Proceedings of the 5th WSEAS Int. Conf. on SIMULATION, MODELING AND OPTIMIZATION, Corfu, Greece, August 17-19, 2005 (pp7-12)

A Flexible Multi-layered Virtual Machine Design for Virtual Laboratories in Grid Systems

DOHAN KIM, ROBERT KENT, AKSHAI AGGARWAL, PAUL PRENEY
School of Computer Science
University of Windsor
401 Sunset Avenue, Windsor, Ontario, N9B 3P4
CANADA
http://www.hpc.uwindsor.ca/

Abstract: - We propose a flexible Multi-layered Virtual Machine (MVM) design intended to improve efficiencies in grid computing and to overcome known problems that exist in grid systems. We present a novel approach to building virtual laboratories by adapting MVMs within distributed and grid systems, thereby providing enhanced flexibility and reconfigurability. The MVM is a service-oriented grid architecture to discover, describe and retrieve platform independent configuration models for each resource usage pattern and provides virtual laboratory users with a logical view of the grid. It consists of three layers: OS-level, Queue-level, and Component-level VMs. MVMs virtualize system resources, topologies, networks, policies and services. In our framework, the virtual machines can be created "on-demand" and their applications can be distributed at the source-code level, compiled and instantiated in run-time.

Key-Words: - Multi-layered Virtual Machine, Grids, Service-Oriented Architecture

1 Introduction

In recent decades, virtual laboratories have provided an environment to test various research models, in the modelling phase, for cost and time savings [1]. A virtual laboratory can be dynamically organized, whereby its topology is adapted on-demand by research communities. Our goal is to create a virtual laboratory framework, using multi-layered virtual machines (MVM), in which virtual machines are scalable, reconfigurable, and flexible. These design goals are based on the following: *Reconfigurability* - Any group of virtual machines should be able to be reconfigured at run-time. *Flexibility* – Data, resource and network service components must interact seamlessly in a grid environment. *Dynamicity* - Virtual machines require the ability to be created and destroyed on-demand, regardless of network location while providing at least one simplified, logical view of any relevant underlying system. *Fault/Attack Isolation* – If a service suffers from faults or attacks, side effects should be minimized.

We assert that reconfigurable and layered abstract machines can simplify the complexities of current distributed and grid systems. By raising the level of abstraction for both resources and their networks, we can apply the logical resource usage models to a wide variety of heterogeneous environments.

The MVM model consists of three layers: OS-level, Queue-level, and Component-level VM layers. The OS-level layer is a virtualized operating system that can be dynamically deployed and moved around computer networks; the queue-level layer virtualizes data communication interfaces and the component-level layer provides services in an architecture-independent, transparent manner. These components specify the virtual topology, parallel communication patterns, and resource characteristics, and can be thought as Platform Independent Models (PIMs) of Model Driven Architecture (MDA) [2]. Further, component VM layer maps these PIMs into Platform Specific Models (PSMs).

This paper presents a virtual machine architecture for a virtual laboratory framework that meets the goals described above. Our multi-layered, flexible virtual machine architecture allows us to reconfigure the virtual machine itself at run-time rather than at compile- or deployment-time.

Section 2 presents the design of the Multi-layered Virtual Machine (MVM) and its three layers. Section 3 discusses the design of our framework by using MVMs including our virtual network model for grid computing. Section 4 presents some implementation and experimental results using MVMs. Finally section 5 presents concluding remarks and future work.

2 A Multi-layered Virtual Machine

2.1 OS-level VM layer

We take advantage of existing OS-level VM technology [3] to satisfy part of our design goals, especially fault/attack isolation. In our MVM design, distinct OS-level VMs can be multiplexed on a hardware abstraction layer called the Virtual Machine Monitor (VMM). In addition to the isolation feature of the OS-level VM technology, the OS-level VM can be used to set up a virtual network, which allows the setting up and testing of experimental services.

Among other matters, we intend to support scenarios in which multiple, independent jobs are running on the same physical machine, but simulating two virtual machines, each with its own processor, sharing and resource management rules.

One is then able to customize each OS-level VM with its own specific sharing, connectivity, jobs and resource management rules as if multiple, autonomous, dedicated host machines were running. In such cases OS-level VMs are allocated on appropriate VMMs.

Current projects, including VMPlant [4] and SODA [5], show that the on-demand, dynamic instantiation of VMs can be accessed through a Service Oriented Architecture (SOA). In a similar way, we can store a wide variety of OS-level VMs in the virtual backend, and then retrieve and assemble them with the Queue-level VMs by utilizing existing grid middleware methods.

The OS-level VM in the MVM can also facilitate the underlying system to be maintained in a partitioned way. This means that nearly all manipulations inside an OS-level VM do not affect the configuration values in another OS-level VM on the underlying system.

The principal advantages to the OS-level VM approach is that of virtualizing user accounts, monitoring facilities, logging and system services.

Yet another advantage of utilizing the OS-level VM as a building block of the MVM is that we can migrate the processes in the OS-level VM including the OS-level VM itself to other locations across a network, grid, or off-line storage. The strength of OS-level VM based checkpointing is that all volatile execution states of running processes (including disks, memory, CPU registers, I/O devices, etc) can be encapsulated [6]. Indeed, the entire running MVM, including queue-level and component-level VMs, and all of their respective applications, can be encapsulated.

2.2 Queue-level VM layer

The queue-level VM virtualizes data communication interfaces by using virtual queues. We advocate that "enqueuing" and "dequeuing" are the most common characteristics of any computer systems available. Any computer system should enqueue and dequeue its data including instructions. The actual computation is determined by how a given machine decodes items from the queue, and how it encodes its output. Our design of the queue-level VM in the MVM is intended to be fully generic and flexible, allowing us to (re)configure and control how to communicate with an arbitrary computer system. Once this is configured at run-time, one can build a flexible and scalable system on top of it.

The queue-level VM consists of VM interface logics, VM interface controller, and VM encoder/decoder. The VM interface logics include the virtual queue types (i.e. FIFO queue, priority queue, etc) and policies for queue management. They are specified by service-oriented components which enable virtual machines to register and retrieve information on how to encode, decode, and process those items.

The "enqueuing" and "dequeuing" portions of the queue-level VM can be extended to a group of nodes. When a group of queue-level VMs are organized under certain sharing rules for the specific experiment, it provides a Single System Image (SSI) to an external user. Several virtual queues can be allocated for the enqueuing process in the queue-level VM, so that we can assign different queues to different jobs, allowing each job to be run concurrently in each queue in SSI. Each connected VM can provide either instruction stream, or data stream, for the parallel computation. Thus, we can simulate Flynn's MIMD architecture [7], by organizing a group of queue-level VMs of MVMs.

While we may not know exactly how to interact with data communication interfaces for an arbitrary computing system, with MVM one is able to look at the virtualized queues, extract items, and then apply various rules to these items.

2.3 Component-level VM layer

Our VM middleware shares certain problems, including resource discovery, resource allocation, resource management, authorization/authentication, policy enforcement, as with the existing grid middleware. As a consequence of virtualization, we can dynamically reconfigure and customize all VM components using service-oriented technologies such as web services.

Another advantage of the component-based technology is reusability. Once we register the connectivity, resource, and job profiles for a certain experiment, we might reuse the same profiles, with different parameters later and, possibly, in different job scenarios and topologies.

3 A Proposed Virtual Laboratory

The proposed virtual laboratory has three different layers to serve as the basis of a dynamic, flexible test environment for a wide range of research scenarios including: virtualized resources, virtualized networks, and policy-based reconfigurable components. For scalability, the virtual front-ends and the virtual clusters in virtualized resources are organized in a decentralized way. However, a pure decentralized system has several drawbacks, such as bandwidth overuse by message flooding, and maintenance difficulties [8]. Thus, we advocate using a super-peer based P2P system for aggregating virtual resources. Each virtual community has one or more peer groups, and each peer group has a super-peer. The super-peer in each peer group acts as the virtual front-end, in that it provides a group of peers with virtual back-end information. This data includes URI location of the virtual backend, and the ways on how to retrieve information from the virtual back-end.

3.1 Virtualized Resources

The virtualized resource layer consists of virtual back-ends, virtual front-ends and virtual clusters. The virtual back-ends consist of several remote servers that provide the virtual front-ends and virtual clusters with our VM image, resource discovery, bootstrapping, storage service, and so on. Servers in the virtual back-end can be one of these types: high performance computers; workstation clusters; or data stores. The main building blocks of the back-end servers are bootstrap nodes, information service servers, image servers, and authentication and authorization servers. Service-oriented grid architectures, such as OGSA, virtualize back-end servers as “services”. We also use service-oriented grid architecture for platform independent configuration models for each resource usage scenario. The virtual front-end is the super-peer module of our virtual laboratory. We use the term “virtual front-end” rather than “front-end”, in that the virtual front-end node is itself a MVM node with additional flexibility and reconfigurability.

The primary purpose of the virtual front-end is to schedule the incoming jobs to the available resources in the virtual cluster by using the queue-level VM. We note that scheduling problems in grid computing are known to be NP-complete problems [9].

Using the component-level VM, we can configure scheduling policies to our schedulers in the virtual front-end nodes, giving flexibility to the local and global schedulers. For a specific experiment in the virtual community, the virtual front-end controls the connectivity and sharing rules for the virtual cluster. The connectivity and sharing rules consist of the service-oriented components, and are manipulated by the component-level VM in the MVM. Thus, we are able to map the logical connectivity (virtual topology) and sharing rules to the physical connectivity and sharing rules for the virtual cluster.

The virtual cluster in our virtual laboratory is organized as a group of MVMs, dynamically and on-demand. The virtual front-end node initially retrieves the available resource information of the virtual cluster from the information service server. Then the virtual front-end node notifies a group of available nodes in the community and organizes the virtual cluster.

3.2 Virtualized Networks

Certain kinds of grid applications require virtual topologies to specify the logical arrangement of tasks. In some situations we are required to specify each stage in the topology for high performance pipelined computing, such as the butterfly computation of the Fast Fourier Transform (FFT) [10]. However, in current grid applications, we need to denote the virtual topology at the source-code level.

We emphasize that there is no requirement for the source/destination field for the message passing API in the MVM toolkit; instead, we specify the virtual topology and network information outside, rather than inside the programming source codes. All connectivity information is determined outside the grid program, enabling efficiency and runtime-reconfigurability for job execution. Additionally, this scheme allows a grid resource scheduler or allocator to select and map process-to-processor in grid environments in dynamic and adaptive ways, as the message passing API for MVM is not bound to specific source or destination job id at the source-code level. Further, we provide job distribution at the source-code level, allowing us to compile and instantiate jobs at runtime rather than at deployment time. The virtual topology for grid job execution is one of our platform independent configuration models which is a grid service.

3.3 Policy-based Reconfigurable Components

According to Appleby [11], "policy-based computing" is "a software paradigm that incorporates a set of decision-making technologies into its management components in order to simplify and automate the administration of computer systems". The policy components in the MVM are designed to achieve this viewpoint, allowing dynamic adjustment of the behaviour of the group of MVMs in run-time, without modifying its internal implementation. The policies in our virtual laboratory framework are deployed and reconfigured at different levels of abstraction. We divide the policies into three levels: inter-domain level, domain (virtual community) level, and general node level. The inter-domain level policies may require a mediator to manage semantic heterogeneity and integration of multiple heterogeneous policies for each domain. We mandate that inter-domain resource sharing in the virtual laboratory be subject to the inter-domain policies.

The domain level policies apply to the virtual community to specify the security rules, resource sharing rules, privileges for each participant, fault recovery mechanisms, scheduling and monitoring mechanisms, and so forth. The domain-level policies include how and when the partitioning takes place, and how to monitor each partition. For both inter-domain and domain level policies, the Policy Management Point (PMP), Policy Decision Point (PDP), and policy repository should be located in the virtual back-ends. Our "policy-based reconfigurable components" are located in the Policy Enforcement Point (PEP); that is, the broker module of the highest level super peer in the hierarchical tree. Once the highest level super peer in the virtual community enforces these policies, the lower level super peers in the hierarchical tree retrieve them from the highest level super peer if needed.

The general node level policies determine the interface logics, such as encoding/decoding types, queue type, maximum number of queue-level VMs, and so forth. They also specify how to monitor the performance and availability of each node, and what metrics are used for them. The general node level policies do not require external PDP and PMP, thereby allowing the user to set his/her policies for MVM.

The policy-based reconfigurable components consist of three main building blocks: security handlings, fault tolerance, scheduling and monitoring. Full implementation of these components is not yet complete, but our work is based on the following considerations:

The security handlings deal with both authentication and authorization. The authentication policy specifies what kind of authentication mechanism is used for a certain virtual community. For scalability, we advocate the PKI-based GSI authentication model [12], which supports "single sign on" and "delegation" capabilities.

The authorization policy determines whether a certain user is allowed to do an action by using a certain amount of resources. The Role-based Access Control (RBAC) [13] is emerging as an authorization mechanism for large-scale systems in which both policies and user roles are stored in attribute certificates to provide integrity. It is based on user-to-role and role-to-permission assignments. We note that context and content-based constraints for the extended RBAC Model are discussed in [14].

The monitoring reconfigurable policy determines how often the resources in a virtual community are monitored, and what metrics are used to monitor them. Each monitoring result is reported to the super-peer which determines what policy components should be replaced to optimize the virtual cluster. The scheduling reconfiguration policy determines what policy will be applied for resource scheduling in a virtual community. The optimal scheduling policy depends on each monitoring result, allowing the scheduling policy to be dynamically adjusted for system status in the virtual community. The scheduling policy is also affected by the resource reservation policy. The resource reservation policy determines whether the reservation is allowed, how the reservation is made, and what requirements need to be satisfied for resource reservation.

The fault tolerance reconfigurable components consist of MVM checkpointing and MVM failure recovery policy components. The MVM checkpointing policy determines what kind of checkpointing method is used for a virtual community and how often the checkpointing is to be conducted.

We envision that the automatic runtime policy reconfiguration and enforcement allows our virtual machines to be self-configurable in order to optimize themselves in grid systems.

4 Implementation and Results

We have developed the MVM toolkit to evaluate the essential functionality of the MVM framework. This toolkit is a testing tool for MVM framework in distributed systems and grids and it is still work in progress. We have used the Apache web server

2.0.52, gSOAP 2.7.0 [15], Globus 3.2.1[16], POSIX IPC, POSIX Threads, and BSD Sockets, running under Fedora Core 3 Linux systems (kernel 2.6.9) for our implementation and various experiments. The current features of MVM toolkit are as follows:

(1) *Source-code level grid job distribution*: Traditional computing architectures, such as MPI and PVM, need to recompile jobs for every participating node at deployment time. In addition, one must connect each machine, update each program, and recompile or reconfigure them for each job scenario. Within the MVM framework, users modify job profiles and MVM automatically reconfigures, deploys and runs the system.

(2) *Component-based grid job (re)configuration*: In the MVM framework, each user sends a job profile to a resource broker to request a unique runtime environment for each resource usage case. The MVM toolkit takes the job profile data structure and renders it as a SOAP document intended to provide resource sharing over grid environments using standard web services.

(3) *Virtual network approach for grid job execution*: Each user can create a virtual network at runtime, specifying the virtual topology for grid jobs. All connectivity information is determined outside the grid application, allowing reuse or redeployment of grid communication patterns.

(4) *P2P Web services and P2P socket approach*: Each node has both server and client modules for socket and web services. This means that each node publishes its service using WSDL and accesses other nodes using a SOAP interface. Additionally, similar transactions can happen through traditional sockets.

(5) *On-demand creation and termination*: The MVM processes do not have to run continuously. Whenever a node is invoked from other nodes, it can initialize itself and launch tasks for a particular use.

In more detail, the "on-demand" creation and termination mechanism for the MVM is as follows:

a. Only the server runs for each node and the MVM process is not loaded yet.
b. After contacting a bootstrap node, the MVM client retrieves the broker address and invokes the components of the broker node by using SOAP. The MVM client sends a job profile data structure to the broker node at this phase.
c. The MVM process is loaded by the SOAP invocation, and the process instantiates the MVM proxy and the MVM queue threads, if required.
d. The broker node selects a job group and awakens all the nodes in that job group. Each node in a job group instantiates its proxy and queue-level VM module if required.
e. Each node communicates with other nodes by using its proxy with BSD sockets.
f. The broker node generates the job instantiation messages for each participating node, and sends these messages to each participating node in a concurrent way. According to the job instantiation messages, queue-level VM spawns child processes and initializes the inter-process communication (IPC) subsystem. Processes in a local host enqueue or dequeue their data and instructions via IPC, and processes between different hosts enqueue or dequeue their data via their proxies.
g. When a job has finished its operation, it reports to its resource broker. The resource broker then broadcasts a job termination message to a job group. A proxy module for each job group reads the message, and sends a terminate signal to all on-demand created processes.

Using various services within Globus, we have tested the MVM framework by writing codes that are semantically equivalent to those of a series of programs developed under Globus using MPICH2 [17]. The MVM framework properly, dynamically, creates the proper topologies to run the various job instances. Although the job run-time performances are lacking, relative to MPICH2, this is due to the fact that the current MVM model is not performance optimized at the API level. Further, it is important to note that, in contrast to MPI based programming, all topology considerations are dealt with exclusive to the program and within MVM itself.

5 Conclusion and Future Work

In this paper we have presented an architecture for a Multi-layered Virtual Machine design intended to provide a foundational mechanism for supporting virtual laboratory infrastructure.

Through the use of virtualization we have reduced, if not eliminated, the costs associated with topology configuration and the deployment of codes on a grid. We described how virtualization technology can provide runtime flexibility and automation for grids. By specifying virtual topology and network information outside, rather than inside, source codes, we may reuse grid communication patterns by reusing virtual topologies.

In our framework, multi-layered virtual machines can also be created and destroyed "on-demand". This capability provides further flexibility and automation in grid environments. It lessens the need for manual effort and intervention in configuring and deploying jobs, thereby reducing management and maintenance costs.

We have not yet implemented all features of our system. We envision enhancing our "reconfigurability" mechanism as a policy-based web service. In the spirit of autonomic computing, automatic runtime policy reconfiguration and enforcement will permit our virtual machines to be self-configuring to some degree.

The implementation and testing of interface modules for existing OS-level VMs has to be extended to complete our framework. Specifically, in the area of security, we aim to implement a grid-based trust system and protection against malicious codes and attacks.

We are also planning to extend our virtual data communication interface model to optical networks and wireless networks. Hence, we plan to incorporate User-Controlled Light Path (UCLP) [18] services into the MVM framework by using our component-level VM layer services.

References:

[1] P.D. Preney, R.D. Kent, "Toward a Model of Models. Part 1", *High Performance Computing Systems and Applications, Andrew Pollard et al, eds, Kluwer Academic Publisher, pp:33-38,* 2000.

[2] Object Management Group, "OMG Technology Explained", *webpage: http://www.omg.org,* 2001.

[3] J. Sugerman, G. Venkitachalam, B. H. Lim, "Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor", *USENIX Annual Technical Conference, pp: 1-14, USENIX Association,* 2001.

[4] I. Krsul, A. Ganguly, J.Zhang, J. A. B. Fortes, R. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing", *SC 2004, webpage: http://www.sc-conference.org/sc2004/schedule/ pdfs/pap305.pdf,* 2004.

[5] X. Jiang, D. Xu, "SODA: a Service-On-Demand Architecture for Application Service Hosting Utility Platforms", *IEEE International Symposium on High Performance Distributed Computing (HPDC-12), pp:174-183,* 2003.

[6] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, M. Rosenblum, "Optimizing the Migration of Virtual Computers", *ACM SIGOPS Operating Systems Review, vol. 36, Issue: SI, pp: 377-390,* 2002.

[7] M. Flynn, "Very High-Speed Computing Systems" , *Proc. of the IEEE, vol. 54, Issue:12, pp:1901-1909,* 1966.

[8] N. Daswani, H. Garcia-Molina, "Query-Flood DoS Attacks in Gnutella", *ACM CCS, pp: 181 - 192,* 2002.

[9] J.L. Träff, "Implementing the MPI Process Topology Mechanism", *Proc. of the 2002 ACM/IEEE conference on Supercomputing, pp: 1-14,* 2002.

[10] Kent, R.D., Majmudar, N., Schlesinger, M., "Distributing Fast Fourier Transform Algorithms for Grid Computing", *High Performance Computing Systems and Applications, Kluwer Academic Publishers, Nikitas J. Dimopoulos (eds.), pp:407-426,* 2001.

[11] K. Appleby, S.B.Calo, J.R. Giles, K.W. Lee, "Policy-based automated provisioning", *IBM Systems Journal, vol. 43, num. 1, utility computing, webpage: http://www.research.ibm.com/journal/sj/431/appleby.html,* 2004.

[12] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications, vol.15, Issue:3, pp:200-222,* 2001.

[13] W. Zhau, C. Meinel, "Implementing role based access control with attribute certificates", *Proc. Of the 6th International Conference on Advanced Communication Technology(ICACT 2004), vol. 1, pp:536-541,* 2004.

[14] Joshi, J.B.D. Bhatti, R. Bertino, E. Ghafoor, A., "Access-Control Language for Multidomain Environments", *Internet Computing, IEEE, vol.8, Issue:6, pp: 40-50,* 2004.

[15] gSOAP 2.7.0, webpage: *http://www.cs.fsu.edu /~engelen/soap.html.*

[16] Globus 3.2.1, webpage: *http://www-unix.globus.org/toolkit/downloads/3.2.1/*

[17] MPICH2 v.1.0.1, webpage: *http://www-unix.mcs.anl.gov/mpi/mpich2/*

[18] UCLP: User-Controlled LightPaths, webpages: *http://www.canarie.ca/canet4/uclp/*