RC23943 (W0604-125) April 25, 2006
Computer Science

IBM Research Report

Architecture for Service Oriented Solution Delivery Using Grid Systems

Vijay K. Naik, Pawel Garbacki*, Ajay Mohindra
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

*Delft University of Technology

Architecture for Service Oriented Solution Delivery Using Grid Systems

Vijay K. Naik
IBM T. J. Watson Res Center
vkn@us.ibm.com

Paweł Garbacki
Delft University of Technology
p.j.garbacki@tudelft.nl

Ajay Mohindra
IBM T. J. Watson Res Center
ajaym@us.ibm.com

Abstract

Today, service delivery environments are increasingly required to handle complex service requests requiring configuration and deployment of customized solutions. Managing and delivering such services using grid based infrastructure has many attractions including flexible resource management framework, the ability to handle heterogeneous workload and heterogeneous resources, and the ability to incorporate and share resources from multiple administrative and policy domains. However, existing grid systems lack critical system management services such as automated configuration, deployment, and life-cycle services that are taken for granted on other large scale systems. In this paper, we describe Harmony II — an architecture for delivering customized solutions using an approach that is amenable to automation in configuration and deployment of solutions on grid resources. These automations are made possible by using resource virtualization, workload and resource capacity predictions, and on-line matching of workload requirements with available resource capacities.

1. Introduction

In an IT service delivery environment, clients submit requests for services and the requested service is delivered to the requestor as per the client specification or according to a predetermined service delivery specification. In such environments an IT solution is delivered in response to a request such that the requestor is not involved in the configuration, deployment, and any other aspect of the life-cycle management of preparing, delivering, and termination of the service. This means the requestor does not need to know the internal details of the solution configuration or deployment. Typically, the service delivered is not an out-of-box solution, but requires some customization and integration. It may require configuration and integration of one or more applications, middleware components, and subsystems of OS, network, and file systems. Ideally, all such management aspects should handled transparent to the requestor by the entities managing the service delivery.

The separation of service management aspects from service specification and consumption aspects has opened up many opportunities. Service users can leverage this abstraction to realize complex service compositions and think in terms of requesting and receiving complex business solutions on-demand to suit business needs. At the same time, service delivery vendors can think of managing the requested solutions in terms of managing workflows of large number of integrated tasks. By pooling together requests from multiple customers, vendors can realize resource and skill pool consolidation. They can also benefit from the economies of scale by processing and delivering IT services to a large pool of customers by using standardized service components. Realizing this in practice, however, involves creating customer specific solutions that need to be configured, deployed, and managed on-demand. Standardizing these practices is not straightforward.

Grid abstractions provide the flexibility required for handling heterogeneous set of service requests and servicing the components using a common set of resources. Grid abstractions also provide the ability to harvest and aggregate resources from multiple administrative domains to achieve delivery specific business goals such as high throughput, preferential treatment of client requests, and so on. Grid resources belonging to individual administrative domains can enforce their own policies for usage, sharing, and collaboration. Service provisioning on shared resources allows aggregation of resources on-demand and provides the ability to control the utilization of resources. From a management and administration point of view, grid model is highly desirable since control points can be defined and policies can be applied to control individual resource usages and the quality of service delivered. Thus, the grid computing model can be directly adopted in an environment consisting of multiple workloads each requiring customized IT services using a common set of resources. Furthermore, if the fluctuations in the workloads are not correlated, then with grid computing model it is possible to achieve higher operational efficiencies and utilization than is possible by using dedicated resources for each workload.

While there are compelling reasons to adopt the grid model in a request oriented service delivery environment, in practice, there are many challenges that need to be overcome before realizing the full potentials of grid computing in a service delivery environment. One difficulty is the lack of grid mechanisms for automating configuration and deployment operations. Another difficulty is in controlling resource sharing by grid workload. A third difficulty is the absence of services for detecting and handling faults in the physical resources. For commercial workloads, these important characteristics are taken for granted on traditional platforms such as the mainframe systems.

In this paper, we describe a framework called Harmony II that provides a platform for service execution on a grid infrastructure using standardized service management practices. In the context of the grid resource sharing model, Harmony II can be viewed as an extension of the grid framework optimized for a high level of automation of the deployment and provisioning of customized services. Harmony II does not replace existing grid middleware, but rather builds on top of it.

The seamless integration of Harmony II with the existing grid infrastructures is possible through the use of resource virtualization. Service executions are orchestrated such that their performance impact on each other and external workloads is kept within the limits prescribed by the resource usage policies even in the presence of faulty or malicious services. With the use of multiple, customizable virtual machine templates, we allow the delivered services to fully customize their execution environments, which are no longer dependent on the fixed configuration of the grid resources.

The rest of the paper is structured as follows. The overview of the Harmony II system architecture is provided in Section 2. Section 3 discusses the automation aspects of service management in our architecture. The most important details regarding the Harmony II implementation are given in Section 4. The results of the performance evaluation are presented in Section 5. Finally, the concluding remarks and prospects for the future work are described in Section 6.

2. Harmony II Architecture

Harmony II architecture is guided by the following requirements: (i) Support for heterogeneous service requests, (ii) Support for heterogeneous resources, (iii) Support for grid-based service configuration, deployment, and provisioning, (iv) Optimization of system throughput, resource utilization, and service availability, (v) Flexibility in defining local resource usage policies, and (vi) Adaptability to changes in resource availability and client demands.

As shown in Figure 1, Harmony II architecture consists of three functional components: (i) Service Provisioning Infrastructure, (ii) Service and Resource Management Infrastructure, and (iii) System Data. We describe these in some detail in the following.

2.1. Service Provisioning Infrastructure

Service Provisioning Infrastructure consists of four layers that manage the dependencies between the service instances, the virtual resources that they occupy, and the physical grid resources that host the virtual resources.

The services provided by Harmony II are invoked by either batch or interactive applications called here the *Service Clients*. The complexity of the underlying distributed service providing infrastructure is hidden from the service clients behind the abstraction of the Access Layer encapsulated by the *Gateway* component. Gateway is a well known access point to the system where service clients direct their requests. Those requests are then repacked and rerouted to an appropriate service instance where the requests are processed. The responses are returned to the Gateway, which then routes then back to the service client. The client request routing is performed by the Gateway in a transparent manner, meaning that clients do not have any influence on or knowledge of the selection of the service instance that will handle the request. The Gateway may also perform a few other functions such as service client authentication and authorization, tracking the status of client requests in case of asynchronous service invocations, and providing service orchestration, if necessary. In Harmony II, there is one logical Gateway. However, for achieving scalability, multiple physical entry points may be provided, with client traffic balanced among these physical Gateways, in a way transparent to the clients. For the purpose of this paper, we consider the system architecture that consists of one logical Gateway, which maps onto one physical Gateway.

The service instances, which form the Service Layer, are not deployed directly on the physical resources, but are rather embedded inside Virtual Machines (VM). Services may have multiple instances deployed inside different VMs. Depending on the policy defined by the services and compatibility issues between service deployments, multiple service instances may reside inside a single VM. Deploying multiple service instances inside one VM trades the service environment isolation aspects offered by the single-service deployments for the physical resource capacity savings brought by reducing the number of VMs in the system.

The Virtual Resources Layer consists of the virtualized resources and the associated control infrastructure. Every VM in our system is controlled from inside the VM by the *Virtual Machine Manager (VMM)*. The VMM, which runs as a privileged process or a daemon, is responsible for collecting the CPU, memory, and disk resources usage of its VM. When the resource consumption of the controlled VM exceeds an acceptable level, the VMM performs a graceful shutdown of the VM.

The Physical Resources Layer consists of the resources

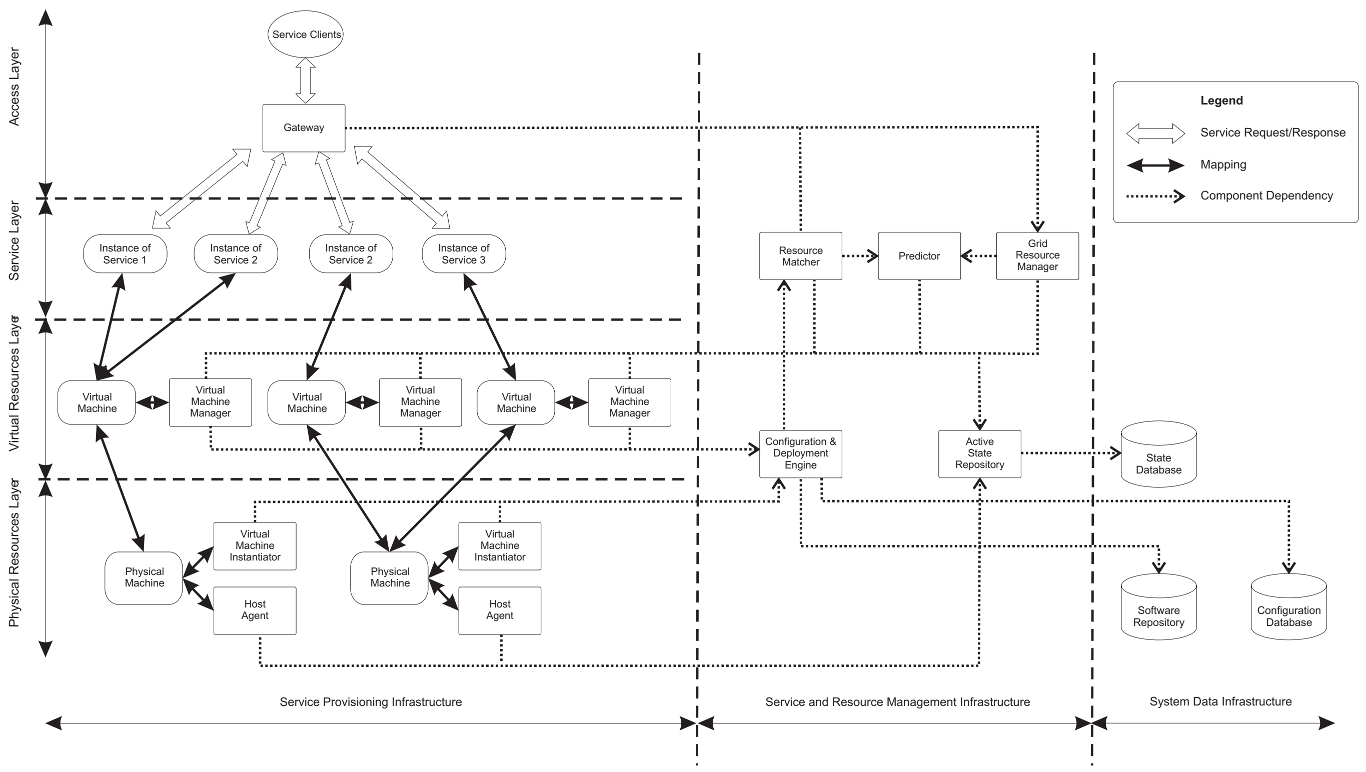

Figure 1. The components of Harmony II architecture.

associated with Physical Machines. Resources may join and leave this layer dynamically. Typically, the owner of a resource determines when the resource may participate in the grid environment, and puts a limit on the acceptable load exercised by the service computations. This could be done by defining a policy or by a direct intervention. Collectively, the Physical Resources Layer forms the basis for all the resources available to the services and, ultimately, the quality of the services delivered by Harmony II is determined by the quality of the resources available in the Physical Resources Layer.

The key component in enforcing policies of physical resource usage is the *Host Agent (HA)*. HA is as a privileged process running directly on the physical resource OS, monitoring the CPU, memory, and disk resources usage by the VMs and ensuring that none of the local policies is being violated. The specific task of instantiating VMs on physical resources is assigned to *Virtual Machine Instantiator (VMI)*. VMI is not a constantly running service, but rather an application which is activated only when a new VM is to be created. Consequently, in a stable situation, VMI does not consume any of the local resources.

2.2. Service and Resource Management Infrastructure

In a grid environment as the one considered here, the management infrastructure has to ensure that there are adequate physical resources available to host the virtual machines while enforcing local policies, and at the same time fulfill certain QoS requirements. Providing this functionality requires coordination of the actions taken at different layers of the Service Provisioning Infrastructure. In Harmony II , the components providing the management functionality across the Service Provisioning Infrastructure layers are collectively called Service and Resource Management Infrastructure.

The resource monitors (HA and VMM) report the collected usage statistics to the *Active State Repository (ASR)*. AST provides a set of interfaces that give the decision making components access to the gathered system data. To reduce the amount of data transferred over the network, ASR partially processes the data locally and sends only the computed statistics over the network.

The *Predictor* generates forecasts of the future service workload characteristics and resource availability based on the current system state as well as historical data. Forecasting of grid resource usages has been shown to be a difficult problem [7]. Service behavior may vary over the time, and there is no single prediction algorithm that fits all different service workloads. With generality in mind, we have integrated with our predictor a wide range of forecasting algorithms, starting with simple methods such as running mean or exponential smoothing, to end up with current state-of-the-art approaches such as ARIMA, Holt Winters, FFT, Wavelet, or Kalman Filter. For each prediction method, we measure its accuracy in a certain context, e.g., estimation of a load exercised by the clients of a particular service, and select the most reliable method for this context. In this respect, our prediction approach is similar to the one adopted in the Network Weather Service [11].

The responsibility of the *Resource Matcher* is to select the physical machine that will host a service instance embedded inside a VM. The selection of the physical resource is done based on two factors: the compatibility with service requirements and predicted resource capacity. The compatibility aspects address the suitability of the resource hardware/software configuration to host the service instance. The resource capacity predictions, on the other hand, help to ascertain that the resource will be able to handle certain level of service load in the future. For details on the architecture and design such a Resource Matcher, we refer the interested reader to [8].

The variety of the resources, so typical in a grid environment, in combination with the diversity of the supported services requires a highly flexible solution for configuring these services on the grid resources. In Harmony II, the component that customizes the process of instantiation of new VMs and installation of services and dependent software components inside those VMs is the *Configuration and Deployment Engine (CDE)*. The customization of the configuration and deployment process is achieved through the use of deployment scripts with detailed description of the installation procedure. Deployment scripts have a form of XML documents and are, thus, portable across multiple operating systems.

The *Grid Resource Manager (GRM)* deals with the high level QoS aspects of the service provisioning infrastructure. The objective of GRM is to guarantee that there are enough resources allocated to services to meet certain QoS requirements, while ensuring that the service workload does not violate the resource usage policies. To achieve those objectives, GRM determines the amount of the physical resource capacity that should be assigned to each service and defines how client requests should be distributed among available service instances. The responsibility of the GRM is limited to defining the system-wide policies directing the Service Provisioning Infrastructure adaptation, not enforcing these policies, which is left to other components such as Gateway, Configuration and Deployment Engine, and Resource Matcher.

2.3. System Data Infrastructure

The System Data Infrastructure consists of the databases and repositories storing the static information about service requirements and configuration instructions, as well as dynamic system state.

The *Software Repository* provides the software packages, e.g., application server binaries or database drivers,

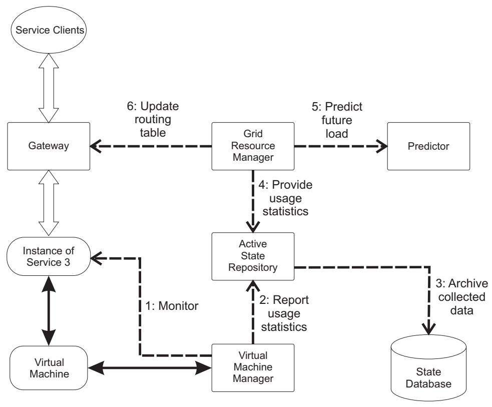

Figure 2. Interactions between workload management components.

which are required by the service runtime. The Software Repository is accessed whenever a new service needs to be configured inside a target VM. The repository provides all the software packages that have to be installed on the target VM before deploying the service.

The *Configuration Database* contains description of the inter-software package dependencies and special-case configuration options that are translated by the CDE to configuration commands in a deployment script. The information stored in the Configuration Database is service specific in the sense that it describes for each service the configuration options that conform to the requirements of that service.

State Database provides a backend storage for the statistics collected by the Active State Repository. Current and historical resource usage characteristics, service client arrivals, loads of service instances are the types of data stored in the State Database.

3. Automation of Service Management

The automation of service management is provided in our system at three levels: (i) adaptive workload management, (ii) resource discovery, virtualization and aggregation, and (iii) on-demand service provisioning.

3.1. Adaptive Workload Management

The requests of service clients are routed taking into account the changes in the resource availability as well as service level agreements between resource owners and service providers.

Figure 2 presents the interactions between the components involved in the adaptive workload management. The Virtual Machine Manager monitors the services deployed inside its VM, collecting information about the virtual resource usage (interaction 1 in Figure 2). The collected resource usage statistics are periodically reported by the VMM at the Active State Repository (2), and are eventu-

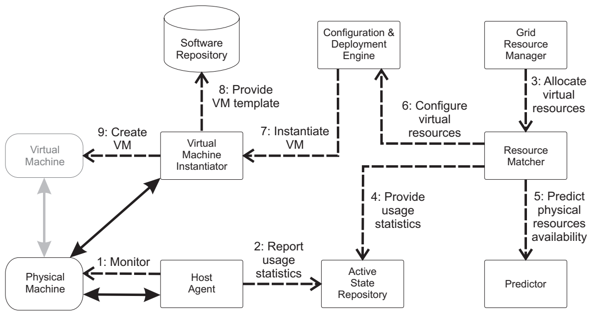
Software Repository
Configuration & Deployment Engine
Grid Resource Manager
8: Provide VM template
3: Allocate virtual resources
6: Configure virtual resources
9: Create VM
7: Instantiate VM
Virtual Machine
Virtual Machine Instantiator
Resource Matcher
4: Provide usage statistics
5: Predict physical resources availability
2: Report usage statistics
1: Monitor
Physical Machine
Host Agent
Active State Repository
Predictor

Figure 3. Interactions between resource discovery, virtualization and aggregation components.

ally archived in the State Database (3). The resource usage statistics are continually analyzed by the Grid Resource Manager (4). Predictor plays an important role in this analysis, providing estimates of the future load (5). The forecasted resource load combined with the information on the resource usage policies, provide a basis for constructing client request routing rules for the Gateway (6). These rules are described by a data structure called routing table. The routing table contains a list of all service instances with assigned weights. The weight represents a fraction of client requests that should be directed to this particular service instance.

3.2. Automated Resource Discovery, Virtualization and Aggregation

Resources in Harmony II are discovered and virtualized only when they are ready to share capacity. The resource administrator has the full authority to define the conditions under which a resource is allowed to host services. Harmony II supports services that require a set of resources (e.g., one resource may be the database, the other — the machine running the application server). In general, services are provisioned with the aggregation of resources satisfying the service requirements.

Figure 3 depicts the interactions between components providing the resource discovery, virtualization, and aggregation functionalities. Host Agent continuously monitors local VMs (1) reporting their resource consumption to the Active State Repository (2). At some point, the Grid Resource Manager may decide, based on the request arrival rate estimates, that a new service instance is required to handle the increasing clients' load. In such a case, a message requesting allocation of virtual resources for the new service instance is sent to the Resource Matcher (3). The Physical Machine that will host the VM encapsulating the new service instance is selected by the Resource Matcher based on the predicted resource availability computed from the historical data provided by the ASR (4 and 5). Once the target physical resource is selected, the Configuration and Deployment Engine takes over, and initiates the process of configuring a new VM (6). CDE cooperates with the Virtual Machine Instantiator running on the target physical machine to determine which VM template is compatible with both the physical machine OS and the service which is going to be deployed inside the VM (7). Finally, the selected VM template is fetched from the Software Repository (8), configured, and instantiated (9).

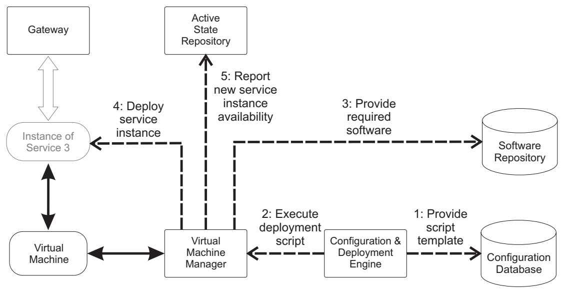
Gateway
Active State Repository
5: Report new service instance availability
4: Deploy service instance
3: Provide required software
Instance of Service 3
Software Repository
2: Execute deployment script
1: Provide script template
Virtual Machine
Virtual Machine Manager
Configuration & Deployment Engine
Configuration Database

Figure 4. Interactions between service provisioning components.

3.3. On-demand Service Provisioning

The number of instances of a particular service depends on two factors: clients demand on this service and the availability of resources which are capable of hosting service instances. In response to increased client demand, new service instances are automatically deployed on available resources.

Figure 4 describes the process of configuring a new service instance. After the VM was instantiated from a template, the Virtual Machine Manager contacts the Configuration and Deployment Engine requesting instructions on how to configure and deploy the service in the virtual environment. CDE constructs a deployment script which describes the steps that have to be followed to configure the service execution environment and to deploy the service. Deployment scripts are created from templates stored in the Configuration Database (1). The deployment script consists of a list of OS independent instructions which are translated to the OS specific commands by the VMM (2). The deployment script may contain instructions to install new software packages, which are then obtained from the Software Repository (3). After the configuration of the virtual environment is finished, the service itself can be deployed (4). Once the service is operational, the Active State Repository is informed about its availability (5). The Grid Resource Manager will include the new service instance in the generated routing table, such that Gateway can start redirecting client requests to this new service instance.

4. Harmony II Implementation

We have developed a prototype implementation of the Harmony II architecture. Here we describe and motivate the selection of particular programming languages and third-party components used by this prototype.

The request router in the Gateway is implemented as a servlet which exploits the filtering technique [6] to intercept the service client requests. The type of the service to be invoked is determined from the information included in the request header (e.g., the SOAP header in case of web service requests made over the SOAP protocol). The request is then redirected to a service instance selected according to the routing table, and the service response is sent back to the client. The redirection is fully transparent to the client who is oblivious to the intermediary proxy. Although currently Gateway supports service requests send over the HTTP protocol only, the extension with additional protocols such as SMTP, JMS or even raw TCP is quite straightforward.

A particular hypervisor solution which we use in Harmony II is the VMWare Workstation [5]. The choice for VMWare is motivated with the simplicity in which VMs can be created and maintained, and the flexibility of changing the VM configuration even after the VM image was created. The VM templates in Harmony II are configured with Red Hat Enterprise Linux as the guest operating system.

Host Agent, as the only component continually running on each physical resource, has to be lightweight so that its local resource consumption is negligible. Keeping this in mind, we have implemented HA in C++ programming language. HA uses a low level operating system API to access the local resource usage statistics. Currently, we provide the HA implementation compatible with the MS Windows family operating systems.

The Virtual Machine Instantiator and Virtual Machine Manager are invoked only when a new VM is created, so they could be implemented with less consideration of minimal resource usage than HA. For portability reasons we have written both the VMI and VMM in the Java programming language.

The components of the Service and Resource Management Infrastructure are encapsulated as web services. These services are described in WSDL and communicate with each other using SOAP over HTTP. The Predictor service uses the advanced statistical models for time series analysis provided by the R-Project [3]. The functionality of the R-Project is accessed through R-Serve [4], a network server which enables remote access to R-Project computation engine. The prediction algorithms are described in a high level mathematical modeling language, the R language. The consequence of this modular design of the Predictor is that a new prediction algorithm can be easily added to the framework without any changes in the Predictor core.

The Software Repository is simply an FTP server that stores the software packages and VM templates in the file system. The State Database is built on the IBM DB2 DBMS. The Configuration Database has a form of a collection of XML-based templates of deployment scripts. The Configuration and Deployment Engine customizes these templates for a particular service and resource instance, converting them into deployment scripts. As a particular format of a deployment script we use the Apache Ant [1] language. The Apache Ant project defines a set of OS-independent primitives which enable installation and configuration of software packages. Moreover, Apache Ant provides a Java-based, OS-portable interpreter of the Ant scripts, which we integrated with the VMM.

5. Evaluation

5.1. Experimental Setup

We evaluate the performance and scalability of Harmony II infrastructure using the following setup: the Gateway, Active State Repository, Grid Resource Manager, and Configuration and Deployment Engine are deployed inside a WebSphere server running on a Windows XP, Pentium IV 3GHz CPU, 1GB RAM machine. The Resource Matcher and Predictor reside on a Windows 2K server with dual Xeon 2.6 GHz CPU and 2.3 GB RAM. The FTP server of the Software Repository is installed on a Red Hat Enterprise Linux host with dual Xeon 2.6 GHz CPU and 1 GB RAM. Both the State and the Configuration Database run on a Win2K, dual Xeon 2.6 GHz CPU and 2.3 GB RAM.

The grid resources are represented by a set of 13 heterogeneous hosts, running 5 different operating systems. The resources of 10 among these hosts are shared between Harmony II services and external workload, and 3 are dedicated to run only the Harmony II services. The shared hosts are the desktops and laptops of developers in our lab. The shared hosts are actively used during our experiments, resulting in non-trivial external load being present on these hosts. The detailed configurations of all the 13 hosts are presented in Table 1. The last column of Table 1 describes the local resource sharing policies defined for the non-dedicated machines. For the purpose of the experiments we use simple policies that set the limits on the number and memory consumptions of the VMs that are allowed to run simultaneously on the physical machine. Additionally, the Host Agents running on the shared hosts do not allow the Harmony II service load (measured in CPU and memory usage) to excess a certain threshold of the total host capacity in the presence of an external load (when the hosts are being actively used by other applications).

To show the diversity of the types of services supported by Harmony II , we configure the experimental environment with three different services. The first service is the Weather Forecast described in [10]. Weather Forecast is a popular application server benchmark implemented as a web service that makes use of a backend database (in our case a DB2 database running on a dedicated dual Xeon 2.6 GHz CPU and 2.3 GB RAM machine, shared by all instances

Host names	Host configuration	VM instantiation policy
Beat1, Beat2	Win2K Server, 2 x Xeon 2.6GHz CPU, 2.3 GB RAM	2 x 512 MB RAM
Beat3	Win2K Server, 2 x Xeon 2.6GHz CPU, 2.3 GB RAM	1 x 512 MB RAM
Beat6	Win2003 Server, 2 x PIII 866MHz CPU, 1 GB RAM	1 x 512 MB RAM, 1 x 384 MB RAM
Ritmo0, Ritmo1, Ritmo2	Win2003 Server, 2 x PIII 866MHz CPU, 3 GB RAM	2 x 512 MB RAM
Beat0	ESX Server, 2 x PIII 900MHz CPU, 1.7 GB RAM	Dedicated machine
Megha-0	WinXP, PIII 1.2 GHz, 1 GB RAM	1 x 512 MB RAM
Jolly-boy	WinXP, PIII 790 MHz, 512 MB RAM	1 x 384 MB RAM
Amsterdam	WinXP, PIII 1.2 GHz, 512 MB RAM	1 x 384 MB RAM
Beat4, Beat5	RedHat EL 3.0, 2 x Xeon 2.4GHz CPU, 1.5 GB RAM	Dedicated machines

Table 1. Configuration of physical and virtual resources used in experiments.

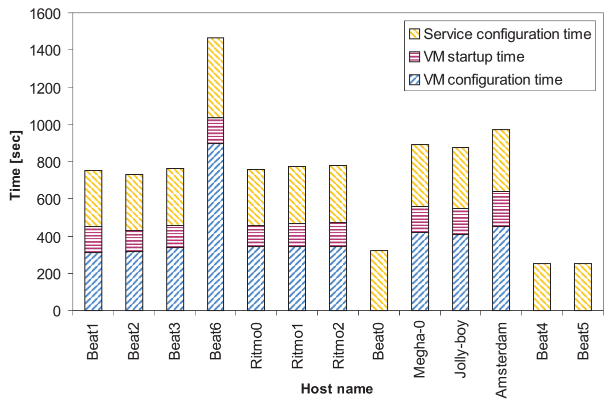

Figure 5. Weather Forecast service deployment times for different hosts.

of the Weather Forecast service). The second service used in our experiments is the BidBuy auction web service from the Jakarta Tomcat test applications suite. The instances of BidBuy run inside Jakarta Tomcat application servers. The third tested service is very different from the previous two as it has a form of an application executed in batch mode rather than as a web service. The application that we use, FracGen [2], generates complex fractals, which is a resource and time consuming process. It is, thus, natural that clients requesting fractal generation do not wait at the Gateway until the computation is finished, but rather disconnect immediately after submitting the fractal generation request, and then periodically check if the results are available. Note, that the batch application is a typical example of a grid job.

5.2. Performance Results

5.2.1. Service Deployment Performance

In the first series of experiments we assess the performance of the service deployment process. For each machine in the simulated grid environment we compute the time required to deploy a new service instance on that machine. As explained in Section 3, service deployment is a multi-stage process. Figure 5 presents the duration of the three stages of the Weather Forecast service deployment measured for the hosts listed in Table 1. The VM configuration time represents the time required to fetch the VM template from the Software Repository and configure it according to the local resource usage policies. The VM startup time describes how long it takes to boot the VM and start the VMM. Finally, the service configuration time is the duration of the service execution environment configuration and service instantiation. The Weather Forecast service, in particular, requires a WebSphere server and DB2 client to be installed and configured inside the VM before the service itself can be instantiated.

For all the machines, but one the total service deployment time is between 10 and 16 minutes. The higher overhead of the Beat6 machine comes from the fact that that machine has a significantly slower network interface, which results in a longer time required to obtain the VM template from the Software Repository. Beat0 is running ESX Server OS which offers a native support for VM technology. Instead of fetching a VM template from the Software Repository each time a new service is deployed, Beat0 keeps a pool of 'empty' VMs that are available for deploying of new services. Consequently, the VM configuration and VM startup phases are excluded from the service deployment process on Beat0. Beat4 and Beat5 are running service instances directly on the host OS. Although we did not describe such a setup in Section 2, the generality of the Harmony II design allows to eliminate the Virtual Resource Layer, which improves the performance of the service deployment and execution. We allow, however, to omit the Virtual Resource Layer only on dedicated machines, where no local resource usage policies are present.

5.2.2. System Throughput

In the second experiment we study the effect of the amount of resources delegated to run service instances on the system throughput. For this experiment, we have created a multi-threaded request generator, that generates service requests at an increasing rate until the system becomes saturated and some requests are dropped. The instances of all three service types, Weather Forecast, BidBuy and FracGen

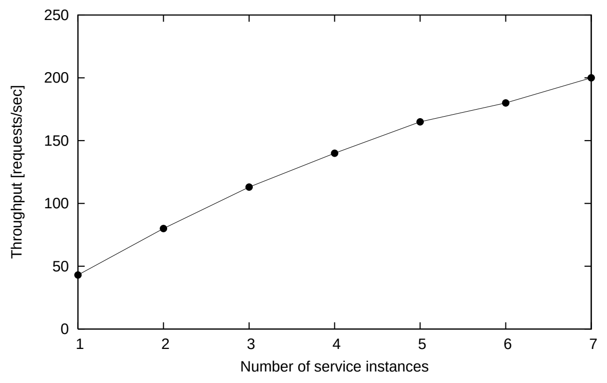

Figure 6. System throughput.

are present in the system during this experiment. The target service for each generated request is selected randomly and uniformly.

Figure 6 presents the throughput as a function of the number of service instances, averaged for the three service types. System throughput increases rapidly with the number of service instances, indicating that Harmony II is capable of handling proportionally higher workload by adding more resources. The fact, that the throughput does not increase linearly with the number of service instances has a few reasons. First, the capacities of individual hosts and VMs differ between each other. Second, there are only 13 physical machines in the system, while in our experiment we configure up to 21 service instances (7 instances for each of the 3 service types). In consequence, multiple service instances share the hosts competing for access to physical resources.

6. Concluding Remarks

In this paper we have described the Harmony II architecture and the design and implementation of a prototype system based on this architecture. The primary goal of the Harmony II system is to deliver customized IT solutions using resources managed as a grid. Based on the client requests, in Harmony II, customized solutions are composed from a standardized set of component templates. The templates provide sufficient flexibility so the composed solutions perform seamlessly. The solution is delivered by deploying the components on resources that are managed as a grid based system. Preliminary performance results from the implemented prototype system indicate that the system is capable of configuring and deploying customized solutions on demand and provide the desired level of automation in delivering requested services.

The Harmony II approach brings automation to the configuration and deployment of service components taking into account current demands on the system as well as the availability and constraints on the underlying physical resources. Our approach does not require service requesters to be aware of how a service is composed or how it is delivered. All service management aspects are handled transparently to the requestor. Similarly, the system provides control points for service delivery teams to prioritize workloads and to control the degree of resource sharing and hence the quality of service delivered to clients.

Existing grid-based systems provide facilities to manage physical resources, but those decisions are exposed to the grid clients. Moreover, the grid clients are required to adopt their requests to the allocated physical grid resources. Existing configuration and deployment mechanisms also tend to be hardwired and specific to the target system. Run-time systems based on such mechanisms tend to be brittle and unable to withstand any failure in the underlying physical infrastructure. Harmony II, on the other hand, maintains a layer of indirection between the physical resources and the service requests in the form of the virtual resource layer. This allows the system to manage fluctuations in the underlying physical infrastructure transparently from the service layer and vice versa. We note here that the work described by Siddiqui et al. has similar goals as ours [9]. However, the focus of their work is on application-level component management to enable automatic deployment of software components. The focus of our work is on service management taking into account generic workloads and a generic set of grid resources.

References

[1] Apache ant project page. http://ant.apache.org/.

[2] Fracgen project page. http://shapeshifter.se/code/fracgen/.

[3] R-project page. http://www.r-project.org/.

[4] R serve project page. http://stats.math.uni-augsburg.de/Rserve/.

[5] Vmware project page. http://www.vmware.com/.

[6] The essentials of filters. Sun Developer Network, March 2005. http://java.sun.com/products/servlet/Filters.html.

[7] N. H. Kapadia, J. A. B. Fortes, and C. E. Brodley. Predictive application-performance modeling in a computational grid environment. In *High Performance Distributed Computing*, Redondo Beach, California, USA, August 1999.

[8] V. Naik, C. Liu, L. Yang, and J. Wagner. Online resource matching in a heterogeneous grid environment. In *IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, Cardiff, UK, May 2005.

[9] M. Siddiqui, A. Villazon, J. Hofer, and T. Fahringer. Glare: A grid activity registration, deployment and provisioning framework. In *Proceedings of SC 2005*, Seattle, WA, November 2005.

[10] U. Wahli, T. Kjaer, B. Robertson, F. Satoh, F.-J. Schneider, W. Szczeponik, and C. Whyley. *Web Services Handbook Development and Deployment*. IBM Red Book, July 2005.

[11] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, March 2003.