A Scalable and Adaptive Cloud-based Message Brokering Service

Frank Yong-Kyung OH*, Shin-gyu KIM*, Hyeonsang EOM*, Heon Y. YEOM*
Jongwon PARK**, Yongwoo LEE**
*School of Computer Science and Engineerig, Seoul National University, Korea
**School of Electrical and Computer Engineering, University of Seoul, Korea
{ykoh, sgkim, hseom, yeom}@dcslab.snu.ac.kr, {comics77, ywlee}@uos.ac.kr

***Abstract*—Recently, message brokering services that are based on publish and subscribe paradigm are widely adopted in many areas such as remote facilities monitoring. Typically, they support a large scale distributed network of exchanging messages between services and clients. In this paper, we introduce a cloud-based message brokering service which allows better scalability and dynamic load balancing between brokers. Our system framework monitors each virtual machine (VM)'s resources and in order to meet the service level agreement (SLA), a VM is rented or recollected automatically by the system. The framework also provides the best available broker that a client can use for connections. Our experimental results show that our system framework can achieve up to 58.60% of scalability improvement at the maximum number of subscriber connections.**

***Keywords*—NaradaBrokering, Cloud Computing, Virtualization, Xen, Scaliability**

I. INTRODUCTION

The systems supporting publish and subscribe paradigm have been developed as a messaging infrastructure for collaboration, Grid applications, and peer-to-peer (P2P) networks. In the past decade, the systems like this have matured significantly through extensive functional testing with various configurations in both academic and industrial areas. In particular, NaradaBrokering [1, 2, 3] has gained major role in Grid, Web Service, P2P, and Audio/Video Conferencing. Unlike in a traditional centralized client/server network, NaradaBrokering allows publisher and subscriber to exchange data in transparent fashion. Publisher and subscriber are independent and do not need to know anything about each other for delivering data. Data are delivered to its destination by the brokers. NaradaBrokering not only provides reliable data delivery from source to destination, but also provides many different features such as message encryption, compression and fragmentation, multiple transport protocols and addressing, and fault tolerance related features [4].

NaradaBrokering truly provides distributed service by allowing subscribers to connect to a broker nearest to the client's location for using network service. However, concentration of subscribers to a specific broker may lead to very poor network bandwidth utilization and affect latencies associated with other services [5, 6]. In order to circumvent the problem of concentrated connections, NaradaBrokering provides a broker locator to limit the concurrent connections at a broker node and let subsequent connections to be forked off to the best available broker for balancing the overall connections. Nevertheless, the entire service may be affected in case entire broker network is overloaded by too many subscribers. We could add additional brokers to the network, but this must be done manually by system administrators and also there is no resource monitoring feature within NaradaBroker systems. Systems like Tele-Management System [7] where NaradaBrokering is mainly used for communication may have critical performance issues.

Cloud computing like Amazon's EC2 brings many key services to customers over the Internet. Depending on the need of computing resources and services, virtual machines are dynamically provisioned and service use is charged on a utility computing basis. If we simply offer NaradaBrokering as a service within the cloud, we can take advantage of its self-service or self-management and full pack distributed and parallel computing powers [8].

In this paper, we present a cloud-based solution to address aforementioned problems of NaradaBrokering networks. In case continuous incoming requests overload the entire systems, a pre-configured virtual machine (VM) can be automatically added in the cloud platform infrastructure when the load reaches pre-defined threshold. To have this kind of system framework, the following two functions are needed: ability to monitor system resource (CPU, memory, and network I/O) usage and automatic VM allocation/deallocation. And also load-balancing feature is necessary at the client end to get the best available broker.

The remainder of this paper is organized as follows. In section 2, we present an overview of NaradaBrokering and cloud computing/virtualization technology. Section 3 proposes a monitoring management framework for NaradaBrokering and discusses its implementation (NBVirMon). Section 4 outlines experiment setups and incorporates experimental results. Finally, section 5 concludes the paper and discusses some of our future work.

II. Related Work

A. NaradaBrokering

NaradaBrokering is a distributed messaging infrastructure and provides a set of capabilities for reliable/robust flexiable messaging. The NaradaBroker network is hierarchical, cluster-based structure, which allows to support dynamic and large heterogeneous clients [9]. NaradaBrokering provides a notification/event based framework to efficiently route messages from publishers to only the registerd subscribers of the message. During the routing of messages within the brokers, every broker computes the shortest path to reach target destinations while skipping failed brokers and the associated targeted brokers are the only ones involved, provide faster data delivery [10]. Having many features supported, NaradaBrokering is popularily used as a messaging infrastruture.

B. Cloud Computing and Virtualization

In cloud computing, there are three typical types of cloud services offered such as Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). IaaS has been the most potential computational model for cloud computing. We can solve the problem of overloaded network services in the NaradaBrokering systems through IaaS, where the VM resources can be rented automatically.

III. Monitoring Management Framework

In this section, we propose a monitoring management framework for virtualized NaradaBroker systems and describe its design and implementation in more detail including overall architecture of the framework and each component's role.

A. System Framework

As depicted in Figure 1., Xen virtualization is installed on each physical node and multiple VMs are running on the virtual platform. To ease the virtual infrastructure management, we have used OpenNebula to deploy and manage multiple VMs. The OpenNebula daemon is installed on a management node, separated from the virtualized platform. There are two components in the system framework: VMAgent and MasterAgentServer. The first component, VMAgent running with a broker server is responsible for gathering the system usage information for each VM. Periodically, the VMAgent sends collected data to the second component, MasterAgentServer.

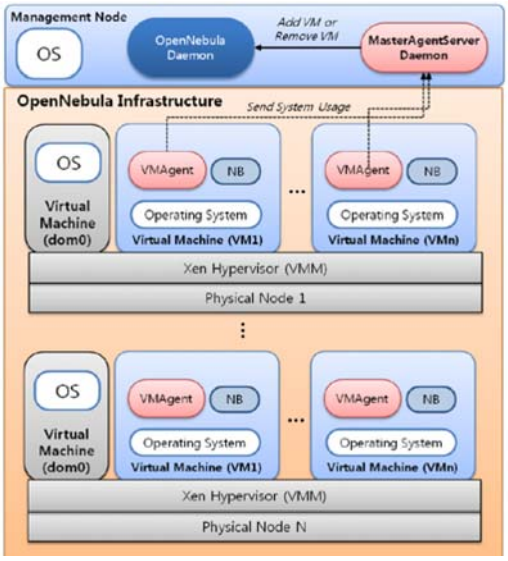


Figure 1. Monitoring system framework architecture.

The MasterAgentServer is installed on the same management node as OpenNebula daemon.

According to the usage information provided by VMAgents, the MasterAgentServer checks against the pre-defined threshold value for each resource (CPU, memory, network I/O) and decides whether to add/remove VM from the virtual infrastructure. Additionally, the MasterAgentServer computes overall usage given different weight values (priorities) assigned to each resource for NaradaBroker and returns the least loaded VM for any users to use for connections.

B. Implementation

We have implemented a prototype of the monitoring management framework for NaradaBroker, which is called NBVirMan (NaradaBroker VIRtual Manager). In this subsection, we will discuss its implementation briefly. We choose Xen hypervisor as the virtual platform because it is being developed extensively as a open-source and also choose Java as a primary programming language since it is used in NaradaBroker as well.

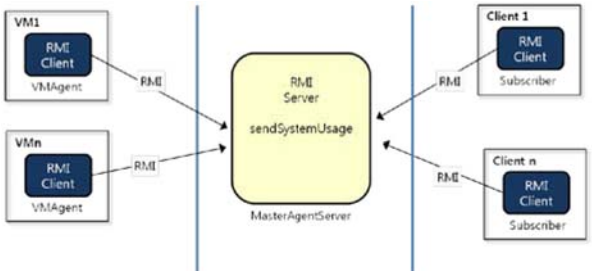


Figure 2. Communication between MasterAgentServer and VMAgent.

VMAgent: The VMAgent in each VM runs as a daemon. It is responsible for periodically sending system usage information of CPU, swap, memory, and network I/O for each VM to the MasterAgentServer daemon in the management node. To gather each system resource usage, /proc file system is used to collect CPU, swap and memory utilization, and transmitted network packages.

RMI (Remote Method Invocation) is used to transfer data between MasterAgent and VMAgent. Transparently, RMI allows a client (local) to invoke a method on a remote object which simplifies distributed application development. As shown in Figure 2 data are easily transfered to MasterAgentServer simply by calling remote method in RMI server.

MasterAgentServer: The MasterAgentServer runs as a daemon in the management node. It is responsible for analyzing system resource utilization from the data received from all VMAgents. And also it checks for any resources that may over or under the pre-defined threshold value. If any resource utilization exceeds the threshold, then MasterAgentServer issues a command to add a VM to the virtual platform. If the utilization is under the certain value for some period of time, then an unused broker VM is removed from the platform. Additionally, getBestAvailVM_Broker() method is implemented and can be used by any clients either publishers or subscribers, whoever want to connect to the best available broker.

OpenNebula configuration: To have the maximum scalability of broker network, it is important to place a VM in a separate physical machine. When setting up OpenNebula, we choose a striping policy for VM placement which allows to create a VM on those nodes with less VMs running first. This policy is best fit to our need, but it does not gurantee to have one VM per physical machine. To have strict policy like this, we may use the Haizea lease manager as a scheduling module in OpenNebula or implement our policy as a separate module. This work is left out for future work.

IV. EVALUATION

In this section, we discusss our results from the experiments performed on the OpenNebula infrastructure.

A. Test Environement

We have measured the performance and scalability in our framework. In our benchmarks we have used a setup as depicted in Figure 3. All machines from A to E are separate physical machines except a publisher executed within the domain 0 of the machine A. The publisher generates 64Kbytes of type string and uses TCP as the base transport protocol for communication within the system. Each subscriber machine (D and E) creates 50 threads incrementally at every given time interval (ex. 30 secs or five mins), up to a maximum of 500 connections.

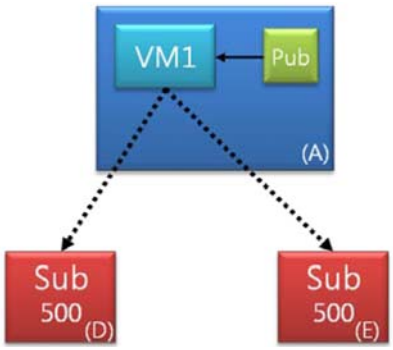


Figure 3. Experiment Setup.

We used the time-stamped value within the event header to measure performance. Each message issued by the publisher is time-stamped and upon receipt of the message at the subscriber after traversing through the broker networks, the transit delay is computed, taking the time-stamp difference between the last packet and the first packet received.

All machines involved in the experiments have Linux as the operating system and are connected over gigabit Ethernet. Each physical node A, B, and C machine runs Xen 3.3 and all VMs run the same OS and kernel as the host machine. The detail OS and hardware information are shown in Table 1.

B. Experiment Result

To measure the scalability of NaradaBrokering within the virtual infrastructure, we test it first by evenly dividing up the subscribe connections among VMs. For example, testing with one VM, each subscriber machine (D and E) would have exactly 250 connections. Testing with three VMs, each subscriber machine would have 33 connections to three different brokers. And then we compare the results with using our monitoring framework, NBVirMan. The results are shown in Figure 4. Here, we measure overall average transmission time as the number of subscriber thread increases. The result for "three VMs" is the most optimal case where all connections are divided evenly. For the case of NBVirMan, one VM is started initially and then additional VM is added (indicated as a bar in the graph) as the network load exceeds the threshold. The NBVirMan shows good overall scalability close to the optimal case.

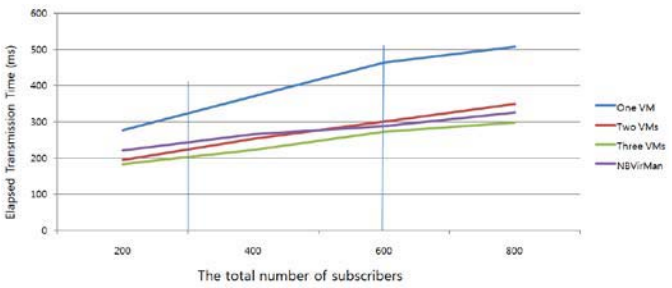


Figure 4. Overall average transmission time for different sizes of subscribers.

V. Conclusions

In this paper, we presented NBVirMan, a monitoring system framework for virtualized NaradaBrokering systems that automates the task of monitoring and determining VM allocation/deallocation based on pre-defined threshold values. Incorporating NaradaBrokering in the cloud platforms gives a lot of benefits towards Green IT by reducing power consumptions. In future work, we plan to test the VM placement policy with Haizea lease manager or just implement our own module in the OpenNebula.

Acknowledgment

This study was supported by the Seoul R&BD Program (10561), Seoul, Korea. The ICT at Seoul National University provides research facilities for this study.

References

[1] The NaradaBrokering Project at the Community Grids Lab, http://www.naradabrokering.org.

[2] Shrideep Pallickara and Geoffrey Fox, "On the Matching Of Events in Distributed Brokering Systems," in Proc. of IEEE ITCC Conference on Information Technology, vo.2, pp.68-76, 2004.

[3] G. Fox and S. Pallickara, "The Narada event brokering system: Overview and extensions," in Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02), pp.353–359, 2002.

[4] Shrideep Pallickara, "NaradaBrokering and its Applications - SC03 Handout", http://grids.ucs.indiana.edu/ptliupages/publications.

[5] Geoffrey Fox and Shrideep Pallickara, "An Event Service to Support Grid Computational Environments Journal of Concurrency and Computation: Practice & Experience," Special Issue on Grid Computing Environments, vo. 14(13-15), pp.1097-1129.

[6] G. Fox and S. Pallickara, "JMS compliance in the Narada event brokering system," in Proc. of the International Conference on Internet Computing (IC-02), vo.2, pp.391-397, 2002.

[7] Han H., Yeom H., Jung H., Lee Y., Jeong C., Kim S., Yu Y, "Management of remote facilities through a ubiquitous grid middleware," in Proc. of the 11th international conference on Advanced Communication Technology, vo. 3, 2009.

[8] Cloud Computing – Wikipedia, http://en.wikipedia.org/wiki/Cloud_computing.

[9] Shrideep Pallickara and Geoffrey Fox, "NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids," in Proc. of ACM/IFIP/USENIX International Middleware Conference Middleware, 2003.

[10] S. Pallickara, G. Fox, and H. Gadgil, "On the Creation & Discovery of Topics in Distributed Publish / Subscribe systems Topic Creation request Brief Overview of NaradaBrokering," in Proc. of the IEEE/ACM Grid, pp. 25-32, 2005.