Harmony: A Desktop Grid for Delivering Enterprise Computations

Vijay K. Naik[1], Swaminathan Sivasubramanian[2], David Bantz[3], Sriram Krishnan[4]

Abstract

This paper describes *Harmony,* a practical grid infrastructure built using personal computer resources. Harmony addresses the key concerns of end users for responsiveness, privacy and protection by isolating the grid computation in a virtual machine on the PC and by implementing a layered resource management architecture to divert workload to unutilized computers from those currently experiencing high levels of interaction. The use of a virtual machine separates the interactive workload software environment from that for the grid workload. Harmony also addresses the key concerns of enterprise IT by automating initial resource assignment and by automatically reallocating workload so as to meet quality of service goals. We have implemented a prototype of Harmony and demonstrated its capability to protect interactive performance. Our preferred grid workload is transactional – a key characteristic of commercial applications. The implementation uses Web Service-based interfaces, so the programming model of Harmony is compatible with and familiar to enterprise developers. We believe that Harmony demonstrates practical exploitation of a hitherto underutilized resource of considerable capability, with the potential to complement, or even in some cases replace, dedicated server-based resources.

1. Introduction

Modern organizations, both academic and industrial, depend on network-attached personal desktop and mobile computer systems. A characteristic of these systems is that they are relatively resource rich (in terms of CPU power, memory, and disk capacity) but are utilized only for a fraction of the time during a day. Even during the time they are in use, their average utilization is much less than their peak capacity. For example, we measured the CPU utilization of developers' desktops in our labs. We found that the average CPU utilization for Intel Pentium III 866 MHz machines running Windows OS to be less than 10% during working hours and close to zero during non-working hours. These systems represent significant computing resources that are underutilized.

Harnessing these underutilized computational resources for organization-wide computing needs, however, has not been practical. The difficulties arise primarily from the following reasons: (i) integrity concerns of desktop owners; (ii) performance impacts to desktop users; (iii) unpredictability in the performance of grid computations; and, (iv) the complexity of the infrastructure needed to harness the idle resources. Standards-based grid technology [1, 2] addresses only the last of these concerns. As a practical matter, grid based computing is more mature on Linux, whereas almost all desktops run the Microsoft Windows OS.

In this paper, we describe a virtual machine-based grid infrastructure called *Harmony* developed in our lab. Its design addresses all of the above difficulties, while adhering to the grid standards. Grid computations are executed in hypervisor-based virtual machines, such as supported by the VMWare Workstation application [3]. In Harmony, execution of grid computations is orchestrated such that their performance impact on workstation responsiveness is negligible and the integrity of the workstation environment is maintained even in the presence of a faulty or malicious grid computation.

The objective and design of the proposed infrastructure differs from the popular grid infrastructure models. In our system, the granularity of resource availability is much

[1] Corresponding Author: IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, E-mail: vkn@us.ibm.com

[2] Dept. of Computer Science, Vrije Universiteit, Amsterdam, 1081 HV, The Netherlands, E-mail: swami@cs.vu.nl

[3] IBM T. J. Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, E-mail: dbantz@us.ibm.com

[4] Dept of Computer Science, Indiana University, Bloomington, IN 47405, E-mail: srikrish@cs.indiana.edu

finer than that of the latter. This is because the availability of a desktop is governed by the local policies set by the desktop user, which typically is giving higher priority to their interactive workload. Such a situation calls for a resource management solution capable of hiding the variability in the resource availability among grid nodes, while striving to maintain a reasonable level of Quality of Service (QoS) to the grid users at the same time.

Another difference in our approach and the traditional Grid approach is that we target our infrastructure towards handling of transactional Grid workloads. Such workloads differ from batch type of computations in granularity and interactivity. Transactional workloads are typically finely grained and are much more interactive than the latter.

The contributions of this paper are as follows: (i) we describe our novel grid-based infrastructure developed using desktop resources and identify the issues involved in building it, (ii) we describe the resource management mechanisms that we have developed to simultaneously preserve desktop interactivity and maintain grid service availability and (iii) show the feasibility of concepts discussed here by means of our prototype implementation, which delivers grid services using Web service protocols and interfaces.

The rest of the paper is organized as follows: We outline our layered system architecture in Section 2 and present our integrated management infrastructure in Section 3. We describe our experience with building this architecture in Section 4. We discuss the related work in Section 5 and conclude with a brief discussion on future directions in Section 6.

2. Architecture Overview

Our architecture is driven by two objectives: (i) Deploy and enable transactional services (e.g., financial, accounting, billing, customer relations, or supply-chain management related transactions) as grid services[5] and (ii) Use desktop based resources to provision these transactional services. Availability and responsiveness to client demands are the key criteria that a transactional service provider must meet. The primary figure-of-merit (i.e., expected QoS) for such services is throughput and response time. This means the architecture should be able to deliver a requested service on demand from the clients and it should be able to adjust the capacity of each service so as to meet the intensity of the demand. The client requests can be complex (e.g., requests resulting in a workflow), request arrival rates can be unpredictable, and clients may have multiple levels of

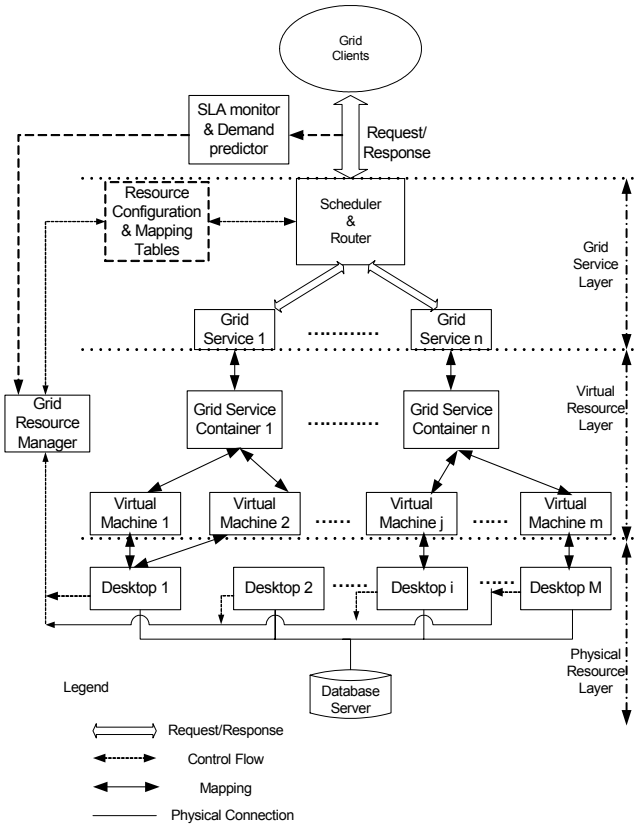

Figure 1: Three-layer architecture for the Peer-based Enterprise Grid: (i) Grid Service Layer, (ii) Logical Resource Layer, and (iii) Physical Resource Layer.

service-level-agreements (SLA) with the service provider. The architecture needs to address these requirements.

The use of desktop-based resources gives rise to a different set of requirements. The primary purpose of desktops is to serve the desktop users by providing a high degree of interactivity and responsiveness. These resources are to be used to provision the transactional services according to some policy defined by the desktop user or by system administrators. Each desktop may have a unique local policy, which may change over time. Examples of local desktop policies include: (i) interactive workload always has the highest priority, (ii) allocate no more than a certain percent of the desktop resources to grid services at any given time, (iii) dedicate certain fraction of the resources for grid computations, (iv) allow participation in the grid computations only during certain time of the day or on certain days of the week. Thus, policy enforcement requires evaluation of certain conditions, which may be static and predictable or dynamic and unpredictable such as the current interactive workload. Moreover, policies may be defined using a combination of static and dynamic conditions. The architecture needs to take into account policies and the heterogeneity in the capacities associated with each desktop resource while addressing the availability, throughput, and responsiveness requirements associated with the transactional services. Rest of this

[5] In this paper, the term *grid service* refers to the software entity that can be invoked remotely over the network and shared among multiple clients of that service. A grid service may be invoked directly in response to a client request or it may provide management or a support service needed to generate the response.

section gives an overview of our architecture and the extent to which it addresses these requirements. We also point out specific design points we have chosen for Harmony, which is an instance of this architecture.

The architecture is defined using a layered approach. This allows addressing the requirements of grid workload and of transactional grid services separately from the requirements of interactive workload and desktop related policies. The architecture, as shown in Figure 1, has three layers: (i) The Grid Service Layer, (ii) The Logical Resource Layer, and (iii) The Physical Resource Layer.

Each layer is associated with Control and Management Components (CMCs). The interactions among the CMCs and the functionality they provide largely define the architecture.

The SLA Monitor and Demand Predictor, shown in Figure 1 is one such CMC. This component monitors request arrivals per Grid service type and per Grid client class basis. It also monitors SLA violations on a per client basis. In addition, predictions on future arrival rates are made for each Grid service type. Based on the predicted arrivals and available Grid service capabilities, a scheduling strategy for request processing is adopted to meet the SLA requirements. This process is repeated frequently as arrival patterns change and/or as the Grid service capabilities change. Some of examples of scheduling strategies are weighted round robin, priority based scheduling (with priorities derived from SLAs), one-to-many scheduling (i.e., simultaneous processing of a request on multiple Grid service instance to overcome uncertainties in service capabilities), and so on.

The CMCs in the Physical Resource Layer enforce desktop related policies, monitor and analyze the interactive workload, and predict the short range availability and capability of the desktop system for a particular Grid service. The CMCs in the Logical Resource Layer act as coordinators between the Grid Service Layer and Physical Resource Layer. We now describe some of the salient points of each layer.

2.1 Grid Service Layer

The Grid Service Layer is concerned with deploying and provisioning transactional grid services in response to requests from grid clients or from other grid services. This layer is also concerned with managing multiple instances of a grid service in response to current or anticipated grid workload and the routing of client requests to appropriate service instances so as to meet the QoS requirements.

The grid service layer consists of an entry point to the grid and a mechanism to invoke the grid services in response to requests from grid clients. The entry point to the grid is referred to as the *Gateway*. Grid clients direct their requests to the Gateway, which then repackages and reroutes those requests internally to an appropriate grid service instance where the request is processed. The response is returned to the Gateway, which then routes it back to the original grid client. In Figure 1, the SLA Monitor & Demand Predictor Component, the Scheduler & Router component, and the collection of Grid Services constitute the Grid Service Layer.

The Scheduler & Router component in the Gateway provides three main functions: (i) Determining the types of services to deploy and the number of instances of each type to deploy at any given time, (ii) the order in which client requests are to be served and the number of requests to be served simultaneously, and (iii) routing grid client requests to service instances in a transparent manner. The Gateway may also perform a few other functions such as grid client authentication and authorization, tracking the status of client requests, and providing service orchestration, if necessary. For a more detailed discussion on the Grid Service Layer, we refer interested readers to [4].

In Harmony, which is an instance of this architecture, there is one logical Gateway. However, for achieving scalability, multiple physical entry points may be provided, with client traffic distributed roughly equally among these physical Gateways, transparent to clients. For the purpose of this paper, we consider the Harmony architecture that consists of one logical Gateway, which maps onto one physical Gateway.

The grid services respond to requests from grid clients or from other grid services. In case of a transactional service, interactions with one or more database services are an essential part of the service. To streamline these interactions (from development, deployment, fault tolerance, and administration point of view), a trend in enterprise computing is towards the use of standard container technologies. To conform to this trend, we use J2EE compliant grid services in the form of Web Services and industry standard containers that support Enterprise Java Beans (EJBs) and Web Services.

In our architecture, each grid service provides a web service interface, which can be described by WSDL, and is amenable to SOAP based interactions (over HTTP or JMS). Multiple web services deployed in one or more containers may combine to form a single grid service (e.g., workflow management). In addition, the same service may be deployed in more than one container and collectively these services may serve one or more grid clients. The database servers maintain consistency among related grid client requests.

2.2 Logical Resource Layer

The logical resources are the resources on which grid services are deployed. These in turn are mapped onto the physical resources made available by the Physical Resource Layer. The role of this layer is to mask the changes in the Physical Resource Layer from the Grid Service Layer.

In Figure 1, the Logical Resource Layer consists of the Containers for deploying grid services, the Virtual Machines (VM) where the Web Service Containers run, and the Virtual Machine Manager (VMM) that runs inside each Virtual Machine. The Web Service Container supports a multi-threaded environment and is capable of providing simultaneous service (i.e., provisioning of multiple service instances) to multiple requests from grid clients. When deploying grid service instances, the Gateway takes into account the state and capacity of the Containers and of the corresponding VMs. This is done in coordination with the VMMs.

Another important role of the layer is to satisfy the constraint of providing isolation between the grid computations and the local, non-grid computations taking place on the desktop. In Harmony, this is accomplished by virtualizing the desktop resources using hypervisor-based technology. On each desktop, a VM complete with its own OS, is deployed using the virtualized resources of that desktop. The operating system, as seen by the end user of the desktop, is referred to as the *host OS* and the operating system on top of the virtualized resources, is referred to as the *guest OS*. The hypervisor provides the separation between the guest OS and the host OS. To the host OS, all the activities associated with the guest OS are viewed as part of a single application with a single context. On the other hand, computations inside the guest OS, which may support multi-tasking, take place *as if* the guest OS were running directly on top of the physical resources.

To describe the role played by the Logical Resource Layer, in Figure 2 we present a schematic of a desktop with a VM running on top of a hypervisor. The VM participates as a logical resource and has a guest OS possibly different from the host OS. As seen in Figure 2, two types of resource managers are involved: Host Agent, and Virtual Machine Manager (VMM). We will describe role of the Host Agent shortly. Together they are collectively responsible for enforcing the local policy of the desktop and to provide the resource availability estimate of the desktop.

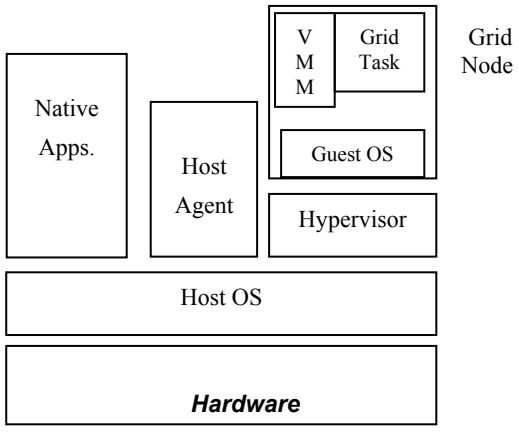

Figure 2: A schematic of a desktop with one virtual machine running with its own guest OS. The Virtual Machine Manager and the Host Agent provide the management controls at the Logical and Physical Resource Layers, respectively.

In Harmony, VMM runs as a privileged service or daemon in the guest OS. It is responsible for collecting and reporting resource usage information of its VM. It also continuously collects the policy and predicted resource availability information from the Host Agent for its desktop. The collected resource availability information is reported to the Grid Resource Manager (GRM), described in Section 3.1. This information is used in predicting availability and capacity of the virtual and the physical resource at a future time interval.

VMM is also responsible for (i) instantiating the grid service Containers, (ii) deployment of specific grid services inside a specific Container, and (iii) deployment of any batch/stand-alone jobs outside the Containers. The VMM performs these actions upon receiving a request from the Gateway. It adjusts the priority levels of grid services within the Container or the VM, whenever the current priority levels are observed to be inadequate in meeting the guaranteed service level agreements. On the other hand, when the grid usage policies set by the desktop users are found to be in violation, VMM may lower the priority of the grid services or may even terminate the grid services.

2.3 Physical Resource Layer

The Physical Resource Layer consists of the resources associated with network-attached desktops and workstations. The resources of primary concern are CPU, memory, storage (temporary and permanent), and network bandwidth. Resources may join and leave this layer dynamically. Typically, the owner of a resource may dictate when the resource may participate in the grid environment. This could be done by defining a policy or by direct intervention. The logical resource layer virtualizes the physical resources before making them available for deploying grid services. Collectively, the physical resource layer forms the basis for all the resources available to the grid and, ultimately, the quality of the services delivered by the grid layer is determined by the quality of the resources available in this layer.

In general, there is a many-to-one mapping between logical resources in and the physical resources. This mapping may change dynamically as the resources join and leave the Physical Resource Layer.

The CMC in the Physical Layer is the Host Agent as shown Figure 2. The Host Agent runs as a privileged service in the host OS and provides current resource usage information to the VMM. The host agent is a key component in enforcing the local policy as it monitors the resource usage by the desktop owner and the VM and ensures that the local policy is not being violated. For example, if the local policy dictates that grid computations can be active only if there is no interactive workload, then the host agent monitors the CPU and memory consumption of both the workloads. If

the interactive workload starts, then host agent sends a signal to the VMM to suspend any grid computations.

The Host Agent also estimates the future resource availability of a physical resource (e.g., based on time-series analysis of past usage patterns) and send these predictions to the VMM. Finally, it monitors the status of individual VMs, checkpoint them and restart them, if they fail and thus, ensuring the continuous availability of a virtualized resource.

3. Integration Across the Three Layers

In the desktop-based enterprise grid considered here, the Management System has to enforce the local policies and, simultaneously has to ensure that there are adequate physical resources for the logical resources to map onto, so that grid services can be delivered with a required level of QoS to grid clients. This requires coordination among the three layers discussed above. In the architecture, as shown in Figure 1, Grid Resource Manager (GRM) integrates the control and monitoring information flowing across the three layers. In the following, we first explain GRM architecture and functionality in some detail and then explain how actions of GRM bring about the desired effects.

3.1 Grid Resource Manager

GRM is a Control and Management Component that operates across the three layers. It acts as the coordinating component across the three layers. It is a logically centralized entity that may be scaled up by organizing in a hierarchical manner. GRM facilitates (i) desktop resource discovery, (ii) detection of resource availability and unavailability, (iii) detection of resource capability (e.g.., which resource is capable of deploying a certain type of grid service), and (iv) allocation of desktop resources to fulfill predicted demand.

GRM keeps track of (i) current grid workload, (ii) expected grid workload in the near future, (iii) logical resources needed to meet the QoS requirements associated a given grid workload, (iv) current available physical resources and their capabilities, and (v) expected availability and capabilities of physical resources in the near future. Information related to current and expected grid workload and the QoS requirements is obtained from the SLA Monitor & Demand Predictor component of the Gateway. Information related to current availability and capability of physical resources is obtained from individual VMMs and Host Agents. It then normalizes the raw capacity of the desktop against a standard platform. In case the desktop node is to be shared among multiple Grid services, it further reduces the available capacity in proportion to the share made available to other Grid services. This represents the maximum normalized capacity available to a particular Grid service. It uses built-in heuristics and algorithms to compute the effects of policies on the future availability and capabilities of the physical resources. This is used in computing the predicted available capacity, at a future time, from a desktop resource for a Grid service. It also computes the uncertainty in each prediction.

Based on the expected grid workload, GRM creates a list of required grid services and computes the number of instances needed for each type of grid service. It also determines the number and type of logical resources needed to run the grid services. The QoS requirements associated with a grid service determine the capacities of the logical resources needed to deploy the grid service. Using the information on the predicted availability and capabilities of physical resources, GRM creates an approximate mapping between the logical resources needed and the physical resources predicted to be available, so that the requirements and capabilities match as closely as possible. Since there are several hard problems involved in this process, GRM makes use of heuristics and approximations. Because of the uncertainties in the predictions, exact solutions are not necessarily worth the cost.

GRM communicates the information on grid services and their mapping onto logical resources to corresponding VMMs as well as to the Scheduler & Router component of the Grid Service Layer. It informs each VMM the maximum number of service instances of a Grid service to deploy. It informs the Scheduler & Router, the number of instances of each grid service available at its disposal, their locations, predicted QoS for each instance, and the uncertainty in the predictions. This resource allocation and mapping information is represented by the Resource Configuration & Mapping Tables shown in Figure 1. It also provides a mapping between logical and physical resources. GRM continuously updates these tables as new information becomes available. As grid workload changes or as the availability of physical resources changes, new instances of logical resources are created or destroyed and the underlying mappings are updated by the GRM. This information is setup in the Scheduler such that client requests are scheduled onto a particular instance of a grid service without taking into account the variability in the underlying mappings.

The Scheduler uses this information to determine the number of service instances to deploy for each Grid service for which it anticipates demand. The number of instances deployed is proportional to the allocated capacity and to the expected demand. When requests arrive, the Router routes those requests to the physical resources where the service instance is actually deployed.

The Scheduler also takes into account the uncertainty in the predicted allocations. When the uncertainty is high, it may decide to schedule a request on more than one service instance simultaneously, making sure that the service instances are mapped on-to distinct physical resources. In such cases, the Router replicates a request and multicasts it to multiple instances of the same Grid service.

For a more detailed discussion on the Grid Service Layer and its interactions with GRM, we refer interested readers to [4].

Since the number of resources (logical and physical) in the grid can be large, management of these resources can introduce a scalability problem. However, this can be handled by adopting a hierarchical control structure for GRM. One way to achieve this is by dividing the large pool of desktops into small pools, each pool managed by a lower-level GRM. A higher-level GRM manages a group of these intermediate-level GRMs.

3.2 Organization in Harmony

In Harmony, we only consider desktop-based resources. Monitoring of desktop resources is performed by Host Agents. As mentioned earlier, each Host Agent provides data on past usage patterns of at the desktop as well as the QoS delivered. Past usage patterns can be analyzed to make predictions with certain level of confidence. Note that because of the collective nature of the desktop resources and with high number of participating desktops, estimates for the individual desktop resources need not be highly accurate. As long as there are sufficiently large number of desktop systems that are potential candidates for participation in the grid computations, and with sufficient variation in the usage across the available pool of desktop resources, estimates on the availability of resources can be made with high degree of confidence. However, this requires an efficient mechanism to detect physical resources that are currently available and then map logical resources onto the available physical resources, so the grid services can be delivered at the desired throughput levels.

To realize the above described goals, the Harmony management system performs the following functions: (i) using monitoring and analysis, estimate the resource availability on each desktop; (ii) map logical resources onto physical resources and schedule grid computations on the grid service nodes; (iii) monitor individual desktop resources to ensure that the scheduled grid computations do not violate the local desktop policies. To perform these functions, the Harmony management system uses a hierarchy of controls and monitoring mechanisms. The resulting layered architecture is shown in Figure 1.

Thus, the layered resource management architecture described above enables separation of concerns as well as simplifies many of the design issues. Control strategies in one layer can be changed without affecting the structure of the resource managers in other layers. For example, a change in desktop resource prediction mechanism will not affect the design and implementation of the grid resource management layer.

In case of transactional workload, each transaction needs to be scheduled based on the transaction requirements and service level agreements with the grid client. For example, a transaction may need to be processed within a certain amount of time after its arrival. This includes any queuing delays at the server. To handle such service guarantees, we use a *service request handler* that selects and routes service requests to appropriate grid nodes so that the specified QoS requirement is met with high probability.

4. Our Experience with Harmony

4.1 Implementation

We have built a prototype implementation of Harmony using desktops used by developers in our lab. Each desktop runs a VmWare Workstation with Linux OS as the guest OS. On each guest OS, IBM's WebSphere Application Server AEs 4.0 [5] is instantiated as a service container. On each application server, web services are deployed. The web services are described using WSDL documents and use SOAP-based interactions over HTTP.

A dedicated server is used for the instantiating the Gateway. The same server is used to run GRM. Another dedicated server is used to run IBM DB2 server (DB2 UDB Enterprise Edition). When transactions are processed in a Grid Service node, an EJB is in the service node interacts with this backend database server.

The GRM is implemented as a web service, with VMMs updating their resource information by making web service calls to the GRM. VMM is implemented as a stand-alone java program. The Gateway's service request handler is implemented by modifying the Apache SOAP RPC Router [6]. The RPC Router receives requests from clients for Grid services and routes the requests based on a routing table populated by the GRM. The WebSphere Application servers running in VMs process the service requests and the results are returned to the Gateway. The Gateway's service request handler receives the result, repacks and sends it back to the client. As noted in Section 3, for reasons of scalability, multiple instances of service request handler can be run.

The Host Agent is implemented using C++ and monitors the processor utilization using Windows kernel APIs, such as QuerySystemInformation(). The host agent monitors the overall processor utilization and the individual processor utilization of VmWare process, thereby deducing the processor utilization of the interactive workload.

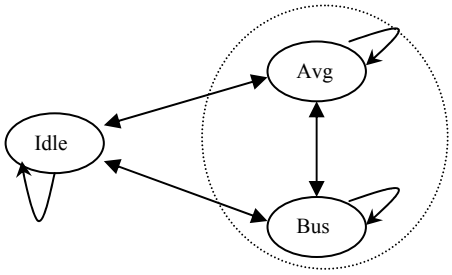

Figure 3: Workload model used to predict the resource availability information. Idle – maximum resource availability, Average load – average resource availability, Busy state – minimum resource availability

The host agent models the interactive workload utilization as a Markov chain with three states: Idle, Average, and Busy (Figure 3). The host agent updates the transition probabilities among the three states by observing its interactive workload behavior. From this model, the host agent predicts the probability for the system to be "idle" from its current state. For example, if an user works with only bursts of busy workload and leaving the system idle rest of the time, the transition probability from idle to idle will be high. The accuracy of this simple prediction scheme increases as it learns more about the desktop user's workload pattern. We are currently investigating complex prediction scheme that uses different Markov transition probability matrices for different times of a day. To ensure maximum possible responsiveness to the interactive workload, we set the priority of VmWare process to be low. Hence, when a new interactive workload is instantiated, the VmWare process gets swapped out.

We have deployed a few transaction-based applications with Web service interfaces on the desktops. One such example is a WorkOrder Management service, which creates/updates/schedules the work orders of an organization by querying it with the database containing its existing order information.

In our model, since Grid computations are encapsulated as web services, a computation-oriented job, if wrapped by an EJB, can be run in any of the grid nodes, with VMM handling the initial deployment of the EJBs.

Although the database services themselves could be deployed on top of the VMs provided by the logical resource layer, for performance reasons we confine the database servers to dedicated backend resources.

4.2 Performance Results

The performance of our Grid infrastructure is measured by the *throughput* as seen by Grid clients. Throughput is calculated as the number of client service requests processed by the grid per unit time. We studied the performance of our grid infrastructure for two different scenarios: (i) the effect of number of grid nodes available on the throughput and (ii) the effect of varying desktop availability on the throughput.

We have performed the experiments with following machines: 1 dedicated IBM eServer 1Ghz running RedHat Linux 7.1 with I GB RAM, 4 desktops with 900 Mhz Pentium III processor and 256 MB RAM running Windows XP as host OS and RedHat Linux 7.1 as its guest OS.

4.2.1 Effect of number of grid nodes

In this experiment, we study the effect on throughput as a function of the number of desktop nodes participating in the grid. For this experiment, we created a traffic generator that generates grid service requests, with an exponential inter-arrival time. We studied the average throughput for various arrival rates for different number of desktop nodes. We first measured the grid throughput when a single dedicated server handles the requests. We repeated the same experiments for cases when more desktop nodes (2 and 4) participate in the grid and the results are given in Figure 4.

As expected, the grid throughput increases with increase in the number of nodes, since more nodes are available to process the client requests. It can be seen that, for an arrival rate of 20 requests/sec, the throughput increases from 1.1 (for a single server case) to 3.5 (with addition of four desktop nodes). However, the improvement in throughput does not increase linearly with the increase in the number of grid nodes, which might be due to the difference in computational capacities of individual desktop nodes and the dedicated server. This experiment shows that desktop nodes are capable of running enterprise transaction workloads and can improve the overall throughput.

The scenario depicted in this experiment, where the throughput is improved by adding more desktop nodes to the grid, in addition to a dedicated server, is representative of a real-life case, where an overloaded server is offloaded by diverting the requests to a backup server. In such a case, the desktop grid can act as a backup server in taking the service requests when the dedicated server is overloaded.

4.2.2 Effect of variability in resource availability

The first experiment just confirmed the improvement in the throughput with increase in the number of nodes. However, it did not discuss about the effect of variability in the desktop availability on the grid throughput. In this experiment, we study the effect of variation of desktop availability on the throughput.

We varied the availability of 3 desktop nodes, such that on an average only one of these desktop nodes are available to the grid at a given time, while the others run a busy computing workload. We kept 2 other nodes dedicated to the grid. Hence, in both setups, the number of virtual resources available to the grid is the same (3). However, in the first case, the 3 virtual resources maps on the 3 dedicated physical resources, while in the second case the

mapping of 3rd virtual resource changes dynamically with change in the availability of the 3 desktops. This study will test the sensitivity of our GRM to changes in resource availability. The results are given in Figure 5.

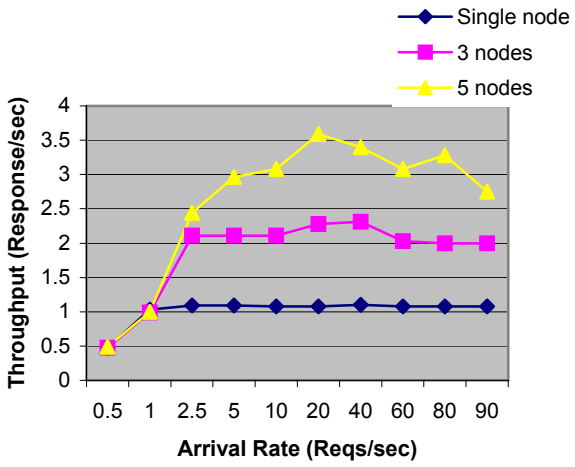

Figure 4: Effect of arrival rate on throughput for different number of grid nodes

As seen in Figure 5, the difference between the throughput between the dedicated case and that of the variable case is visible only during higher arrival rates, and even for those cases, the difference in throughput is not large. This shows the effectiveness of our control architecture in masking the variability of resources..

Another important performance issue in our architecture is the overhead introduced by running computations on VmWare instead of native platform. In [7], the authors have shown that the overhead of running computations inside VmWare to be less than 10%. Our experience confirms this observation. Thus, by using a virtual machine to perform grid computations we pay only a small penalty in terms of performance, but realize high benefits in terms of security, isolation, and control.

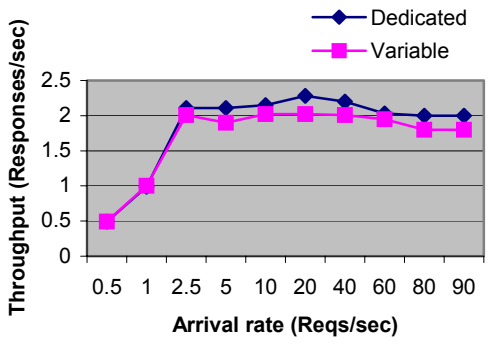

Figure 5: Effect of variability in resources on throughput

5. Related Work

5.1 OGSA and Globus Toolkit

Our work has several similarities with certain services provided by the Globus Toolkit. However, there are key differences as well. Globus provides resource allocation and management of active jobs via the Grid Resource Allocation and Manager (GRAM) [8]. It also provides standard mechanisms for publishing and retrieving resource status and configuration information via the Monitoring and Discovery Service (MDS) [9]. GRAM can be configured such that it sends resource information to MDS, and clients can then query MDS to find suitable resources. In our system, the GRM is responsible for storing and retrieving resource information, and can be thought of as our replacement for MDS. The VMMs are responsible for resource allocation and reporting information to the GRM, and hence can be thought of as our replacement for GRAM.

Conceivably, we could have implemented our GRM and VMMs using MDS and GRAM respectively. The GRM could act as a thin wrapper around MDS, and could retrieve resource information from it in order to create and send routing tables to the Gateway. GRAM could be used instead of the VMM, with certain modifications to send prediction information to the MDS along with observed resource information. However, the stable versions of Globus (2.x) do not have explicit support for Web services and their hosting environments, such as WebSphere. This was the major motivating factor for our approach.

With the advent of OGSA, the Grid is moving towards a Web services based approach. Since our GRM and VMMs are already Web services, we do not foresee any major hindrances in making our system OGSA compatible. In the future, we plan to be compatible with OGSA and modify our GRM and VMMs such that they use the OGSA compatible versions of MDS and GRAM respectively. In addition, all our Grid services will obey the two-level naming scheme suggested by OGSA, with each of them having a unique Grid Service Handle (GSH) and a Grid Service Reference (GSR).

5.2 Peer to Peer Computing

The goal of Peer to Peer (p2p) technologies is to leverage idle cycles and storage at the edges of the Internet [10]. Its focus on decentralization, instability, and fault tolerance exemplifies areas that have been omitted from emerging Grid standards, but will become more significant as the system grows [11]. Typical p2p systems provide solutions for categories of applications, such as file sharing (e.g. Gnutella), distributed computation (e.g. SETI@home), and anonymity (e.g. Freenet). Our system is similar to the p2p systems that provide distributed computation by leveraging idle cycles from available computational resources on the internet. However, it differs from most such systems in the sense that it does not restrict itself to jobs of any particular

kind, e.g. searching for extra-terrestrial intelligence (as in SETI@home). In addition, the transactional workloads we support require higher interactivity with the Grid clients, unlike other traditional p2p distributed computing systems.

5.3 Other Systems

In a typical grid environment, grid management services are provided to mask resource management related issues from the grid user. To the grid user, resources appear as if they are part of a homogeneous cluster and are managed in a dedicated manner for the user, when in fact the resources may be widely distributed, loosely coupled, and may have variable availability and response time characteristics. Grid management services [1, 2, 12] attempt to keep track of the resources and services delivered and try to match the demand with the supply. As long as the available supply of resources exceeds the demand, the grid services only have to manage the mapping of the resources to the consumers of the resources. Today many efforts [8, 13, 14, 15] are focused in streamlining the process of searching for grid resources and towards managing and monitoring of the resources, so that meaningful service level agreements can be set and achieved. This scenario plays out well only when resources are dedicated for delivering grid services, whereas in our system the resources are not assumed to be dedicated and vary in instantaneous availability.

The Xenoservers described in [16] provide a secure infrastructure for running untrusted applications. Similar to our work, Xenoservers are hypervisor based. However, in the case of Xenoservers, the entire OS runs over an hypervisor, which makes the approach unsuitable for capturing the idle cycles from a desktop system. Moreover, the approach described in this paper is targeted towards use of persistent grid services that participate in grid computations when the underlying desktop resource allows them to do so. The infrastructure described in [16] is targeted more towards conventional grid applications that are instantiated along with a grid user request.

We note here that the basic objectives of our project are similar to other Distributed Processing Systems (DPS) such as Condor [17] and Legion [18], in terms of utilizing the computational power of idle workstations. A novelty of our architecture is that it offers better host resource control for transactional applications, where individual service times are short but overall throughput over a large number of requests is important. By using persistent grid services and by directing the grid service request traffic to appropriate grid service, throughput of the grid computations can be controlled easily even when the underlying resources keep moving in and out of the grid pool. Furthermore, the virtual machines are self-contained and can be easily managed by the host OS, facilitating migration of computation, check-pointing and recovery. A similar PC-based grid infrastructure built by Entropia, called DCGrid platform [19]. DCGrid platform provides a secure platform for executing native Win-32 applications. The platform guarantees isolation of execution of external applications through a secure technology and provide job-scheduling schemes to preserve the interactivity of the desktops. However, the usage of virtual machines as in our architecture, in addition to preserving the integrity of the desktop also provides a computational environment, wherein each VM can be treated as an individual machine by itself. This enables us to potentially run any kind of application (windows or Linux applications, by running different OSes in different VMs) and services, such as web services in an easier manner.

Grid systems suitable for commercial and transactional applications are described in [20] and [21]. In [20], the authors describe an "on demand" use of grid resources to offload peak workload from dedicated web servers to idle servers in a proxy grid. They also discuss scheduling and traffic modeling related performance issues in the context of commercial applications. In [21], the authors describe a grid system that manages its resources so as to conform to the service level agreements (SLA) between grid customers and grid service providers. In both [20] and [21], dedicated set of servers are assumed for forming grid systems, where as our work focuses on scavenging idle resources from desktop systems. Nevertheless, there are several complimentary aspects. In particular, traffic monitoring for SLA enforcements is one such area and use of shared resources is another complementary area. Recall from Figure 1 that our architecture incorporates a component called SLA Monitor and Demand predictor. However, in Harmony, we have not implemented this component. On the other hand, the resource management infrastructure in Harmony is geared towards using shared resources – shared across virtual grid systems or shared among grid and non-grid applications.

6. Conclusions and Future Work

In this paper, we have presented a grid architecture suitable for deploying transactional workload using idle resources from desktop systems. This architecture is built on grid standards using virtual machines running in workstations and desktop PCs. We have designed and implemented Harmony, which is an instance of this architecture. Harmony is capable of utilizing the idle computational and memory resources of workstations, even those in active use.

Our infrastructure has the following advantages:

i. Limited Intrusion: Grid computations run only on virtual machines, any malicious code/error will not have any effect on the host applications and OS.

ii. Fault tolerance: As virtual machines can be easily and effectively check pointed and migrated to another node.

iii. Host OS independence: Candidate nodes need not run the same host OS; VMs permit the guest OS to be chosen independently of the host.

iv. Ease of developing and deploying applications: Since grid nodes are homogeneous, grid computations need not be written for different platforms and any grid computation can be deployed to any grid node.

We are also planning to extend our architecture to utilize the resources provided by departmental and enterprise servers, where the resource availability of the servers is governed by their local policies. We are also investigating other resource prediction algorithms to increase the accuracy and efficiency of the grid resource availability forecaster.

Despite the fact that our experiments were performed on a limited number of desktop nodes, our design is scalable due to our scalable resource management structure, via the use of a hierarchical system of GRMs and Gateways. We are planning to conduct more experiments with larger number of participating desktop nodes in order to confirm this theory.

REFERENCES

[1] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," http://www.globus.org/research/papers/ogsa.pdf, as of Nov. 2002.

[2] I. Foster, C. Kesselman, (eds.), "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, 1999.

[3] "VMWare Workstation's Manual," http:// www. vmware. com/ support/ ws3/docs/, as of Nov. 2002.

[4] V. Naik, S. Sivasubramanian, and S. Krishnan, "Adaptive Resource Management and Workload Scheduling for a Peer Grid," IBM Research Report RC 22839, July, 2003.

[5] IBM WebSphere, http://www.ibm.com/websphere/, as of Nov. 2002.

[6] Apache SOAP Documentation, http://xml.apache.org/soap/docs/index.html, as of Nov. 2002

[7] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes, "A Case for Grid Computing on Virtual Machines," Technical Report TR-ACIS-02-001, University of Florida, August 2002.

[8] K. Czajkowski, I. Foster, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," *In. Proc.* IPPS/SPDP '98, Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 62-82, 1998.*

[9] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. "A Directory Service for Configuring High-Performance Distributed Computations", *In Proc.* 6th IEEE Symposium on High-Performance Distributed Computing, *pp. 365-375,* 1997.

[10] C. Shirky. "What is P2P And what isn't?" http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html, as of June 2003.

[11] J. Ledlie, J. Shneidman, M. Seltzer, and J. Huth, "Scooped, Again", *In Proc,* 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Feb 2003.

[12] 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.

[13] B. Lee and J. B. Weissman, "An Adaptive Service Grid Architecture Using Dynamic Replica Management", *In Proc.* 2nd Intl. Workshop on Grid Computing, Nov. 2001.

[14] R. Buyya, D. Abramson, J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," *In Proc. of* The 4th International Conference on High Performance Computing in Asia-Pacific Region, Beijing, China, May 2000.

[15] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," International Journal of Software: Practice and Experience, May 2002.

[16] D.Reed, I. Pratt, P. Menage, S. Early, N. Stratford, "Xenoservers: Accounted execution of untrusted code," *In Proc.* IEEE Hot Topics in Operating Systems VII, Mar. 1999.

[17] Michael Litzkow, Miron Livny, and Matt Mutka, "Condor - A Hunter of Idle Workstations", *In Proc.* 8th International Conference of Distributed Computing Systems*,* pp. 104-111, June, 1988.

[18] A. S. Grimshaw, et. al., "The Legion Vision of a Worldwide Virtual Computer," Communications of the ACM *January,* 40(1), 1997.

[19] ''DCGrid Platform", http://www.entropia.com

[20] C. Crawford, D. Dias, A. Iyengar, M. Novaes, and L. Zhang, "Commercial Applications of Grid Computing," IBM Research Report RC 22702, January 2003.

[21] A. Leff, J. Rayfield, and D. Dias, "Service-Level Agreements and Commercial Grids," IEEE Internet Computing, Special Issue on Grid Computing, 7(4), July-August, 2003.