

Architecture for Resource Allocation Services supporting Interactive Remote Desktop Sessions in Utility Grids

Vanish Talwar
HP Labs
vanish.talwar@hp.com

Bikash Agarwalla
Georgia Tech
bikash@cc.gatech.edu

Sujoy Basu
HP Labs
sujoy.basu@hp.com

Raj Kumar
HP Labs
raj.kumar@hp.com

Klara Nahrstedt
UIUC
klara@cs.uiuc.edu

ABSTRACT

Emerging large scale utility computing systems like Grids promise computing and storage to be provided to end users as a utility. System management services deployed in the middleware are a key to enabling this vision. Utility Grids provide a challenge in terms of scale, dynamism, and heterogeneity of resources and workloads. In this paper, we present a model based architecture for resource allocation services for Utility Grids. The proposed service is built in the context of interactive remote desktop session workloads and takes application performance QoS models into consideration. The key design guidelines are hierarchical request structure, application performance models, remote desktop session performance models, site admission control, multi-variable resource assignment system, and runtime session admission control. We have also built a simulation toolkit that can handle mixed batch and remote desktop session requests, and have implemented our proposed resource allocation service into the toolkit. We present some results from experiments done using the toolkit. Our proposed architecture for resource allocation services addresses the needs of emerging utility computing systems and captures the key concepts and guidelines for building such services in these environments.

Categories and Subject Descriptors

C.2.4 [Distributed Systems]: Network Operating Systems, Client/server;
D.4.1 [Operating Systems]: Process Management—*Scheduling*;
D.4.7 [Operating Systems]: Organization and Design—*Interactive systems*

General Terms

Design, Management, Performance

Keywords

Resource Allocation Service, Grid Computing, Remote Desktop Sessions, QoS.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2nd Workshop on Middleware for Grid Computing Toronto, Canada
Copyright 2004 ACM 1-58113-950-0 ...\$5.00.

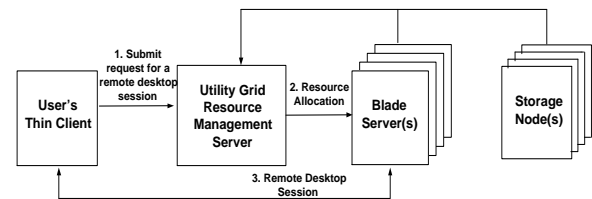


Figure 1: High level conceptual view of the system supporting remote desktop sessions

1. INTRODUCTION

Today's enterprise IT systems are being consolidated into centralized data centers for reducing cost and to improve manageability. Efforts are now being made to increase the degree of sharing of these consolidated computing and storage systems and to provide them to the end-user as a utility. Such systems are being coined as Utility Computing Systems or *Utility Grids*. In such systems, geographically distributed data center sites host the shared IT infrastructure - blade servers and storage servers, which are allocated *dynamically* and *on-demand* to the applications of the end-user. These applications could be enterprise three-tier applications, batch applications, and interactive applications. In this paper, we are particularly interested in interactive applications hosted on shared blade servers in the data center. These applications are then viewed by the end-user through remote desktop sessions provided through technologies like Citrix [1], Microsoft Terminal Servers [2], VNC [3]. The examples of applications viewed through such sessions belong in the vertical segments of financial services, CAD/CAM applications, and office applications like MS Word, MS Outlook, MS Excel etc. Figure 1 shows the conceptual view of such a system.

One of the fundamental system management services needed in the middleware to enable the vision of Utility Grids is a *Resource Allocation* service. This service is responsible for the *dynamic allocation* of a fraction of a blade servers' resources in response to an end-user request. Today's IT systems typically pre-install, pre-allocate, and reserve the servers and storage resources for end-customers' applications, leading to over-provisioning and higher costs. On the other hand, a utility computing system envisions servers and storage resources to be sharable across end-customers' applications and be allocated dynamically as the need arises. This brings out the need for a resource allocation service that has to consider the real-time system utilization of the blade servers, and the dynamic requirements of requests while making an allocation decision. The resource allocations made must further meet the min-

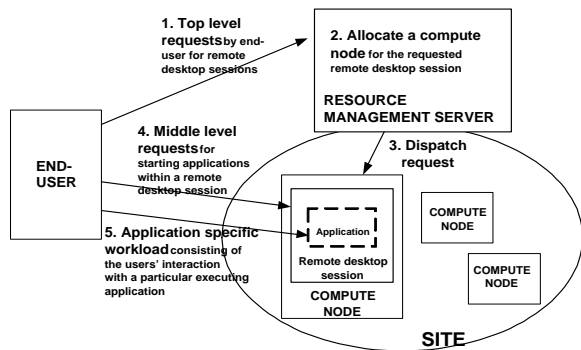


Figure 2: Flow diagram in the proposed system

imum performance requirements of the hosted applications, while maintaining a high system utilization by avoiding over provisioning of resources to the applications. Prior work has looked at building resource allocation services for supporting batch applications [4], and three-tier enterprise applications [5]. We would like to address the needs of interactive remote desktop sessions which are typically more sensitive to performance needs. We make the following contributions through this paper:

- Architectural design guidelines and detailed description of a resource allocation service for supporting interactive remote desktop sessions in Utility Grids. In summary, the key guidelines are a hierarchical request structure, application performance models, remote desktop session performance models, site admission control, multi-variable resource assignment system, and runtime session admission control.
- Simulation toolkit that implements the resource allocation service and some experimental results for mixed workloads obtained using the simulation toolkit.

Our proposed architecture services requests for remote desktop sessions from end-users *dynamically* and allocates *on-demand* a fraction of a blade server in the data center site for the end-users' request. The fraction of the resources to be chosen is determined through the *dynamic* generation of the performance model for the requested remote desktop session using pre-generated *application performance models* for the applications that would execute in the context of the requested remote desktop session. The allocation is thus QoS driven and admission control systems are used to enforce QoS. Further, the low level details of the sharing of IT infrastructure is hidden from the end-user and the end-user is provided with a *virtualized* environment with QoS guarantees.

The rest of the paper is organized as follows. In Section 2, we present the proposed architecture. Section 3 describes the simulation toolkit and experiments. We conclude in Section 4.

2. ARCHITECTURE

The system model we consider in this paper is a single data center site. A data center site consists of blade servers (henceforth also referred to as compute nodes), storage servers, and a resource management server. Our proposed resource allocation service components are resident on the resource management server and the blade server as would be explained in Section 2.1 and Section 2.2. Figure 2 shows the flow diagram of the sequence of steps executed in the system. The end-users submit requests for remote desktop sessions to the *Resource Management Server*. The resource management server then allocates a fraction of a blade server's resources for the

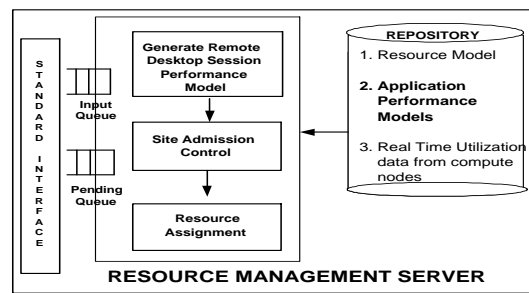


Figure 3: Resource Management Server

user's request for the remote desktop session. A request to start the remote desktop session is then dispatched to the allocated blade server. Once the session is started, the user interactively starts applications through the established remote desktop session connection. This is shown as *middle level requests* in Figure 2. These middle level requests go through a *Session Admission Control System* at the blade server. Once the applications are started, the user interacts with those applications through an *application specific workload*. We thus have a hierarchical request structure in the system, top level requests, middle level requests, and application specific workload, as illustrated in Figure 2.

2.1 Resource Management Server

Figure 3 shows the architecture for the resource management server. It hosts a repository consisting of application performance models, resource models, and the real time utilization data of blade servers. There are two queues, an Input Queue holds the users' requests when they first enter the system; and a Pending Queue holds requests that could not be assigned a blade server that meets the requests' performance requirements. The requests in the Pending Queue wait till the release of resources by the blade servers sufficient enough to meet the requests' performance requirements. The resource models capture the static characteristics of the blade servers eg. the server make, the maximum CPU and memory capacity etc. On selecting a request from the Input Queue, the set of blade servers satisfying the users' preference of static characteristics are obtained through a match of the users' preferences with those in the resource models. Subsequently, a '*remote desktop session*' *performance model* for the requested remote desktop session is dynamically generated based on the list of applications desired in that session. This step uses the application performance models from the repository. The *Site Admission Control System* and the *Resource Assignment System* then make their decisions using this generated model. We describe these in subsequent subsections.

2.1.1 Application Performance Model

The application performance model describes the resource requirements of the application to be able to perform at an acceptable QoS level. Such a model is key to our proposed resource allocation service. Such models would be built offline by the system administrators and populated into the repository. We are interested in building such models for interactive applications hosted on blade servers and viewed in a thin client setting using remote desktop sessions as shown in Figure 1. Below, we describe briefly how to build such models. Subsequently, we give a formal representation for the model.

Application profiling is the basis for building application performance models. Application profiling is done by executing the application in isolation on a standard platform, and then applying a

characteristic workload to the application. The resource consumption of the application is continuously monitored over the entire period of execution. Statistical techniques are then applied to the trace data to determine a desired resource requirement value that should be allocated to the application for acceptable performance. There has been prior work in this area, most of which has been studied in the context of batch and e-commerce applications [6]. The profiling of interactive applications in a thin client setting present additional challenges: Firstly, the execution of an interactive application is primarily influenced by end-user behavior. This user behavior needs to be modeled for the application being profiled and subsequently a synthetic workload needs to be generated conforming to the modeled user behavior. The work being done in this area eg. [7, 8, 9] typically propose the use of states to capture user interactions, and the use of Markov chains to model probabilistic transitions. Additional problems presented by thin client systems is (i) the need to also measure user perceived performance on the client for accurate latency measurements, and (ii) the need to consider the resource consumption of the remote display server in addition to that of the application. We do not go much further into the details of the methodology of building the application performance models and keep our focus in this paper to architectural principles. The reader is referred to related works cited above and also in works of [10, 11, 12, 13] which describe the measurement of the resource consumption of interactive applications in a thin client setting.

Throughout the paper, we represent the application performance model of an application A_i using $A_i = \{C_i, N_i, S_i, L_{N_i}, L_{S_i}\}$, where C_i, N_i, S_i represent the desired CPU utilization (in cycles/second), desired network bandwidth, and desired storage bandwidth respectively for the application. L_{N_i} represents the acceptable network latency between the end-users' thin client and the blade server, L_{S_i} represents the acceptable storage latency between the blade server and remote storage node.

2.1.2 Remote Desktop Session Performance Model

A remote desktop session performance model describes the resource requirement for a remote desktop session. Such a model is generated dynamically for every user request. A remote desktop session, for the purposes of modeling, is viewed as consisting of a remote display server and one or more application processes. For example, a VNC remote desktop session [3] would consist of a VNC remote display server and all the applications running in the context of this VNC desktop session. These applications are started interactively by the end-user and they execute in the context of the remote desktop session on the blade server. All of these applications share the resources allocated to the remote desktop session in which they execute. We next describe a framework for building a remote desktop session performance model.

A remote desktop session performance model is built using the application performance models of the applications which would execute in its context. This list of applications are obtained dynamically through the users' request or it would be inferred based on the users' profile [13]. On obtaining this list of applications, we read in the individual application performance models for these applications from the repository. At the time of generation of the remote desktop session model, we do not however know the execution order of these applications. The users' request and/or users' profile only gives us the list of applications desired during the session. The user could interactively start these applications in various possible execution orders at runtime. The end-user may further decide at run-time to start several instances of each application. Thus, the execution order of applications, and number of instances for each

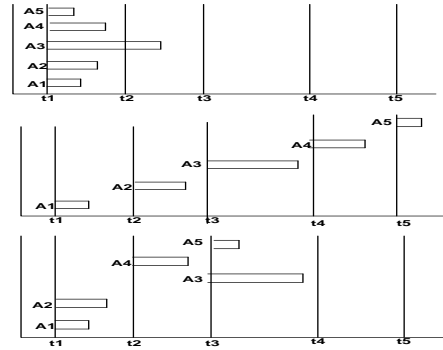


Figure 4: Modeling the resource requirement for a remote desktop session at the Resource Management Server. The top, middle, and bottom graphs show the Simultaneous, Sequential and Mixed execution order of five applications.

application is a run-time decision not known at the time of generation of the remote desktop session performance model at the Resource Management Server. However, the remote desktop session performance model depends on such execution orders. One solution to address this problem would be for the user to specify the execution orders and instances for his desired applications in her request. However, this may not be a very good solution since the user may find it difficult to determine such execution orders at the time of submission of her request. Moreover, since we enable interactivity, the user would like to choose the order and number of instances at runtime. We propose some models for determining the execution orders of the applications. One of these models of execution orders is then selected for a users' request by a policy decision, and then the system generates the corresponding remote desktop session performance model using that execution order. We describe below and illustrate in Figure 4 some of these execution order models and their corresponding remote desktop session performance models. We consider n applications, $A_i, i = 1$ to n , in a users' remote desktop session and the remote desktop session performance model to be represented as

$$RemoteDesktop_i = \{C_{desktop}, N_{desktop}, S_{desktop}, L_{N_{desktop}}, L_{S_{desktop}}\}.$$

The notations are explained in Figure 5.

(a) *Simultaneous execution of an instance of each desired application.* In this case, the aggregate resource requirements for the remote desktop session is modeled as the sum of the individual requirements

$$\begin{aligned} C_{desktop} &= O_C + \sum_{i=1}^{i=n} C_i, \\ N_{desktop} &= O_N + \sum_{i=1}^{i=n} N_i, \\ S_{desktop} &= O_S + \sum_{i=1}^{i=n} S_i, \end{aligned}$$

where O_C, O_N, O_S are the extra overheads that is accounted for due to other processes eg. monitoring software etc., that may run within the remote desktop session at runtime. The latency requirements for the remote desktop session is taken as the minimum of those for the individual application sessions.

$$L_{N_{desktop}} = \min_{i=1}^{i=n} L_{N_i}, L_{S_{desktop}} = \min_{i=1}^{i=n} L_{S_i}.$$

(b) *Sequential execution of the applications.* In this case, the aggregate resource requirements for the remote desktop session is modeled as the maximum of the individual requirements

$$\begin{aligned} C_{desktop} &= O_C + \max_{i=1}^{i=n} C_i, \\ N_{desktop} &= O_N + \max_{i=1}^{i=n} N_i, \\ S_{desktop} &= O_S + \max_{i=1}^{i=n} S_i, \end{aligned}$$

The latency requirements for the remote desktop session is taken as the minimum of those for the individual application sessions.

Symbol	Meaning
UC	Current CPU percentage utilization
UN	Current network bandwidth percentage utilization
US	Current storage bandwidth percentage utilization
Cdesktop	Aggregate CPU utilization in cycles/second for the remote desktop session
Ndesktop	Aggregate network bandwidth for the remote desktop session
Sdesktop	Aggregate storage bandwidth for the remote desktop session
LN desktop	Acceptable network latency for the remote desktop session
LS desktop	Acceptable storage latency for the remote desktop session
P	CPU Processor speed
NT	Total network bandwidth for the compute node
ST	Total storage bandwidth for the compute node
NE	Dynamic End-to-end network bandwidth between the compute node and the users' submission node hosting the display (for remote display traffic)
SE	Dynamic End-to-end storage bandwidth between the compute node and the file server hosting the user's data (for remote storage traffic)
NLE	End-to-end network latency between the compute node and the users' submission node hosting the display
SLE	End-to-end storage latency between the compute node and the file server hosting the user's data
TC	Maximum Threshold percentage set for the CPU Utilization on the compute node eg. 80%
TN	Max Threshold percentage set for the Network Utilization on the compute node
TS	Max Threshold percentage set for the Storage Utilization on the compute node

Figure 5: Notations

$$L_{N_{desktop}} = \min_{i=1}^{i=n} L_{N_i}, L_{S_{desktop}} = \min_{i=1}^{i=n} L_{S_i}$$

(c) *Mixed Case* when some applications are executed simultaneously, and some others are executed sequentially. In this case, the resource requirement is either modeled as a value based on history based prediction between the two extremes of simultaneous execution and sequential execution, or it is negotiated with the end-user.

(d) *Unknown Profiles of the applications*. This would be the case when the applications are being executed for the first time, and the system is completely unaware of the resource requirements. In this case, the resource requirements for the remote desktop session could be modeled assuming worst case requirements (like requiring the maximum permissible resources on a node), or the user could specify the requirements.

2.1.3 Site Admission Control System

The Site Admission Control system uses the remote desktop session models for admission control decisions. This system is responsible for determining if a blade server with its current resource utilization values can meet the resource and latency requirements for a given remote desktop session. The resource assignment heuristics are then applied to only those resources that satisfy the admission control test. The Site Admission Control system takes as input: *Remote Desktop Session performance model* for the requested list of applications, *blade servers* that satisfy the coarse grain static resource requirements for the user's request, and the *real time resource utilization* values of the nodes. Below is the admission criterion. Please refer to Figure 5 for the notations.

$$P * (T_C - U_C) \geq 100 * C_{desktop},$$

$$\min((T_N - U_N) * N_T, 100 * N_E) \geq 100 * N_{desktop},$$

$$\min((T_S - U_S) * S_T, 100 * S_E) \geq 100 * S_{desktop},$$

$$N_{LE} \leq L_{N_{desktop}}, S_{LE} \leq L_{S_{desktop}}.$$

The expressions on the left side of the comparison operator represent the currently available resources on the compute node (blade server) and those on the right side of the comparison operator represent the resource requirement for the remote desktop session. The admission check is thus to compare that the currently available resources on the compute node can satisfy the required values for the requested remote desktop session. Note that due to the heterogeneity in the hardware platforms eg. CPU, we have to normalize the values of the quantities before comparison eg. CPU utilization is expressed in cycles/second.

2.1.4 Resource Assignment System

The Resource Assignment system is responsible for assigning

one of the blade servers which satisfies the site admission check, for the users' request. It takes into consideration the remote desktop session performance model, and aims to minimize the wait time for requests. The wait time in this section refers to the time it takes for the blade server to be assigned to a user since receiving the request. Unlike batch job submissions, a user after submitting the request for remote desktop session typically waits for the blade server to be allocated to him immediately. In our system, the wait time is dependent on (is the summation of) the wait time in the *Input Queue*¹, the wait time in the *Pending Queue*² waiting for resources to become available, and processing overhead of the admission control and assignment algorithms. We allow for priorities to be assigned to requests based on the profile of the user. The requests would be picked from the Input Queue based on priority, thus reducing the wait time for higher priority requests in the Input Queue.

Multi Variable Best Fit Algorithm

Figure 6 presents the pseudo code for a multiple variable best fit algorithm that takes resource requirement heuristics into consideration for resource assignment. Note that at this point only those blade servers are being considered which satisfy the Site Admission Control test. Also, for the use case scenarios being considered by us, each request requires only a single blade server. However, as mentioned earlier, we allow resource sharing i.e there could be multiple remote desktop sessions allocated on the same blade server simultaneously. A Best Fit algorithm for assigning blade servers to remote desktop sessions would always try to pack up bins tightly thus reducing the possible fragmentation. This would enable us to assign more sessions onto the blade servers and should help in reducing the wait time for the requests in the Pending Queue. We therefore consider a Best Fit algorithm for resource assignment. However, we have to consider multiple variables in the algorithm - CPU, network bandwidth, and storage bandwidth. For a particular remote desktop session, one or more of these resources may be a bottleneck resource. We introduce weight functions corresponding to each of these fine grain resources and adjust the weight assignment accordingly for the bottleneck resource variables. For example, for CAD design sessions, the CPU would be the bottleneck resource variable and we should give more weightage to CPU utilization values for such sessions. Similarly for financial transaction applications, the storage bandwidth would be the bottleneck resource variable, and for office applications, the network latency would be the bottleneck resource variable. Further, the algorithm determines the difference between the available and required resource utilizations, and assigns the weight functions as inversely proportional to these delta values. Thus, it does weighted best fitting along multiple dimensions. The weights are assigned for the different parameters/variables as functions, and we pick the compute node that has the highest aggregate weight across dimensions. The resource and latency requirements used for the remote desktop sessions in the algorithm are those obtained from the remote desktop session performance model described in Section 2.1.2.

2.2 Runtime System

Figure 7 shows the components of the runtime system. The runtime system components are resident on the blade server and are responsible for resource allocation at runtime. Unlike traditional batch applications, end-users can interactively start applica-

¹Input Queue is the queue into which the requests are placed as they arrive into the Utility System.

²Pending Queue is the wait queue into which requests go if all the eligible blade servers for a request do not have enough available resources to satisfy the Site Admission Control performance criterion test.

1. For each compute node which satisfies the Site Admission Control test
 - a. Determine the free CPU cycles, network bandwidth, and storage bandwidth available on this compute node for a users' request.
 - b. Determine the delta values between the available resources from step a., and the desired resources for the requested remote desktop session. These delta values are denoted as $C_{\text{delta}}, N_{\text{delta}}, S_{\text{delta}}, NL_{\text{delta}}, SL_{\text{delta}}$
 - c. We now assign the following weights:

$$W_C = f(C_{\text{delta}} \text{ Compute Intensiveness})$$

$$W_N = f(N_{\text{delta}} \text{ Average expected display data size})$$

$$W_S = f(S_{\text{delta}} \text{ Data intensiveness})$$

$$W_{NL} = f(NL_{\text{delta}} \text{ Interactiveness})$$

$$W_{SL} = f(SL_{\text{delta}} \text{ Data intensiveness})$$
 The weights ($W_C, W_N, W_S, W_{NL}, W_{SL}$) are inversely proportional to the first parameter ($C_{\text{delta}}, N_{\text{delta}}, S_{\text{delta}}, NL_{\text{delta}}, SL_{\text{delta}}$) and directly proportional to the second parameter (Compute intensiveness, Average expected display data size, Data intensiveness, Interactiveness, Data intensiveness) respectively.
 - d. The effective weight of this compute node for the currently considered assignment is $W_{\text{effective}} = W_C + W_N + W_S + W_{NL} + W_{SL}$
2. Pick the compute node with the maximum assigned weight $W_{\text{effective}}$ for this request. In case of equally ranked compute nodes, we pick the one with the least load where load is defined in terms of CPU utilization

Figure 6: Pseudo code for multi variable best fit algorithm for resource assignment

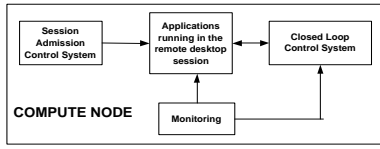


Figure 7: Runtime system

tions throughout the lifecycle of the remote desktop session. This requires the resource allocation service to also have runtime components.

A Session Admission Control system exists at the blade server for every executing remote desktop session. Once the remote desktop session is started, this system receives the middle level requests from the end-user for starting new applications. It is then responsible for determining if the resources allocated to a remote desktop session can allow the starting of the new application while meeting the resource and latency requirements of the new application and without violating the resource requirements of existing running applications in the remote desktop session. The Site Admission Control system makes an admission decision for the remote desktop session assuming the resource requirements specified in the remote desktop session performance model described in Section 2.1.1. However, once the remote desktop session is started on the blade server, the end-user can interactively start the applications in an execution order different from that considered while building the remote desktop session performance model. She may also start several instances of the applications. Hence, we need to perform a Session Admission Control check at the blade server to check dynamically if there are enough resources available for the application without violating the resource availability for currently running applications. If the execution order and application instances during runtime are always as derived using the model in Section 2.1.1, then the session admission control test would always succeed. However, this may not happen in reality especially in the Mixed case in the model, and hence the Session Admission Control system is needed to enforce admission control during runtime.

Let $A = \{A_1, A_2, \dots, A_k\}$ be the current set of applications running in a remote desktop session. Let $A_{(k+1)}$ be the application session for which we are making an admission control decision. Then the Session Admission Control decisions are:

$$C_{\text{desktop}} \geq O_C + \sum_{i=1}^{i=k+1} C_i,$$

$$N_{\text{desktop}} \geq O_N + \sum_{i=1}^{i=k+1} N_i,$$

$$S_{\text{desktop}} \geq O_S + \sum_{i=1}^{i=k+1} S_i,$$

$$L_{N_{\text{desktop}}} \leq L_{N_{(k+1)}}, L_{S_{\text{desktop}}} \leq L_{S_{(k+1)}},$$

where O_C, O_N, O_S are the CPU, network, and storage utilization respectively due to other processes, eg. monitoring software etc., running within that remote desktop session. The expressions on the left side of the comparison operator in the equations above represent the resource requirements for the remote desktop as captured by the remote desktop session performance models. The expressions on the right side of the comparison operator represent the actual resource utilization by the current set of applications and the requested new application. The admission check is thus to compare that, if the given application is admitted, then the total resources allocated for the remote desktop session can continue to satisfy the resource requirements of all the currently running applications as well as that of the requested application.

The Session Admission Control system makes the admission decisions assuming the pre-determined application performance models. The system is complemented with a closed loop control system which would obtain the monitored resource utilizations of the applications and take appropriate enforcement actions in case of violations from the consumption expected as per the application performance model. Other existing research efforts and systems like [14] are addressing this closed loop control system.

3. SIMULATION

We have built a simulation toolkit for the utility system that can handle mixed(heterogeneous) batch and remote desktop session requests, and have implemented our proposed resource allocation service into the toolkit. We have not at the moment implemented the session admission control system into the simulator and assume that the application requests arrive during runtime in the same execution order as assumed at the Resource Management Server. Each blade server is modeled as having two network interfaces - one for the display traffic for interactive sessions to the end-user's thin client, and the other for storage traffic to file servers. We also model the end-to-end network bandwidth and latency between the blade server and the end-user submission nodes, as well as the end-to-end storage bandwidth and latency between the blade servers and the file servers. For both batch and interactive requests, we assume in the current implementation that the requests are picked from the Input Queues as First Come First Served (FCFS) semantics with no priorities. The batch requests are assigned blade servers using a Least Loaded algorithm, and the requests for remote desktop sessions are assigned blade servers using the Multi Variable Best Fit algorithm as described in the earlier section. The site admission control system implementation for a remote desktop session request checks for performance criterion described in Section 2.1.3. For a batch request, we check if there is a minimum required threshold CPU utilization available on a blade server. During the simulation, the CPU utilization for a batch request is guaranteed at least the the minimum threshold and is allowed to exceed the minimum threshold only in case of available CPU cycles. The resource utilizations for the remote desktop sessions are always guaranteed to be equal to that of the value decided through the resource requirement modeling of the remote desktop session.

Using our simulator, one can design and perform various interesting experiments some of which are: (i) Evaluating the trade offs of various resource sharing strategies among mixed workloads (batch and remote desktop sessions): We can have experiments comparing complete sharing of resources among mixed workloads, with those that partition resources among the workloads, and with those with no sharing at all, (ii) Evaluating the proposed resource allocation strategy for interactive remote desktop session work-

Request type	CPU Utilization	End-to-end network bandwidth for display traffic	End-to-end storage bandwidth	Duration in wall clock time
'Heavy Remote Desktop Session'	15% guaranteed on a 2 GHz machine	15 Mbps	150 Mbps	6 hours
'Light Remote Desktop Session'	10% guaranteed on a 2 GHz machine	10 Mbps	100 Mbps	1 hour
'Heavy Batch Job'	Minimum threshold of 35% on a 2 GHz machine	0 Mbps	300 Mbps	4 hours at 35% CPU Utilization on a 2 GHz machine
'Light Batch Job'	Minimum threshold of 5% on a 2 GHz machine	0 Mbps	100 Mbps	3 hours at 5% CPU Utilization on a 2 GHz machine

Figure 8: Application and Remote Desktop Session Performance Models for the experiments

Experiment Type	Remote Desktop Interactive Session Requests	Batch Job Requests	Arrival Rate for Interactive Session requests	Arrival Rate for Batch Job requests
Day Time Experim (12 hours)	'Heavy Remote Desktop Session' requests only	'Light Batch Job' requests only	Poisson distribution; requests arrive at 6 hours into the experiment	Poisson distribution; requests arrive throughout the 12 hour experiment
Night Time Experim (12 hours)	'Light Remote Desktop Session' requests only	'Heavy Batch Job' requests only	Poisson distribution; requests arrive throughout the 12 hour experiment	All requests arrive in a batch at the beginning of experiment (Bursty arrival at time 0 of the experiment)

Figure 9: Request description

loads with naive solutions that don't use performance models and/or admission control, (ii) Evaluating the trade offs among various weight assignments in the resource assignment system, (iv) Evaluating the resource allocation strategies under inaccurate application performance models, (v) Evaluating capacity planning strategies through evaluation of system utilization under various workloads and different resource allocation and sharing strategies.

We have performed some experiments for (i). We evaluated the trade off of throughput and wait time for a mixed workload between a completely shared resource sharing strategy for the batch and interactive session workload, with a no sharing strategy. The application performance model and request characteristics for the experiments are shown in Figures 8,9. The results are summarized in Figure 10, 11. Due to lack of space, we do not elaborate here much on the details. In summary, the results in the Figures show that for a reasonable set of requests, a single system of blade servers is able to handle a particular class of mixed heterogeneous DayTime and NightTime requests of batch and interactive session requests without a very significant degradation in overall performance for the system. Such a system would thus be more cost effective than building separate grids for those batch jobs and interactive sessions respectively. We would be conducting more experiments for other classes of requests and more heterogeneous data centers to see the performance effect.

4. CONCLUSIONS

Our proposed architecture is addressing the needs of supporting remote desktop sessions in emerging Utility Grids. The architecture serves as a conceptual guide for building resource allocation services in such systems. The key features is that it enables virtualization, uses application performance models, generates the remote desktop session performance model dynamically as a composition of individual application performance models, uses dynamic real-time utilization values for dynamic resource allocation, and supports the resource allocation needs for remote desktop sessions throughout its lifecycle including at runtime. We have also built a simulation toolkit and implemented the resource allocation architecture into it. Various experiments are possible using the toolkit. We showed some initial results on trade offs among resource sharing strategies among mixed workload of batch and interactive remote desktop sessions. As future work, we would be doing more

Day Time experiment with No Resource Sharing among mixed workloads			
	100 batch jobs on dedicated 100 nodes	200 batch jobs on dedicated 100 nodes	500 'Heavy' Interactive Remote Desktop Sessions on dedicated 100 nodes
Throughput: Finish Time in minutes	728	730	716
Max Waiting time (minutes)	0	0	0

Day Time experiment with Complete Resource Sharing among mixed workloads		
	100 batch jobs and 500 'Heavy' Interactive Remote Desktop Sessions on 100 shared nodes	200 batch jobs and 500 'Heavy' Interactive Remote Desktop Sessions on 100 shared nodes
Throughput: Finish Time in minutes	728 (Batch jobs) and 722 (Interactive Sessions)	730 (Batch Jobs) and 724 (Interactive Sessions)
Max Waiting time (minutes)	6	11

Figure 10: Results for Day Time experiments

Night Time experiment with No Resource Sharing among mixed workloads			
	30 'Light' Interactive Remote Desktop Sessions on dedicated 100 nodes	200 'Light' Interactive Remote Desktop Sessions on dedicated 100 nodes	500 Batch Jobs on dedicated 100 nodes
Throughput: Finish Time in minutes	65	779	622

Night Time experiment with Complete Resource Sharing among mixed workloads		
	30 'Light' Interactive Remote Desktop Sessions and 500 batch jobs on 100 shared nodes	200 'Light' Interactive Remote Desktop Sessions and 500 batch jobs on 100 shared nodes
Throughput: Finish Time in minutes	65 (Interactive Sessions) and 660 (Batch Jobs)	779 (Interactive Sessions) and 688 (Batch Jobs)

Figure 11: Results for Night Time experiments

experimental evaluation.

5. REFERENCES

- [1] Citrix. <http://www.citrix.com>.
- [2] Microsoft terminal servers. <http://www.microsoft.com/windowsserver2003/technologies/terminalservices/default.aspx>.
- [3] VNC <http://www.realvnc.com/>
- [4] J. Nabrzyski, J. Schopf, and J. Weglarz. *Grid Resource Management: State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.
- [5] J. Rolia, J. Pruyne, X. Zhu, and M. Arlitt. Grids for enterprise applications. In *Workshop on Job Scheduling Strategies for Parallel Processing*, June 2003.
- [6] B. Urganakar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms, OSDI 2002.
- [7] H. Hlavacs and G. Kotsis. Modeling user behavior: A layered approach, MASCOTS 1999.
- [8] M. Friedrich et al. Stochastic resource prediction and admission for interactive sessions on multimedia servers. In *ACM Multimedia*, 2000.
- [9] G. Haring. On stochastic models of interactive workloads. In *PERFORMANCE '83*, 1983.
- [10] J. Nieh, S. Yang, and N. Novik. Measuring thin-client performance using slow-motion benchmarking. *ACM Trans. Comput. Syst.*, 21(1):87–115, 2003.
- [11] A. Wong and M. Seltzer. Evaluating windows nt terminal server performance. In *Proceedings of the 3rd USENIX Windows NT Symposium*, July 1999.
- [12] B. Schmidt et al. The interactive performance of slim: A stateless, thin-client architecture. In *SOSP*, December 1999.
- [13] Microsoft Corporation. Windows 2000 terminal services capacity planning. *Technical White Paper*, 2000.
- [14] HP Process Resource Manager. <http://http://h30081.www3.hp.com/products/prm/>