Managing Service-Oriented Grids: Experiences from VEGA System Software

YuZhong Sun, Haiyan Yu, JiPing Cai, Li Zha, Yili Gong

Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, China
yuzhongsun@ict.ac.cn

Abstract. Recent work towards a standard based, service oriented Grid represents the convergence of distributed computing technologies from e-Science and e-Commerce communities. A global service management system in such context is required to achieve efficient resource sharing and collaborative service provisioning. This paper presents VEGA system software, a novel grid platform that combines experiences from operating system design and current IP networks. VEGA is distinguished by its two design principles: a) ease of use. By service virtualization, VEGA provides users such a location-independent way to access services that grid applications could transparently benefit from systematic load balancing and error recovery. b) ease of deployment. The architecture of VEGA is fully decentralized without sacrificing efficiency, thanks to the mechanism of resource routing. The concept of grid adapters is developed to make joining and accessing the Grid a 'plug-and-play' process. VEGA has been implemented on Web Service platforms and all its components are OGSI-compliant services. To evaluate the overhead and performance of VEGA, we conducted experiments of two categories of benchmark set in a real size Grid environment. The results have shown that VEGA provides an efficient service discovery and selection framework with a reasonable overhead.

1 Introduction

The Grid has been demonstrated as an efficient approach for providing computational services for various scientific applications [2]. Recently, common demands from the e-Commerce and e-Science communities have forced the convergence of technologies between these two fields. The move towards service-oriented Grids, exemplified by Open Grid Services Architecture [6] .

In a service-oriented architecture (SOA)[7], a management system in charge of service provisioning plays a crucial role in mediating provider-consumer interactions. Challenges for managing a service Grid originate from the nature of the Grid: the diversity of resources and the dynamic behavior of resources.

As the Grid is an open society of resources and users from different application domains, it is not applicable to predefine the unique criteria for classifying resources and all user requirements. Service publishing and matching protocols could be ad hoc or domain specific, making the global service discovery and scheduling more difficult to implement. Resources are often autonomous, which results in their volatile

H. Jin et al. (Eds.): NPC 2004, LNCS 3222, pp. 83-91, 2004.

behaviors. For example, a resource may join or leave the grid at any time without notifying the management system. This makes managing and searching for available resources increasingly complex.

The objective of this paper is to present VEGA [5] (acronym for Versatile services, Enabling intelligence, Global uniformity, and Autonomous control) as a novel approach to deal with these problems. VEGA aims at developing key techniques that are essential for building grid platforms and applications. VEGA has been applied in building a national-wide grid testbed called Chinese National Grid (CNGrid), which targets putting high performance computers in China together to provide a virtual super computing environment.

The rest of the paper is organized as follows. The section 2 introduces the related work. The designing principles of VEGA are depicted in Section 3. The Section 4 describes the architecture. The Evaluation of Grid router is given in Section 5. The Section 6 shows the evaluation of VEGA. The conclusion is depicted in the Section 7.

2 Related Works

A number of research efforts target the management of services for dynamic, heterogeneous computing environment. In web service architecture, meta-information of services are maintained by a centralized repository like UDDI. Services published onto UDDI are location-dependant and dynamic information cannot be reflected.

MDS in Globus has realized a globally uniform naming of distributed resources. In the MDS architecture, information is organized in the strict tree-like topology. The directory service used in MDS is LDAP, which is designed for reading rather than writing. While in grid environments resource may change frequently over time, which could result in a writing bottleneck.

In ICENI, ontology information is annotated with the description of service interfaces, which facilitates the automatic matching and orchestration of services at a semantic level. It leverages Jini technology for dynamic service discovery and publishing with the help of a registry service similar to UDDI.

P2P networks rely on routing techniques for locating resources. In early systems, message flooding is used in which queries are propagated along a dynamically changing path. Unfortunately, you cannot ensure two same queries return the same result set. In later semi-structured systems like CAN, distributed hash table (DHT) algorithm is used for locating a resource within limited hops.

3 Design Principles of VEGA System Software

3.1 Virtualization and Ease of Use

From the users' point of view, resource management should be completely transparent. It is the responsibility of the management system to translate abstract resource requirements into a set of actual resources.

To obtain the full physical resource independent properties in VEGA, we adopt the conception of virtual services and physical services with the Grid Service Space (GSS) model. In this model, we present Virtual Service Space (VSS) and Physical Service Space (PSS) with coessential mapping, scheduling mapping and translating mapping.

In the GSS model, a virtual service is an abstract representation of one or many physical services that have common interfaces. A virtual service can only be mapped to one physical service at one time point. This mapping process is called resource scheduling or resource binding. A programmer can refer to a virtual service by a location-transparent name and hence the application can obtain several benefits such as load balancing (by choosing alternative physical services with lower load), fault tolerance (by switching to a new physical service in response to service failure), locality of service access (by locating a nearer physical service), etc.

3.2 Decentralization and Ease of Deployment

A centralized resource management system is conceptually able to produce very efficient schedules. For the Grid, it is not practicable to build such a centralized entity as resources are owned by various administrative organizations. Conversely, a decentralized structure composed of many management points does scale well with increasing size of the Grid[1][3].

There are two fundamental methods to accomplish resource joining: 1) deploy the service onto a hosting environment and start it 2) register the resource's meta information to the information service so as to make it open to grid users. Normally, node administrators manually do these registering works. However, the Grid may accommodate huge amount of resources varying over time. Manually handling joining and leaving process of those resources could be impossible.

Enlighten by IP network, we designed counterparts of routers and network adapters in a network system, which facilitate automating the discovering, publishing and deployment of resources in a grid environment [4]. In [4], we propose a Routing-Transferring resource discovery model, which includes three basic roles: the resource requester, the resource router and the resource provider. The provider sends its recourse information to a router, which maintains this information in routing tables. When a router receives a resource request form a requester, it checks the routing tables to choose a router for it and transfer it to another router or provider.

4 Architecture

4.1 Layered Logical Overview

Scalability and ease of deployment are two driving forces of our architecture design. We proposed three key components in VEGA: Grid Operating Systems (GOS), Grid Routers (GR) and Grid Adapters (GA). Their responsibilities are similar to the

counterparts of operating systems, IP routers and network adapters in present network systems, respectively.

VEGA is conformed to the OGSA framework. Currently VEGA was implemented based on Globus Toolkit 3, which is used to encapsulate the physical services and physical resources. A Grid Application accesses to virtual services the Grid provides by Vega Grid operating system (GOS) which call the Vega Grid router to discover the Grid or web services. Figure 4.1 shows the layered architecture of VEGA.

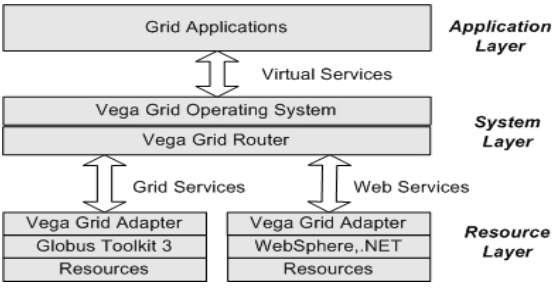

Figure 4.1. The layered architecture of VEGA.

At the resource layer, the distributed resources are encapsulated as grid services or web services hosting by various application servers such as GT3 container and .NET. Like a network adapter, a GA is enabling software for connecting a node machine to the Grid, making services hosted by the node machine known to the nearby GR.

The system layer comprises two components. The interconnection of GRs constitutes an overlay network that provides underlying information service for resource mapping. GRs are capable of routing resource requests to appropriate matching nodes. GOS could aggregate encapsulated physical services together and provide a virtual view for application developers.

At the application layer, developers can use the APIs, utilities and developing environments provided by Vega GOS to build virtual service based applications.

GOS is the kernel of the entire VEGA architecture. It is comprised of job broker service, resource management service, file service, system monitor service, GSML server, and user & CA virtual interface service.

4.2 Runtime Architecture

Figure 4.2 illustrates a runtime architecture of VEGA and information flows when deploying grid resources, locating grid resources as well as using grid resources. Two categories of grid nodes are shown in the figure, one equipped with GA (left outline rectangle) and the other without (right outline rectangle).

A complete service access process from the Grid client-side consists of the following steps: Grid clients submit abstract resource requests to the GOS, then GOS forward the requests to any one GR. The response data will contain a candidate set of physical services. By certain resource selection algorithms, GOS selects one appropriate service from the candidates and deliver subsequent invocations from the client to it. The invocation requests and responses are passed through GOS with the

exception of large bulks of data transferring or other situations where performance becomes critical.

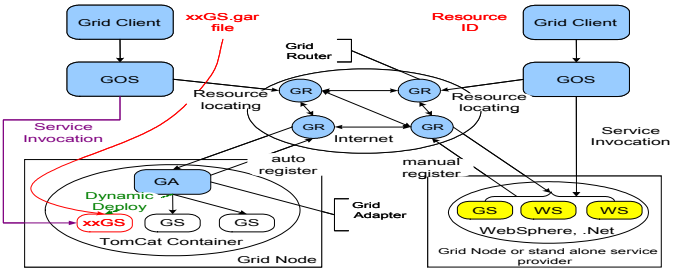

Figure 4.2 A Runtime Architecture of VEGA system software

5 Evaluation of Grid Routers

Grid routers are the backbone of VEGA system, which are transfer stations for resource request. We propose a Routing-Transferring resource discovery model,more details can be get in section 3.2. Figure 5.1 illustrates how a request mr is transferred from router R0 to R2 and eventually U0 finds the resource r which is located on the provider P1.

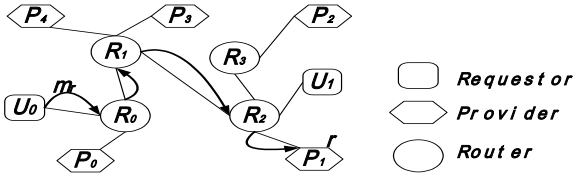

Figure 5.1 The routing-transferring model and the process of locating resource

In [9], based on the router architecture, we propose a three-level fully decentralized and dynamic VEGA Infrastructure for Resource Discovery (VIRD) shown in Figure 5.2. The top level is a backbone consisting of Border Grid Resource Name Servers (BGRNS); the second level is made up of several domains and each domain consists of Grid Resource Name Servers (GRNS); and the third level is leaf layer that include all clients and resource providers. A client can query its designated GRNS in two ways, recursive or iterative, and the server will answer the request by its knowledge about the grid.

We present a resource naming scheme and a link state like algorithm for resource information propagation. The VIRD architecture allows every server has its own data organization, searching and choosing policies.

The analysis shows that the resource discovery time is dependent on topology and distribution of resources. When topology is definite, the performance is determined by resource frequency and location. The result shows that high frequency and even distribution can reduce the resource discovery time greatly.

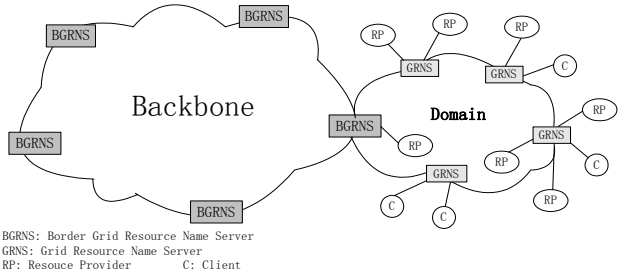

Figure 5.2 The three-level VIRD architecture: a backbone, domains and leaves.

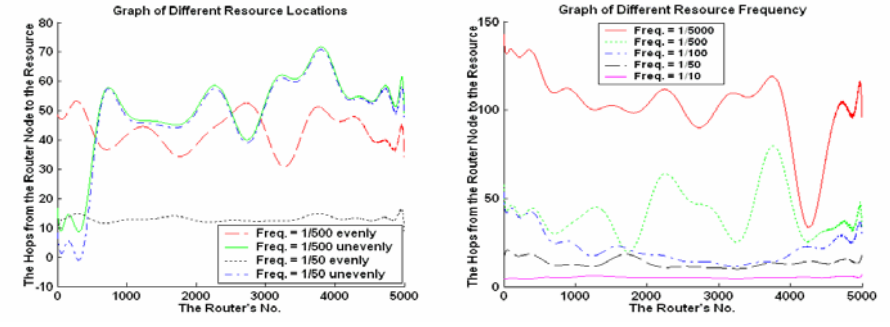

Figure 5.3 Graph of different resource locations and frequency

6 Evaluation of System Overhead and Throughput

In this section, we report preliminary results regarding system overhead and throughput of VEGA. The overhead derives from three aspects: a) The XML-based SOAP protocol overhead. b) key components of VEGA are secure GT3 services that require overhead c) VEGA is written in JAVA. The overall performance is sometimes unstable due to the JVM scheduling mechanisms such as auto garbage collection.

The experiments were done over a non-dedicated network of PC servers in the Institute of Computing Technology (ICT) and National Research Center for Intelligent Computing Systems (NCIC) of China. For simplicity, we deployed one GOS at ICT, two resource routers both at ICT and NCIC. Resource routers connect dozens of nodes mainly PC servers with dual 1800Mhz AMD Athlon processors.

We use two categories of applications as the benchmark set. One is a simple echo application whose execution time on a single machine is so trivial that it can be used to measure the overall system overhead. Another is the notable biology software named BLAST, which is used to measure system throughput.

We divide stages of a job execution into three phases: initialization, execution and ending. The initialization phase starts when the job has just been submitted; it ends when the job's status becomes submitted. The initialization time represents the

overhead of a middle management system. The execution phase is the real execution time of the job. In the ending phase, temporary data or garbage is collected.

Figure 6.1 have shown the affect of different scheduling strategies on the execution time of jobs, and the system overhead. A series of identical job requests are orderly submitted to GOS. The execution time of using load balancing scheduling outperforms using round robin. We observe that with the job index increases, the execution time reduce smoothly. The reason is that since we repeatedly submit the same job, system cache in components takes effect so less time was consumed. It is also showed that the system overhead is relatively large, about 2 and 4 seconds, respectively. This is because the timer on the client side updates each 2 seconds.

Figure 6.1 Execution time of Blast with 1second execution time and one job per time.

Figure 6.2 illustrates the different initialization and ending times for two scheduling strategies, load balance and round-robin. The experiments show the performance of Round-Robin is much better than load-balance. The reason is the load-balance has to poll all the possible sites for online load information. This proves that the Round-Robin outperforms the load-balance in scalability. Another noticeable point is that the stability of Round-Robin is much better than load-balance. The reason is the overhead to poll a site for online load information varies dynamically over time regarding to the system and network situations.

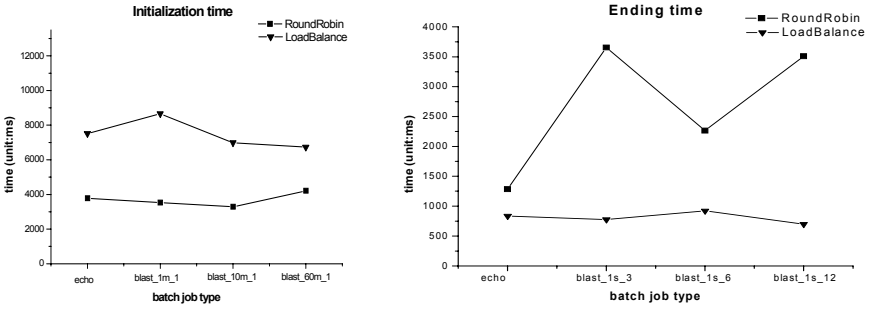

Figure 6.2 Initialization time and ending time of different jobs

Regarding of so many computers in a Grid environment, we should apply as simple load balance algorithms such as Round-Robin as possible. However, Figure 6.3 illustrates that when the scale of Grid is not very much, a job with long enough execution time may suffer the same in the load balance as in the Round-robin in a small or middle scale. According to our experiments, we can suppose randomly selected site algorithm may be as efficient and scalable as Round-Robin. In the future, we should test this kind of load balance strategy.

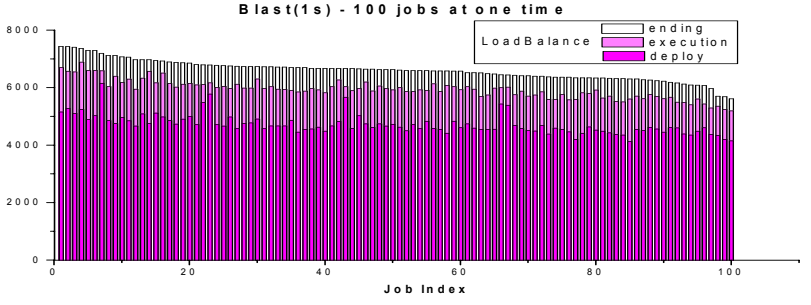

Figure 6.3 Run 100 jobs in parallel on a grid comprised of two servers.

7 Summary

In this paper, we have reviewed the challenging issues on grid service management, descript design principles of VEGA and introduce its architecture. Finally, we give an evaluation of VEGA, including routers, system overhead and throughput.

VEGA combines experiences from operating system design and current IP networks. It is distinguished by its two design principles: a) ease of use, providing users with a location-independent way to access services. b) ease of deployment. The architecture of VEGA is fully decentralized without sacrificing efficiency.

Currently, we are focusing on developing a new version of proof-of-concept prototype of the Vega Grid based on our version 1.0.

References

[1] K. Czajkowski et al., “A Resource Management Architecture for Metacomputing Systems”, Proc. '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp. 62-82, 1998.

[2] K. Czajkowski et al., “Grid Information Services for Distributed Resource Sharing”, Proceedings of the Tenth IEEE International Symposium on HPDC, IEEE Press, 2001.

[3] A. Iamnitchi et al., “On Fully Decentralized Resource Discovery in Grid Environments”, International Workshop on Grid Computing, Denver, November 2001.

[4] W. Li et al., "Grid Resource Discovery Based on a Routing-Transferring Model", 3rd International Workshop on Grid Computing (Grid 2002), pp. 145-156, November 2002.

[5] Z. Xu et al, "Mathematics Education over Internet Based on Vega Grid Technology", Journal of Distance Education Technologies, vol. 1:3, pp. 1-13, July 2003.
[6] I. Foster, et al, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Open Grid Service Infrastructure WG, GGF, 2002.
[7] Randall Perrey, Mark Lycett, "Service-Oriented Architecture," 2003 Symposium on Applications and the Internet Workshops (SAINT'03 Workshops), 2003.
[8] Wei Li, Zhiwei Xu, Li Cha, Haiyan Yu, Jie Qiu, Yanzhe Zhang, "A Service Management Scheme for Grid Systems," GCC2003, 2003.
[9] Yili Gong, Fangpeng Dong, Wei Li, Zhiwei Xu, "A Dynamic Resource Discovery Framework in Distributed Environments," GCC2002, 2002