf(node3)[4:8] { node3 = [Processor == Pentium4] [Rank = Memory] } }

- four_component = g1 = { g2 = { rsc1 = LooseBagOf(c1)[2:4] { c1 = ClusterOf(node1)[4:8] { node1 = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(tb1)[2:4] { tb1 = TightBagOf(node2)[4:8] { node2 = [Clock>=2.048] } } } **FAR** rsc3 = LooseBagOf(c2)[2:4] { c2 = ClusterOf(node3)[4:8] { node3 = [Processor == Pentium4] [Rank = Memory] } } **FAR** rsc4 = LooseBagOf(c3)[2:4] { c3 = ClusterOf(node4)[4:8] { node4 = [(Processor == Pentium4) && (Memory>=4096)] [Rank = Clock * Memory] } }

- five_component = g1 = { g2 = { rsc1 = LooseBagOf(c1)[2:4] { c1 = ClusterOf(node1)[4:8] { node1 = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(tb1)[2:4] { tb1 = TightBagOf(node2)[4:8] { node2 = [Clock>=2.048] } } } **FAR** rsc3 = LooseBagOf(c2)[2:4] { c2 = ClusterOf(node3)[4:8] { node3 = [Processor == Pentium4] [Rank = Memory] } } } **FAR** g3 = { rsc4 = LooseBagOf(c3)[2:4] { c3 = ClusterOf(node4)[4:8] { node4 = [(Processor == Pentium4) && (Memory>=4096)] [Rank = Clock * Memory] } } **FAR** rsc5 = LooseBagOf(tb2)[2:4] { tb2 = TightBagOf(node5)[4:8] { node5 = [(Clock>=2.024) && (Memory>=4096)] [Rank = Memory] } } }

- eight_component = group = { group = { rsc3 = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(clt)[2:4] { clt = TightBagOf(node)[4:8] { node = [Clock>=2.048] } } } **FAR** rsc3 = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [Processor == Pentium4] [Rank = Memory] } } } **FAR** group = { group = { rsc3 = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [(Processor == Pentium4) && Memory>=4096)] [Rank = Clock * Memory] } } **FAR** rsc4 = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [(Processor == Pentium4) || (Processor == Pentium3)] [Rank = Memory] } } } **FAR** group = { group = { rsc3 = LooseBagOf(clt)[2:4] { clt = ClusterOf(node)[4:8] { node = [(Processor == Pentium3) && (Memory>=1024)] } } **FAR** rsc2 = LooseBagOf(clt)[2:4] { clt = TightBagOf(node)[4:8] { node = [Clock>=2.048][Rank=Memory] } } } **FAR** clt = ClusterOf(node)[8:32] { node = [Processor == Pentium3] [Rank = Memory] } } }

- sixteen_component = g1 = { g2 = { g3 = { rsc1 = LooseBagOf(c1)[2:4] { c1 = ClusterOf(n1)[4:8] { n1 = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc2 = LooseBagOf(tb1)[2:4] { tb1 = TightBagOf(n2)[4:8] { n2 = [Clock>=2.048] } } } **FAR** rsc3 = LooseBagOf(c2)[2:4] { c2 = ClusterOf(n3)[4:8] {n3 = [Processor == Pentium4] [Rank = Memory] } } } **FAR** g4 = { g5 = { rsc4 = LooseBagOf(c3)[2:4] { c3 = ClusterOf(n4)[4:8] { n4 = [(Processor == Pentium4) && (Memory>=4096)] [Rank = Clock * Memory] } } **FAR** rsc5 = LooseBagOf(c4)[2:4] { c4 = ClusterOf(n5)[4:8] { n5 = [(Processor == Pentium4) || (Processor == Pentium3)] [Rank = Memory] } } } **FAR** g6 = { g7 = { rsc6 = LooseBagOf(c5)[2:4] { c5 = ClusterOf(n6)[4:8] { n6 = [(Processor == Pentium3) && (Memory>=1024)] } } **FAR** rsc7 = LooseBagOf(tb2)[2:4] { tb2 = TightBagOf(n7)[4:8] { n7 = [Clock>=2.048][Rank=Memory] } } } **FAR** c6 = ClusterOf(n8)[8:32] { n8 = [Processor == Pentium3] [Rank = Memory] } } } } **FAR** g8 = { g9 = { g10 = { rsc9 = LooseBagOf(c7)[2:4] { c7 = ClusterOf(n9)[4:8] { n9 = [(Processor == Pentium4) && (Memory>=4096)] } } **FAR** rsc10 = LooseBagOf(tb3)[2:4] { tb3 = TightBagOf(n10)[4:8] { n10 = [Clock>=2.048] } } } **FAR** rsc11 = LooseBagOf(c8)[2:4] { c8 = ClusterOf(n11)[4:8] { n11 = [Processor == Pentium4] [Rank = Memory] } } } **FAR** g11 = { g12 = { tb4 = TightBagOf(n12)[8:32] { n12 = [(Processor == Pentium4) && (Memory>=4096)] [Rank = Clock * Memory] } **FAR** c9 = ClusterOf(n13)[8:32] { n13 = [(Processor == Pentium4) || (Processor == Pentium3)] [Rank = Memory] } } **FAR** g13 = { g14 = { rsc14 = LooseBagOf(c10)[2:4] { c10 = ClusterOf(n14)[4:8] { n14 = [(Processor == Pentium4) && (Memory>=8192)] } } **FAR** rsc1 = LooseBagOf(tb5)[2:4] { tb5 = TightBagOf(n15)[4:8] { n15 = [(Memory>=4096) && (Clock>=2.048)][Rank=Clock] } } } **FAR** rsc16 = LooseBagOf(n16)[8:32] { n16 = [Processor == Pentium3] [Rank = Memory] } } } } }