Efficient Resource Description and High Quality Selection for Virtual Grids

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

ABSTRACT

Simple resource specification, resource selection, and effective binding are critical capabilities for Grid middleware. We describe the Virtual Grid, an abstraction for providing these capabilities complex resource environments. Elements of the Virtual Grid include a novel resource description language (vgDL) and a resource selection and binding component (vgFAB), which accepts a vgDL specification and returns a Virtual Grid, that is, a set of selected and bound resources. The goals of vgFAB are efficiency, scalability, robustness to high resource contention, and the ability to produce results with quantifiable high quality. We present the design of vgDL, showing how it captures application-level resource abstractions using resource aggregates and connectivity amongst them. We present and evaluate a prototype implementation of vgFAB. Our results show that resource selection and binding for virtual grids of 10,000's of resources can scale up to grids with millions of resources, identifying good matches in less than one second. Further, these matches have quantifiable quality, enabling applications to have high confidence in the results. We demonstrate the effectiveness of our combined selection and binding approach in the presence of resource contention, showing that robust selection and binding can be achieved at moderate cost.

1. INTRODUCTION

While the success and acceptance of Grids continues in the scientific and commercial computing communities [1-3], the technology required to achieve the ambitious grid vision -- flexible, adaptive applications running on large federated and shared resource collections -- is still in its infancy. In fact, the applications in most widespread use on production grid environments (see [4,5]) correspond to loosely-coupled parallel applications, asynchronous workflows, and multi-tier web-application server-database applications. Applications that require tightly-coupled coordination, high performance data movement coordinated with computation, and real-time coupling of instruments remain difficult to design, implement, and manage at runtime. These types of applications are the focus of the research described in this paper; they typically require sophisticated synchronized resource use and management of failures, and their performance depends strongly on the resources used. To achieve their mission goals, these applications need to be resource-aware, but given the current complexity of grid resource environments and their likely future growth to millions, even billions of devices, embedding understanding of the resource environment into each application is impractical.

Supporting demanding grid applications requires effective solutions to four key resource problems: (i) *description*, which specifies the needed resources, (ii) *discovery*, which finds candidate resources, (iii) *selection*, which chooses from the candidates, typically based on static and dynamic resource information, and (iv) *binding*, which allocates resources via negotiation with autonomous resource managers.

We propose a new integrated approach, based on the notion of a *virtual grid* that provides a unified framework for solving all these resource problems. Our approach differs from a wide range of research efforts that address either one or a subset of these problems [6-18]. The virtual grid consists of four key elements: vgDL – an application-oriented language used to specify resource needs, vgFAB – a "finder and binder" that performs integrated resource selection and binding, vgLAUNCH – a scalable application launcher that initiates the application on the bound resources, and vgMON – a distributed monitoring service that ensures that resource expectations are realized throughout application execution. While all four of these elements pose an interesting research challenges, the scope of this paper is limited to the design of vgDL, a prototype implementation of vgFAB, and an initial evaluation of the implementation. Specific contributions of this paper include:

1. **A new framework for application resource awareness called the Virtual Grid:** Virtual grids provide an abstraction of complex grid resource, enabling optimized resource selection, management, and adaptation with robust behavior and a simple, uniform interface.
2. **Design of the vgDL resource description language:** vgDL is a human-readable, structured, hierarchical language in which applications specify their resource needs. It is based on typical resource abstractions used by application developers to manage and optimize performance (or other application execution attributes). vgDL is specifically designed to enable efficient resource selection and binding in large-scale, shared resource environments.
3. **An integrated resource "finder and binder" (vgFAB) algorithm and prototype implementation:** vgFAB combines selection and binding, identifying and allocating resource sets that match a vgDL specification. vgFAB is designed for efficiency, scalability, and robustness to resource contention.
4. **Evaluation of vgDL and of the vgFAB prototype:** Our results show that vgDL specifications in grids with millions of resources can be matched in less than a second, with

[1] Also affiliated with the San Diego Supercomputing Center

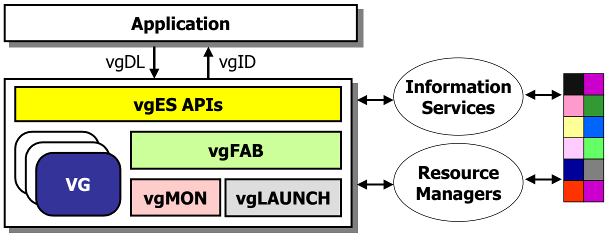


Figure 1. Virtual Grid Execution System (vgES)

obtained resource sets of quantifiable quality. Furthermore, vgFAB identifies many candidate solutions, which is key to efficient binding in the presence of resource contention.

The remainder of the paper is organized as follows. In Section 2 we introduce the notion of Virtual Grid and its architecture. Subsequently, we narrow our focus to the vgDL resource description language and vgFAB resource selection in Section 3. While many implementations are possible, one realization of the virtual grid architecture is described in Section 4, and in Section 5 we evaluate the vgDL language and the prototype. In Section 6 we discuss the implications of this evaluation and survey related work. Finally we summarize our results and describe possible directions for future work in Section 7.

2. VIRTUAL GRID ARCHITECTURE

Our work on Virtual Grids builds on and is informed by a four-year effort to build development tools for adaptive grid applications, the Grid Application Development Software Project (GrADS) [19]. The *Virtual Grid* architecture builds on a key insight from GrADS that application participation (knowledge, expectations) is needed to effectively manage performance in a dynamic grid resource environment. A major innovation in VGrADS is the attempt to couple applications and underlying grid resource management together through an application oriented resource specification (in vgDL) and an active entity (the virtual grid) which is the reification or instantiation of the application's resource environment. The application-oriented language insulates the application from the full complexity of the resource environment, and the lifecycle coupling of the application and underlying resource environment (resources and managers) enables flexible application management of resources at a high level. A Virtual Grid (VG) does not define how the application uses resources with the virtual grid, nor does it provide a functional virtualization as in a virtual machine.

The interaction of an application with the virtual grid system is illustrated in Figure 1. The Virtual Grid execution system (vgES) is realized in the following key elements:

- **vgDL** – a structured hierarchical language for application resource abstractions that is used to identify appropriate resources.
- **vgFAB** – the "finder and binder" that performs integrated resource selection and binding which in combination enable optimized resource choices in a high load resource environment with autonomous resource managers. Given a vgDL specification, vgFAB returns a "virtual grid" (**VG**) handle (i.e., an object) that communicates resource information to the application in terms of resource abstractions.

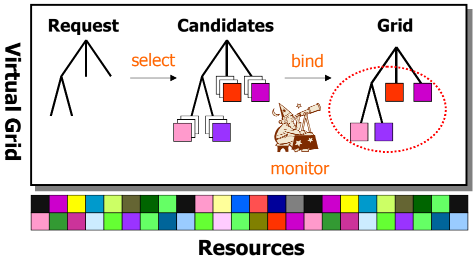


Figure 2. Virtual Grid Architecture

- **vgLAUNCH** – an application launcher that initiates the application on the bound resources.
- **vgMON** – a distributed monitoring component that ensures that resource performance expectations (i.e., vgDL specifications) are met throughout application execution.

VG's allow users to configure their own resources and change configurations according to evolving application requirements and resource conditions. A typical scenario for application and execution system interaction is as follows. First, the application specifies resource requirements in vgDL and presents them to the vgES. Then, the vgFAB finds and allocates resources satisfying the requirements and returns a Virtual Grid instance (vgID), which includes detailed static and dynamic resource information. Using the vgES API, the application can retrieve resource information from the VG instance and initiate tasks on resources using vgLAUNCH. If for some reason, resources in the VG fail to meet expectations, vgMON detects and notifies the application, which may then modify the VG, changing its resource bindings requirements.

Figure 2 depicts how a vgDL specification is turned into a VG instance. The execution system selects multiple candidates and allocates one of them, either choosing the ``best'' one according to application-specific criteria or successively trying to bind candidates until it succeeds. From a user's perspective, these selection and binding operations are hidden in the function of vgFAB. The VG thus realized goes beyond traditional resource selection [6-13]. First, the VG consists of *bound* resources that are allocated to the application and on which the application can be launched via vgLAUNCH. This avoids problems with being repeatedly unable to bind that have been selected due to high contention for resources. Once resources are bound, it is most efficient to provide a simple application launcher, ensuring that resources can be used from the moment they are bound. Second, vgMON provides runtime monitoring of the acquired resources and can facilitate adaptation to changing resource conditions either transparent or application-directed, so that vgDL specifications are met. Once a VG instance is acquired, the application can utilize the resources and use sophisticated, application-specific scheduling and load-balancing techniques to enhance performance.

The vgES system is compatible with existing and future resource information services (MDS [15], NWS [16], and Ganglia [17]) and a wide range of resource management models including best-effort, batch, slices, and even provisioned; all of which are extant in the grid today. For the remainder of this paper, we confine our discussion to the design and evaluation of vgDL and vgFAB.

```
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 "]" ]
RedlineExpr := Redline expression for attribute and constraint [see Figure 3-2]
ArithmeticExpr := Arithmetic expression in Redline for ranking function [see Figure 3-2]
```

Figure 3-1. BNF description of vgDL language

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

Figure 3-2. BNF description for Redline expression for attribute and constraint

3. APPLICATION-ORIENTED RESOURCE DESRIPTION LANGUAGE (vgDL)

A central element of the Virtual Grid approach is a resource description language (vgDL) based on application resource abstractions. One of our requirements is that this new language be human readable and make it easy for users to specify qualitative and quantitative resource requirements. If desired, this language can be easily translated into a machine-readable, interoperable representation (e.g., in XML). Based on detailed study of a number of real-world grid applications, including EMAN [20], EOL [21], LEAD/MEAD [22], GridSAT [23], as well as a large body of ad hoc experience, we identified relevant resource abstractions used by grid applications, as well as a range of typical preferences and attributes. This design rationale is discussed in detail in a technical report [24]. It is important to note that 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 3-1 and 3-2 show the BNF for the language. Note that we have incorporated the RedLine [9] BNF for attribute constraints. Key features of vgDL include

- Resource aggregates
- Network connectivity
- Composition of aggregates
- Ranking (preference)

3.1 Resource aggregates

Based on extensive study of scientific and grid computing applications, we learned that to simplify portability in design for performance (and to manage complexity), simple resource abstractions are often employed. For example, a basic aggregate resource abstraction is that of a "bag" – a heterogeneous collection of nodes – which can exhibit either "tight" or "loose" connectivity among the nodes, as used for instance in parameter sweep applications such as EOL [21]. Another popular resource abstraction is that of a "cluster" – a homogeneous collection of nodes with tight connectivity, which is needed by traditional parallel applications as the ones in the NPB benchmark suite [25]. Applications also compose these abstractions. For instance, the EMAN [19] application uses a set of heterogeneous clusters (each of them a homogenous collection), which is a "loose" "bag" of "clusters". To capture these typical resource abstractions used by grid applications, vgDL contains three resource aggregates, distinguished based on their homogeneity and internal connectivity.

- `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

```
// Redline specification
Cluster
RES = [ClusterNode ISA SET
    [Node ISA [CPUType="Pentium4", OSType="Linux"]]
    For all x in ClusterNode
        x.latency(Node) < threshold;
    Maximize(ClusterNode.cpu)
    Count(ClusterNode) >= 1;
    Count(ClusterNode) <= 32;]
```

```
// vgDL specification
Cluster = ClusterOf(Node) [1:32][Rank=CPUs] {
    Node = [(Processor==Pentium4) &&(OS==Linux)]
}
```

Figure 4. Complexity of query in Redline and vgDL

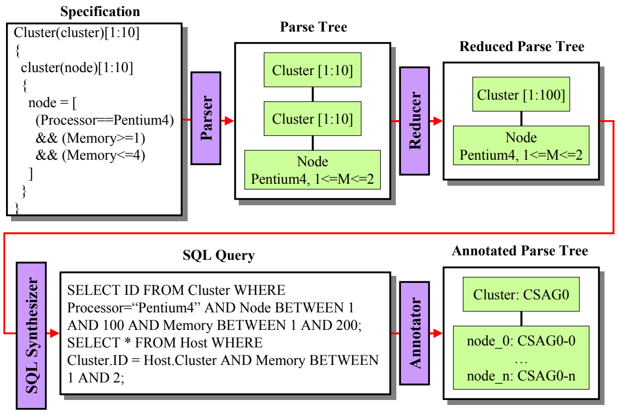
Each aggregate specifies a range for its size (number of elements). Aggregates can be nested as described in Section 3.3. Aggregator properties, and many of the characteristics of vgDL descriptions are *qualitative*. In our analysis of application needs, we found that for many applications, detailed quantitative specifications are a distraction, and as such caused resource specifications to be fragile. We believe our qualitative approach will enable many applications to construct simple, robust, and enduring vgDL specifications.

3.2 Network connectivity

For many grid and distributed applications, inter-node and inter-aggregate network connectivity is critical for performance. vgDL includes 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. `Close` (`highBW`) indicates that latency (bandwidth) between two aggregates is low (high). The properties of the aggregates and operators are intentionally qualitative, and each implementation will use specific default values as definitions for `close` and `far`, as appropriate for distribution of grid resources, and changing as technology advances. For many applications, detailed quantitative specifications are a distraction, and a detriment to simple, robust, and enduring vgDL specifications. Applications that require detailed quantitative measurements of, say, bandwidth, can query the grid information systems once a VG has been instantiated. Such information can be used to apply possibly sophisticated scheduling algorithms and other optimizations at the application level.

3.3 Composition of resources and aggregates

vgDL allows users to define a single resource or an aggregate of resources with certain properties. These properties are embodied in attributes describing the resources, as obtained from grid information services. Aggregate operators can be used to build simple aggregates of individual resources. Description of such aggregates is intuitive and compact. For instance, as shown in Figure 4, a vgDL cluster description is simpler than one in Redline. Furthermore, it is possible to build aggregates of arbitrary depth, e.g., a Bag of Clusters of x86 Nodes. Individual resources or aggregates can be also composed via the simple network connectivity operators described earlier. Users can thus structure resources in top-down fashion and decorate components with additional constraints when desired.



```
Cluster(cluster)[1:10]
{
 cluster(node)[1:10]
  {
   node = [
     (Processor==Pentium4)
     && (Memory>=1)
     && (Memory<=4)
   ]
  }
}
```

```
SELECT ID FROM Cluster WHERE
Processor="Pentium4" AND Node BETWEEN 1
AND 100 AND Memory BETWEEN 1 AND 200;
SELECT * FROM Host WHERE
Cluster.ID = Host.Cluster AND Memory BETWEEN
1 AND 2;
```

Figure 5. Example of resource selection and binding

3.4 Ranking (preference)

In addition to specifying constraints, applications can also express preference using a scalar *ranking function*. The ranking function is a user-defined expression consisting of four basic arithmetic operations, which produces a scalar value that represents quality. The ranking function is used to compare the candidates that meet the vgDL specification constraints to determine which one the application would prefer. If no ranking function is provided, or the ranking function does not distinguish the candidates, the vgFAB has a set of default preferences on the remaining attributes. For instance, for most attributes with scalar values, larger is better (the one exception being network latency). Compared to other systems [6-10], the vgDL ranking functions are more flexible: ranking functions that combine multiple attributes in complex fashion can be easily expressed.

3.5 Range-based specifications and search

In contrast to bilateral matching techniques typified by Matchmaking [6-9], vgDL describes constraints that are naturally mapped to multi-dimensional range search. Applications specify acceptable value ranges for attributes. We exploit this capability to identify sets of candidate solutions – as opposed to a single solution. This helps in shared resource environments where resources will be overloaded or in high demand.

3.6 Other Features

Time and Duration: By default, the execution system allocates resources immediately for unlimited time. For applications or resources with complex temporal constraints, vgDL can describe access to systems with advance reservation.

Resource Quantity: vgDL requests reflect a range of acceptable quantities of resources (i.e. 8 to 64 nodes). For applications, it is sometimes convenient to express total capability of an aggregate. vgDL allows total desired computational capability (service units or SU's) to be tied to a request; enabling coordinated parameter resolution by the vgFAB. Similarly, application can specify total memory or disk capacity using a dot operator together with the aggregate name (e.g. *Clustername*.Memory, *TightBagname*.Disk, etc.).

4. VIRTUAL GRID PROTOTYPE

To demonstrate and evaluate the Virtual Grid Execution System (vgES), we implemented a prototype system. In this system, the vgFAB implements selection based on a local copy of the grid resource information in a relational database. This local copy is

```
ClusterOf<m1, n1> { associator<m2, n2> } or
TightBagOf<m1,n1> {ClusterOf<m2,n2>
  ⇨  ClusterOf<m1+m2, n1+n2>
TightBagOf<m1, n1> { TightBagOf/LooseBagOf<m2, n2> }
  ⇨  TightBagOf<m1+m2, n1+n2>
LooseBagOf<m1, n1> { associator<m2,n2> }
  ⇨  # of associators is between m1 and n1
Component<m1, n1> close Component<m2, n2>
  ⇨  TightBagOf<m1+m2, n1+n2> satisfying both the
     components requirements
ClusterOf/TightBagOf { A far B }
  ⇨  ClusterOf/TightBagOf { A close B }
```

Figure 6. Simplifying Reductions

created and maintained by periodic downloads from grid information services.

Figure 5 depicts the resource selection process and components involved in each step. First, vgFAB builds a parse tree analyzing the given resource description and simplifies this parse tree according to some rules. Based on the simplified parse tree, the SQL synthesizer generates SQL queries for the resources, which capture ranking functions and default preferences. The vgFAB then submits the SQL queries to the resource information database, annotates the vgDL parse tree, and binds the resources (trying new resources if the binding fails). Finally, the vgES returns a handle to a VG instance.

The prototype vgES is built in Java: and uses a MySQL database to store grid resource information. The vgFAB makes queries against this information using the MySQL Connector/J JDBC driver [26].

4.1 Resource Classification

Many existing resource selection techniques exhibit poor scalability because the representation of grid resource information, e.g., in systems such as MDS [15], is resource-oriented, making it hard to retrieve information needed to address users requests efficiently. For example, collecting a set of 50 nodes that have similar configuration across multiple sites might require hundreds or thousands of information services requests. Our approach is proactive, caching much of the available grid resource information locally in a database, and exploiting query optimization technology to answer requests efficiently. In this model, dynamic information can be supplied as periodic updates to the database. Other resource attributes in the database include processor architecture, operating system, as well as network latency and bandwidth. In this approach, large numbers of requests can be supported by a single database (see Section 5), and if needed multiple database systems can be deployed.

Optimal resource selection is typically an NP-hard problem [9], so finding an optimal solution in grids with millions of resources is impractical. Instead, our objective is to find a good solution that has quantifiable quality. To support this, the collected grid resource information is populated into a database according to a *classification* that identifies aggregates used in the vgDL. This supplementary classification information can improve the selection process significantly. Specifically, individual hosts are grouped into clusters and clusters are grouped into tight bags. Singleton hosts either are part of a tight bag or exist as independent entities in the implied loose bag that includes all resources.

```
vgNode={node=[Processor==Pentium4]}
vgCluster=ClusterOf(node)[10:20] {
  node=[(Processor==Pentium4)&&(Memory>=1024)]}
vgTightBag=TightBagOf(node)[10:20] {
  node=[(Processor==Pentium4)&& (Memory>=1024)]}
```

Figure 7. Sample vgDL Specifications

This classification of resources in a real deployment could be achieved by independent (centralized or distributed) agents identify resource classes using the network properties and host configuration information retrieved from external information services systems such as the Network Weather Service (NWS), the (Monitoring and Discovery Service) MDS, or self-classified resources such as a cluster managed by a batch scheduler.

4.2 Simplifying Reductions

Since vgDL can express complex resource specifications that may require exponential search time, our vgFAB prototype exploits aggregate composition semantics to simplify descriptions, enabling their efficient implementation. For instance, a nested cluster expression like *ClusterOf<cluster> [1:10] { cluster = ClusterOf<node> [1:10] { node = ...} }* can be regarded as *ClusterOf<node> [1:100] { node = ... }*. Instead of searching multiple tightly coupled clusters, vgFAB searches a larger cluster that can accommodate multiple clusters, and then partitions the cluster into multiple logical sub-clusters based on the requirements. This simplifying transformation preserves the properties of the vgDL request, but eliminates the expensive pair-wise connectivity checking between clusters.

Figure 6 summarizes the simplifying reductions adopted in our current implementation. A common rule is that if a parent (in parse tree) has stronger constraints than the children, then we can safely simplify the request by requiring the children to follow the parent's constraints. With respect to connectivity, cluster and tight bag are stronger than loose bag while cluster is stronger than tight bag and loose bag with respect to homogeneity. Meanwhile, `close` and `highBW` operators are stronger than `far` and `lowBW`, respectively. If associators with strong connectivity nest weak operators, the weak operators are translated as corresponding strong ones. On the contrary, if strong operators nest associators with weak connectivity, the weak associators are translated into the strong ones. Our simplifying reductions allow us to achieve fast, good solutions for vgDL requests, but do exclude some possible solutions in the grid resource space. Quantifying the impact of these exclusions is an interesting subject for future research.

4.3 Query Synthesis

With a structured resource database and a simplified resource request, the vgFAB next synthesizes a sequence of SQL queries to identify a set of candidates. For instance, to select a tight bag, vgFAB first retrieves information about tight bags from the database and identifies the clusters composing the tight bags. Then, vgFAB searches the hosts composing each cluster. At each step, vgFAB filters out infeasible candidates using a `WHERE`

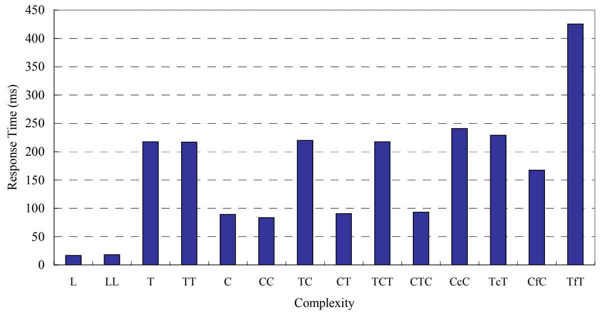


Figure 8. Average response time of queries varying request complexity and size with 1,000,000 hosts (L: LooseBag, T: TightBag, C: Cluster, c: Close, f: Far)

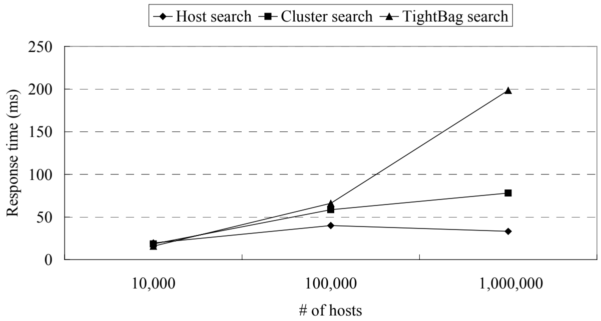


Figure 9. Average response time with different size of resource population

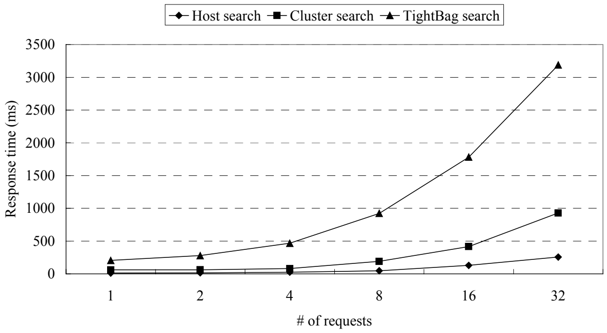


Figure 10. Average response time of concurrent requests with 1,000,000 hosts

clause with the constraints of individual hosts and aggregators. Then, vgFAB 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. vgFAB bounds the number of candidates per request using a `LIMIT` clause. In summary, a general form of synthesized SQL query is:

```
SELECT attributes FROM tables, ranking
function AS rank WHERE constraints ORDER BY
rank DESC LIMIT # of candidates
```

The result of this query is a set of candidates all of which meet the requirements of the simplified vgDL request.

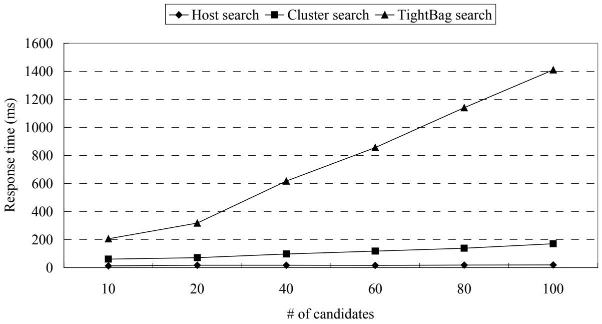


Figure 11. Average response time of a single query varying the number of candidates per request with 1,000,000 hosts

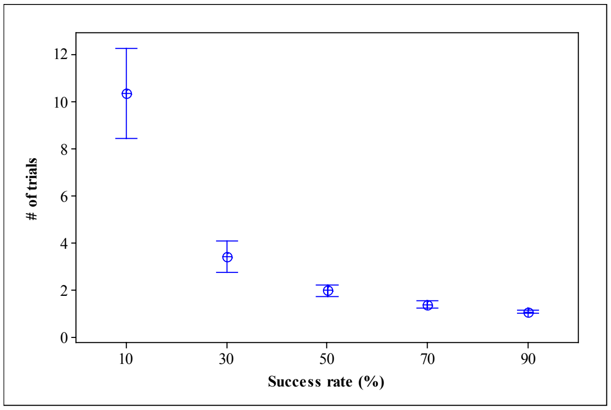


Figure 12. Interval plot of the number of binding trials before success changing the success probability of binding

4.4 Binding and VG Construction

Armed with a set of ranked, satisfactory candidates, the vgFAB can proceed to bind resources (selecting new candidates if binding fails due to resource unavailability or some other reason). With a bound set of resources, the vgFAB constructs an annotated vgDL request, the realized VG representation, decorating it with the physical resource information. This living representation of the virtual grid can then be used by the application for scheduling and execution. If a candidate component corresponds to a simplified vgDL expression, the vgFAB partitions the resource and logically organizes them according to the original vgDL specification. The application can then access the VG via the vgES API, to retrieve logical groups of resources as well as their attributes.

5. EMPIRICAL EVALUATION

We evaluate the efficiency and scalability of our prototype for vgFAB. In addition, we explore how the vgFAB can support the selection of multiple candidates for a single vgDL request and how effectively these candidates can be used to increase success rates for application requests. For experiments, we used variations of the vgDL specifications in Figure 7.

The grid resource environments used are based on state of the art research tools for generating representative structures for grid resources (Kee and Chien’s statistical grid resource generator [27] and Medina, et. al.’s BRITE network topology generator [28]). The vgFAB runs are done on a modest computing system -- IBM Thinkpad X31 with a single 1.4 Ghz Pentium3 processor and 512 MB memory. The vgFAB prototype including the database system is configured on the Thinkpad. Completion time for a request is measured from the time when it is initiated to when a response is received.

5.1 Scalability

To evaluate the scalability of our approach, we vary the vgDL request complexity, the size of the resource population, the number of candidates, and the number of requests.

5.1.1 Request complexity

We first evaluate how the complexity of a vgDL request affects the cost of selection. To increase complexity, we change the depth of nesting, order of aggregates, and operators for connectivity. We used a database with 1,000,000 hosts, finding 10 candidates per request, and repeated 100 times. As shown in Figure 8, nesting the aggregators does not affect the cost of selection. LL (LooseBag of LooseBag), CC (Cluster of Cluster), and TT (TightBag of TightBag) have the same cost as L, C, and T, respectively. This is expected because multiple nested aggregates of same type are reduced to a larger aggregate by the simplifying rules. In contrast, ordering of nested aggregates affects the selection cost. If the top-most aggregate is a tight bag, then the simplified description is a tight bag, but if the top-most aggregate is a cluster, then the simplified description is a cluster. This matches our empirical results quite well. According to the simplifying rules, aggregates with a `close` operator are translated as a larger tight bag. Meanwhile, aggregates with a `far` operator are translated into two independent requests. The results in Figure 8 match this well with the response time of "Cluster close Cluster" close to that of "TightBag" and the response time of "Cluster far Cluster" close to the sum of the response time of "Cluster". In all cases, the cost of identifying ten candidates is modest, much less than one second.

5.1.2 Grid Size

Figure 9 shows how resource population affects response time. For each type, we changed the database population from 10,000 hosts to 1,000,000 hosts. We identified 10 candidates for each vgDL request and repeated the procedure 100 times. Our results show that the cost of selection increases with the size of resource population, but the amount of growth varies by request type. For example, a tight bag search issues large number of queries.

To find 10 candidates for a loose bag for a single vgDL request, 12 queries are required. In vgDL requests to find 10 candidates for a single host, one query is needed, and 11 queries needed to find 10 candidates for a cluster. The number of queries for tight bag is dependent on the number of clusters composing the tight bag.

5.1.3 Number of requests

Finally, we evaluate scalability with respect to workloads with a fixed server configuration. The execution system has a database with 1,000,000 hosts and finds 10 candidates per request. Figure 10 shows the average response time when varying the number of concurrent requests from 1 to 32. For cluster and host search, our vgFAB prototype serves more than 32 requests per second and sustains 8 requests per second for tight bag search. In summary, our results show that resource selection for virtual grids of 10,000's of resources can scale up to grids with millions of resources, identifying matches in less than one second.

5.2 Virtual Grid Quality

In response to each vgDL request, vgFAB returns a VG. In general, the vgFAB identifies multiple candidates for a request in order to improve the chance of successfully binding resources in the presence of resource contention. We present a preliminary evaluation of the trade-off between search result quality, overhead of identifying multiple candidates, and probability of successfully binding a resource.

Figure 11 shows the average response time when different numbers of candidates are considered by vgFAB. We used a database with 1,000,000 hosts and repeated experiments 100 times. The cost of identifying multiple candidates appears to increase linearly with the number of candidates. However, as all of these costs are very low, we have too little information to draw strong conclusions. Note that these candidates are the "best" ones, according to user-specified ranking functions or default preferences. So, if vgFAB examines 10 candidates, the application is guaranteed to obtain a resource set among the 10 best ones.

If binding were always successful, that is, with no contention for resources and no resource failure, only one candidate would be needed. We simulated resource contention and failures by assigning a (uniform) probability of successful binding to all resources. Figure 12 shows 95% confidence intervals plot of the number of binding attempts needed until success versus the probability of successful binding individual resources. The results show that even with low probabilities of binding individual resources (e.g., 10%), vgFAB can successfully bind a VG to meet the vgDL request after a moderate number of tries (<15). The results in Figure 11 show that the cost of considering such a number of candidates is very low (<300ms). Our overall conclusion is that vgFAB can bind good VGs with low overhead even in the presence of high levels of resource contention and/or failures.

6. RELATED WORK

Our work on Virtual Grids builds on and is informed by the four-year GrADS project [19] to build development tools for adaptive grid applications. In the GrADS framework, tight coupling of an individual application basis through performance model and contract monitor was too intensive and complex: given the current complexity of grid resource environments and their likely future growth in scale, embedding understanding of the resource environment into each application is impractical. The Virtual Grid approach takes a step back from these, striking a middle ground between the GrADS approach and pure low level resource oriented schemes, and uses a layered approach (i.e., functional decomposition of discovery, selection, allocation, etc.) that enables simple interaction between the application and the resource environment, thereby yielding a much more flexible architecture. While the VG concept includes a wide range of research fields of Grid middleware, the scope this paper is limited to resource description, selection, and binding.

Resource description and selection is often formulated as a resource matchmaking problem. For instance, Condor ClassAds were proposed for symmetric bilateral matching of a single customer with a single resource [6], and extended to co-allocation (gangmatching) [8] and aggregate characteristics (set matching) [7]. Meanwhile, Redline [9] reinterpreted matching as a constraint satisfaction problem and exploited constraint-solving techniques to implement matching process. Many other techniques are also used for resource selection. Other systems perform resource description and selection (e.g., SWORD [10], SpiderNet [11]) and rely on P2P support for operating in a decentralized manner. Additional related work includes [12,13].

A fundamental difference between our work and that above is our description language based on application-level resource aggregate abstractions. Generally, previous approaches are either not expressive enough to describe such aggregates, or are cumbersome to do so. Further, often these complex descriptions do not provide a simple structure for application scheduling and efficient resource selection. Another important difference between our work and others is our use of ranking functions. For instance, Redline allows users to maximize/minimize a single constraint, and SWORD allows users to define a penalty function for individual resource and sums up the values using internally defined normalized utility function. By contrast, the vgDL ranking functions are more flexible and allow user to express combined preferences over multiple attributes easily.

Finally, although not the main focus of this paper, a key advantage of the VG approach is its support of adaptation: a VG instance is a living entity that is monitored at runtime and for which the application has a handle. A subset of the VG resources can be removed or replaced at runtime by the vgES, and new resources can be added, followed by application rescheduling [18]. No other systems can do this adaptation conveniently without creating an abstraction similar to our virtual grid.

7. SUMMARY & FUTURE WORK

We have introduced the Virtual Grid concept and we have presented a novel structural resource description language that captures relevant resource abstractions for Grid applications, and an efficient resource selection mechanism. Our prototype implementation demonstrates scalability: it is feasible to find and select resources in real time even in enormous resource environments. With a heavy workload of 1,000,000 hosts and 32 concurrent requests per second, the system supports the request load and achieves good performance. Moreover, when the candidates are evaluated using user-provided ranking function, in most cases, the vgES system efficiently finds sets of candidates and can use them to tolerate a high contention environment. We plan to extend these experiments, for instance investigating how the frequency of updates to the resource information database affects performance.

The current implementation of the vgES system uses the Globus infrastructure (Globus Toolkit v2.4) to collect available resource information, provide security, execute application on remote resources, and to transfer application files. Since our vgES implementation is written in Java, we expect to migrate to Globus Toolkit 4.0 and ultimately present vgES services as grid services.

Although the current prototype system is localized, the Virtual Grid concept does not put any restriction on system architecture, and distributed implementations are eminently feasible. For instance, to distribute the workloads due to updates of resource information, the resource information database can be replicated or distributed. 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. We will present a distributed implementation of Virtual Grid execution system in an upcoming paper. In the meantime, we are designing and implementing the other components of the Virtual Grid execution system, including a resource discovery agent, resource performance monitor, and launcher. We expect that the resulting system will provide users with a simple but powerful way to deploy their applications efficiently on large-scale Grid platforms.

8. ACKNOWLEDGEMENTS

The authors and research described here are supported in part by the National Science Foundation under awards NSF EIA-99-75020 Grads and 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.

9. REFERENCES

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

[2] 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.

[3] 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.

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

[5] L. Peterson, D. Culler, T. Anderson, and T. Roscoe. A blueprint for introducing disruptive technology into the internet, HotNets-I, Oct. 2002.

[6] R. Raman, M. Livny, and M. Solomon, Matchmaking: Distributed Resource Management for High Throughput Computing, IEEE HPDC'98, pp. 140-147, July 1998.

[7] 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.

[8] R. Raman, M. Livny, M. Solomon, Policy Driven Heterogeneous Resource Co-Allocation with Gangmatching, IEEE HPDC'03, pp. 80-89, June 2003.

[9] C. Liu and I. Foster, A Constraint Language Approach to Matchmaking. RIDE'04, Boston, 2004.

[10] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. UC Berkeley Technical Report UCB//CSD-04-1334, July 2004.

[11] X. Gu, K. Nahrstedt, and B. Yu, Spidernet: An Integrated Peer-to-peer Service Composition Framework, IEEE HPDC'04

[12] T. Kichkaylo, A. Ivan, and V. Karamcheti, Constrained Component Deployment in Wide-area Networks Using AI Planning Techniques, IEEE IPDPS'03, Apr 2003.

[13] A. Huang, and P. Steenkiste, Building Self-configuring Services Using Service-specific Knowledge, IEEE HPDC'04

[14] I. Foster, C. Kesselman, and S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations. International Journal of High Performance Computing Applications, 15(3):200-222, 2001.

[15] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Grid Information Services for Distributed Resource Sharing, IEEE HPDC'01, Aug. 2001.

[16] 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.

[17] F. Sacerdoti, M. Katz, M. Massie, D. Culler, Wide Area Cluster Monitoring with Ganglia, IEEE Cluster, Dec 2003.

[18] K. Cooper et. al, New Grid Scheduling and Rescheduling Methods in the GrADS Project, Workshop for Next Generation Software, April 2004.

[19] 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

[20] Available from http://ncmi.bcm.tmc.edu/~stevel/EMAN/doc

[21] 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.

[22] Available from http://www.ncsa.uiuc.edu/expeditions/MEAD

[23] W. Chrabakh and R. Wolski, GridSAT: A Chaff-based Distributed SAT Solver for the Grid, ACM/IEEE SC'03, Nov. 2003.

[24] 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

[25] D. Bailey, T. Harris, W. Saphir, R. v. d. Wijngaart, A. Woo, and M. Yarrow, The NAS Parallel Benchmarks, Technical Report, NAS-95-020, 1995.

[26] Available from http://dev.mysql.com/doc/connector/j

[27] Y.-S. Kee, H. Casanova, and A. A. Chien, Realistic Modeling and Synthesis of Resources for Computational Grids, ACM/IEEE SC'04, Nov. 2004

[28] A. Medina, A. Lakhina, I. Matta, and J. Byers. BRITE: An Approach to Universal Topology Generation. International Workshop on Modeling, MASCOTS'01, Aug. 2001.