Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Design and Implementation of Advanced Node-level VO Support Mechanisms
D2.1.5

Due date of deliverable: Novmember 30th, 2008
Actual submission date: December 17th ,2008

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP2.1
Task number: T2.1.7

Responsible institution: ICT
Editor & and editor's address: Haiyan Yu
Institute of Computing Technology
No.6 Ke Xue Yuan Nan Lu
100080 Beijing
China

Version 0.3 / Last edited by ICT Team / Dec 17th, 2008

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.1	15/11/08	Haiyan Yu	ICT	Initial draft
0.2	25/11/08	Haiyan Yu	ICT	Revision based on reviewers' comments
0.3	17/12/08	Haiyan Yu	ICT	Final version

Reviewers:
Adolf Hohl(SAP), Matej Artac(XLAB)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.1.7	Design and Implementation of Advanced version of node-level VO support mechanisms	ICT*, INRIA, TID, STFC, CNR, SAP

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Contents

Executive Summary

In the basic version of VO support mechanisms, the isolation of VO users' access to the same local node is done by mapping VO users into different local accounts. This approach exposes several limitations due to the fact that multiple VO users' jobs are running in a shared local OS instance. For example, one VO user's job could exhaust system resources resulting in the crash of the whole system. Also, it is quite difficult to support jobs that require different execution environments in terms of a specific version of kernel, libraries, and system configurations.

The main objectives of advanced node level VO support mechanisms are i) strong isolation of resource usage and enforcement of fine-grained access control for VOs, in terms of separated security, performance and other QoS constraints among multiple VO users accessing the same physical node. ii) auditing, logging, accounting mechanisms that are needed at node level to provide the information needed by the higher-level VO management and security services.

In this report, we study and try to leverage new standard-based OS extensions supported by the mainline Linux kernel and mainstream Linux distributions, mainly virtualization techniques including full virtualization, para-virtualization and light-weight OS level resource containers, to create isolated execution environments for VO users. Typically these enhancements are done in an incremental manner so as to preserve the semantics and the API provided to other XtreemOS packages as well as normal Linux applications, ensuring backward compatibility.

1 Introduction

The main responsibilities of node-level VO support include authenticating grid users and verification of their security tokens, mapping from grid user identities to local user identities, translating VO-level access control policies into local OS-level access rights and capabilities, tracking and auditing users' accesses to local resources, accounting resource usage, isolation of different users' access to the same node, as well as session management of users' activities.

We leverage OS-level extension mechanisms to implement VO support, which are based on standard (or de-facto standard) interfaces supported by most Linux distributions. With conforming to POSIX-compliant or generic common library APIs, this approach could enable most traditional Linux services and applications to be VO-aware without a heavy change of their source codes.

However, without operating system support, it is difficult to differentiate access control and auditing among different grid users accessing the same node, in a scalable and flexible manner, as well as preventing one user's applications from interfering with the other's in terms of performance and other QoS constraints. OS-level lightweight isolation and enforcement mechanisms, which are already in or approaching the mainline kernels, are investigated and evaluated to be used for better support of VOs. These mechanisms include virtualization mechanisms such as para-virtualization and process containers that evolved from earlier resource containers [7] and security containers [22]. By such way, enforcement of VO policies at a fine-grained level, as well as efficient and guaranteed isolation of VO accesses, could be achieved in the local node. Typically these enhancements are done in an incremental manner so as to preserve the semantics and the API provided to normal Linux applications, ensuring backward compatibility.

The rest of this report is organized as follows: Section 2 presents current status of basic version of VO support and discusses its limitations. Section 3 gives overall objectives and design principles of using virtualization techniques in XtreemOS. Section 4 summarizes state-of-art virtualization techniques and their pros and cons. Section 5 presents the design and implementation of advanced VO support modules based on *libvirt* API library [2] and *cgroup* support [3] in latest kernels. Section 6 concludes this report.

2 Revisiting the Basic Version of Node-level VO Support

D2.1.2 [12] presents the basic version of node-level VO support mechanisms. New PAM and NSS extensions are developed to take over the control of authentication procedure when VO users access a node either by submitting jobs or using a login shell. VO users are dynamically mapped onto local user accounts once a PAM session starts. A dedicated Account Mapping Service (AMS) was developed to serve as the back-end for PAM and NSS extensions, which performs the actual mapping actions based on local configured mapping rules. AMS also acts as the local policy engine for enforcing VO policies (in the basic version, VO policy of access control is quite simple as whether to allow or deny a user's access). The integration of basic version of VO support extensions (i.e. `nss-pam`) with other XtreemOS work packages is described as follows:

- **Integration with XtreemFS** XtreemFS provides two APIs (`xtfs_mount` and `xtfs_umount`) for mounting the home volume of a VO user in the (global) XtreemFS file system. The mounting and unmounting operations are put into the start and end of a PAM session, respectively, i.e., the `xtfs_mount` is called in `pam_open_session` and `xtfs_umount` being called in `pam_close_session`. The PAM module makes the assumption that a VO user has already created her own home volume in XtreemFS before she logins into the node. Correspondingly, a local home directory is created as the default working directory of a VO user.

- **Integration with AEM** A PAM-aware interface library was provided to AEM which relies on the underlying PAM module to launch a job on behalf of the submitting VO user's identity. This library is called by the local node execution manager of AEM. Like XOS-SSH, AEM becomes another PAM-aware application which is able to track all spawned processes of a job after the user identity mapping is done in PAM module.

- **Integration with VOM** The integration with higher level VO Management(VOM) services is done in an *offline* mode due to the nature of PKI-based security infrastructure. Currently the user identity together with other VO attributes (e.g. VO names and groups which the user belongs to) are conveyed in the XOS certificate of a VO user. PAM module processes XOS certificate to get all necessary information to do account mapping.

- **Integration with checkpoint/restart mechanisms** The checkpoint/restart mechanisms require that uid/gid of mapped local users are statically allocated, i.e., after the job restart procedure, the mapped user has the same uid/gid as the one before job checkpointing. This consistency is ensured by PAM module with storing the mapping information in a local repository.

In current implementation of node-level VO support modules (`nss-pam`), isolation of VO users' access to a node is done based on mapping VO users onto different local users. This approach has a few limitations listed below:

- **Performance Issues** Without kernel-level support, there is no good way to effectively control resource usage in a shared environment. One user can consume a large portion of CPU, memory or disk space resulting in fewer resources being available for other users. The `ulimit` could be used to control amounts of consumed resources by a user. For example, the limitation of maximum CPU wall time against a user could be set with `ulimit`, but apparently controlling the percentage of CPU usage is more meaningful. It is possible to use `setpriority` (i.e. `nice`) to change the scheduling priority of a process, process group, or user. To control the percentage of CPU usage by a user, additional monitoring works need to be done against all job processes of the user, to provide an estimation of consumed total CPU time and thus dynamically adjust scheduling priorities of those processes. However, this kind of adjustment is not accurate and becomes very complicated when there are numerous running processes.

- **Stability and Security Issues** Since many local users are sharing the operating system of a node, a buggy or malicious program with one user's jobs could cause a system-wide crash, which brings down other users' jobs on the same node. Also, the local OS's vulnerability is increased as VO users could submit arbitrary jobs that may lead them to be compromised.

- **Deployment Issues** Each local user shares system-wide components with others, such as the same version of kernel, libraries, and configuration files. This severely limits the ability to configure and control the job running environment at the individual VO user level. In other words, all VO users' jobs need to be compatible with the local OS environment and even hardware architecture. This also limits the selection of resource nodes for running jobs with different requirements of environments such as a specific version of kernel, library, and system configuration files.

3 Objectives and Design Principles

The limitations with the basic version of VO support stems from traditional account mapping mechanism where system resources are coarsely protected by user accounts. The objectives of advanced VO support are strong isolation of resource usage and enforcement of VO policies in terms of security, performance and other QoS constraints. Access rights of user applications on end nodes as well as their resource quotas (memory, disk, cpu, net, etc.) defined by VO and/or resource policies must be strictly controlled. Effective resource usage must be accounted and transmitted in real time to higher level services. Isolation of grid applications must be enforced at various levels (performance, namespace, data, etc.) depending on the user requirements.

The key point to advanced VO support is to enable multiple isolated execution environments within a single OS instance. According to Solaris 10 [19], isolation is defined in three aspects: ***security isolation***, ***resource isolation*** and ***fault isolation***.

- **Security isolation** Isolated execution environments for VO users, also called containers or sandboxes, are shielded from the outside world and the running processes of a container are assured that no other users of a container on the same node can see what they are doing, or compromise information. Additionally, an administrator inside of a container (i.e. a VO user granted with admin access rights) only has authority over her own container, so if the container is illegally accessed, the container isolates the intruder inside the boundary.

- **Resource isolation** Each container is assigned a specified allocation of CPU, memory, and disk, according to VO policies set on a VO user. It is possible to allow a VO user's jobs to have bursting resource usage within reasonable limits but not to dominate the physical server's resources. And such, each container could receive a guaranteed level of service.

- **Fault isolation** A fault or a process in one container does not adversely affect processes running in other containers. The container is also independent from the physical hardware, making it possible to move (i.e. checkpoint/restart) a container from one physical node to another with no downtime. The ability to dynamically reconfigure these containers offers increased flexibility in operations and optimized resource utilization.

With virtualization techniques (an evaluation technical report is presented in D2.1.6 [13]), it is possible to create such isolated execution environments (i.e. *Virtual Machines*, *domains*, *guest hosts*, or *containers* in virtualization terminology) that performs as if it were a full node dedicated to the exclusive use of each VO user. To avoid ambiguity, we use the term *domain* to represent the isolated execution environment in following sections.

Though virtualization is a promising way to facilitate the isolation of VO access of a physical node and strict enforcement of resource usage. A few design principles need to be followed in XtreemOS case :

- **Application Transparency** Grid applications running on VOs do not need to be aware of the existence of specific virtualization capabilities of nodes. In other words, application codes do not need to change with nodes that may or may not support virtualization. VO users do not know their submitted jobs are possibly running in *sandboxes* of a machine. It is possible for VO users to submit domain images (a domain image is a packed file providing metadata of resource configurations of a VM and the pre-installed OS image) to a node for running provided that they are granted this kind of access right. In that case, domain images could be stored on XtreemFS and the instantiate of images could be specified in a JSDL file.

- **Scalability** Depending on VO policies, it is possible to allocate one domain on per-VO basis, or one domain for each VO user, or even one domain for each running job. In this document, we only consider the allocation of domains on per-user basis, i.e., all jobs of a VO user are limited to running within one domain. When there are a large amount of VO users accessing the same node, it requires that underlying virtualization mechanisms are light-weight in terms of both running and management overhead. As discussed later, the choice of virtualization solutions is quite difficult as there is no best one among available techniques.
- **Flexibility** With the built-in virtualization support from CPU chips (e.g. Intel Intel VT-x or AMD-V Pacifica and Vanderpool), virtualization techniques are becoming mature and get a wide adoption in commercial IT infrastructures that require improved cost-effectiveness and manageability. It is also hard to predict which technical virtualization solution will dominate in the future OS releases. In XtreemOS case, it is desirable that VO support mechanisms are independent of a specific virtualization solution.

4 Revisiting Virtualization Techniques

Recent years have witnessed the mature and wide application of virtualization techniques. Xen [20], VMWare [6] and Virtuozzo [5] (the commercialized version of OpenVZ [4]) are commonly seen in Web hosting business for providing dedicated Virtual Private Server (VPS) renting. Linux-VServer [15] is extensively used in PlanetLab [11], a worldwide platform provide virtualized "slices" to researchers. Recently Xen based virtualization platform got the focus in cloud computing such as Amazon EC2 platform [1] and related research plans [10] proposed by IBM/Google.

In XtreemOS, before we leverage virtualization techniques, the following questions are to be answered:

- For strong isolation and enforcement of resource usage, which kind of virtualization technique should XtreemOS choose ?

- What kind of changes could happen with other XtreemOS modules if we introduces virtualization in VO support ?

4.1 Comparison of VMM features

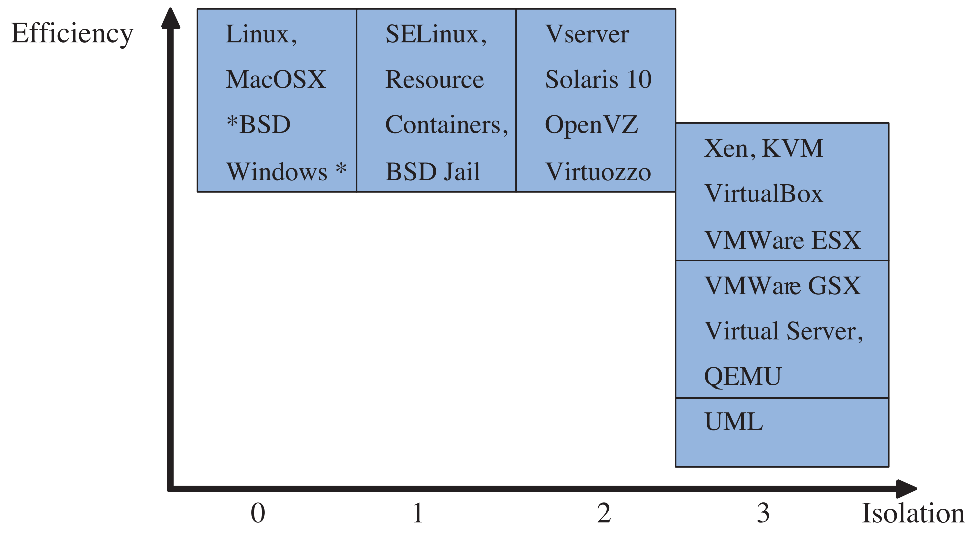

Figure 1: Comparison of existing VMM techniques

Figure 1 and Table 1 are adapted from [23]. Figure 1 summarizes the state-of-the-art in Virtual Machine Monitor (VMM) technology along two dimensions. The x-axis counts how many of the three different kinds of isolation are supported by a particular technology. The y-axis is intended to be interpreted qualitatively rather than quantitatively. Table 1 provides a list of popular features that attract users to VM technologies. The key observation is, to date, there is no VMM technology that achieves the ideal of maximizing both efficiency and isolation, as well as supporting all desired features. A detailed comparison of virtualization techniques are discussed in D2.1.6 [13].

4.2 Comparison of VMM Performance

Using virtualization for high performance computing (HPC) applications is currently limited despite its potential for both improving resource utilization as well as providing resource guarantees to its users. In [14] authors systematically evaluate various VMs for computationally intensive HPC applications using various standard benchmarks. VMWare Server, Xen and OpenVZ are examined respectively to test the suitability of full virtualization, para-virtualization, and operat-

Features	**HyperVisor-based approaches (Xen, KVM, VMWare ESX,VirtualBox,UML etc.)**	**Container-based approaches (OpenVZ, Linux Vserver, Linux Container, etc.)**
Fault Isolation	yes	no
Resource Isolation	yes	yes
Security Isolation	yes	yes
Multiple Kernels	yes	no
Separated root access	yes	yes
Checkpoint & Resume	yes	yes
Live Migration	yes	yes
Live System Update	yes	yes

Table 1: Comparison of common features of VMM

ing system-level virtualization in terms of network utilization, SMP performance, file system performance, and MPI scalability. The conclusion is that operating system-level virtualization provided by OpenVZ provides the best overall performance, particularly for MPI scalability. Similar results are found in [16, 17, 24]. Though there is no evaluation report regarding control group (`cgroup`) support [3] in latest kernel, we believe that it gains better performance than full virtualization solutions, provided that jobs are running with the same version of kernel.

4.3 Impacts on other XtreemOS modules

For advanced VO support by leveraging virtualization techniques, multiple VO accesses to the local node are isolated by domains, be it heavy-weight virtual machines or light-weight containers. While we try to minimize the impact on other XtreemOS modules, the following issues have to be considered:

- **XtreemFS integration** Generally the sharing mechanisms among domains (VMs) is very limited to only network based communication. Some VMMs support direct mounting of file systems between hosts and guests. For applications running in domains to access XtreemFS, either the domain image (OS) is capable of running XtreemFS client or the domain relies on the underlying host to expose XtreemFS into its file system space. For the former approach, it requires that domain images is also equipped XtreemOS or at least the XtreemFS client. In the later case, additional mounting work needs to be done, and the implementation approach depends on specific VMM support or in the worst case using NFS based file sharing.

- **AEM integration** The local execution manager and monitoring service of AEM need a special treat of the case that jobs of a VO user are running in a domain. In other words, jobs running in domains need to be tracked in a different way and thus additional work is required to interact with domains. For example, the local execution manager of AEM needs to be able to manager processes in a domain. This can be accomplished by connecting domains and issuing process related command via remote shell or deploying AEM services into domains. Things become complicated if VO users are permitted to created new domain images or uploading existing images to nodes. In such case, AEM needs to deal with additional management works such as finding matched nodes with virtualization capabilities, transferring domain images or fetching images from XtreemFS to nodes. Finding virtualization-capable nodes would be a matter of using proper/extended JSDL and publishing the virtualization capabilites by the node as another resource. Of course we would need to settle on what type of attributes to use, and whether a domain is advertised as a node, or should we stick to advertising the host only. It is true, however, that AEM would need to set up the user's domain before starting a job, and tearing it down afterwards. It sounds likely that AEM too would need to be installed on the domain, unless we could count the running of the domain as a job, and simply control the domain as a black box.

- **VOM integration** The virtualization support raise new questions for VO policy management services. There are several choices regarding the granularity of controlling resource usage in VOs. Possible policies could be allocating one domain per VO (i.e. enforcing QoS constraints at whole VO level), one domain per VO user or one domain per job. There could be new policy of allowing VO users to submit their own OS images to resource nodes. How to define these policies and how to convey them to the node is a remaining issue.

5 Design and Implementation

5.1 Architecture

As depicted in Fig. 3, a new module named Domain Management Service (DMS) is developed to control all domain related operations in a local node, which accepts given resource constraints for VO users (e.g. specified in VO policies) and creates domains for each VO user. DMS also performs the real-time monitoring of domains to ensure the resource quota is not exceeded. The resource usage enforcement information to DMS is passed by PAM-aware applications, e.g., a job execution manager or a login shell.

With considering the fact that different virtualization support could reside on physical nodes (e.g. Mandriva with Xen, Debian etch with OpenVZ, etc.) and for simplicity of implementation, *libvirt* API library [2] is used to implement DMS. Libvirt provides a stable API for managing virtualization hosts and their guests. It started with a Xen driver, and over time has evolved to add support for QEMU, KVM, OpenVZ and most recently of all a driver called "LXC" short for "LinuX Containers". With a consistent set of APIs, libvirt also adopts standardized configuration format for user-space management applications in the host (and remote secure RPC to the host).

Sub components of DMS are listed as follows:

Domain Controller Domain controller is responsible for creating, destroying, suspending/resuming of domains for VO users. Domain controller interacts the underlying hypervisor for managing domains. By default, there is a hypervisor implemented via `cgroup` extension (discussed later). Domains are persisted into domain repository as domain images when necessary.

Domain Monitor Domains are created with determined resource constraints including CPU percentage, memory quota, storage quota and network bandwidth limitation. Domain monitor keeps an eye on running domain instances using hypervisor dependant APIs (e.g. XenMon [18]). Domain monitor is able to notify domain controller when domains are exceeding their resource usage limits.

Domain Repository The domain repository is used to store templates of domain configurations (e.g. hardware resource configuration defined by `virDomainDefineXML`) and domain OS images/snapshots. For example, when a VO user leaves a node temporarily, the allocated domain image could be suspended into domain repository to save system resources. When a VO user access the node next time, her domain is quickly put into running status (e.g. via `virDomainCreate`).

5.2 Default DMS Implementation on cgroup support

By default, we use a light-weight OS level virtualization method as the hypervisor to manage domains. As discussed in D2.1.6, the `cgroup` subsystem appearing in Linux kernel (version 2.6) has been seen as a new partition technology. It provides a mechanism for aggregating/partitioning sets of processes, and all their future children, into hierarchical groups with specialized behaviors. Linux allows a control group of process hierarchy to be associated with subsystems (scheduling, memory management, accounting, etc. A subsystem is a module, which makes use of the process grouping facilities provided by cgroups, to treat groups of processes in particular ways. A subsystem is typically a "resource controller" that schedules a resource or applies limits predefined for each cgroup.

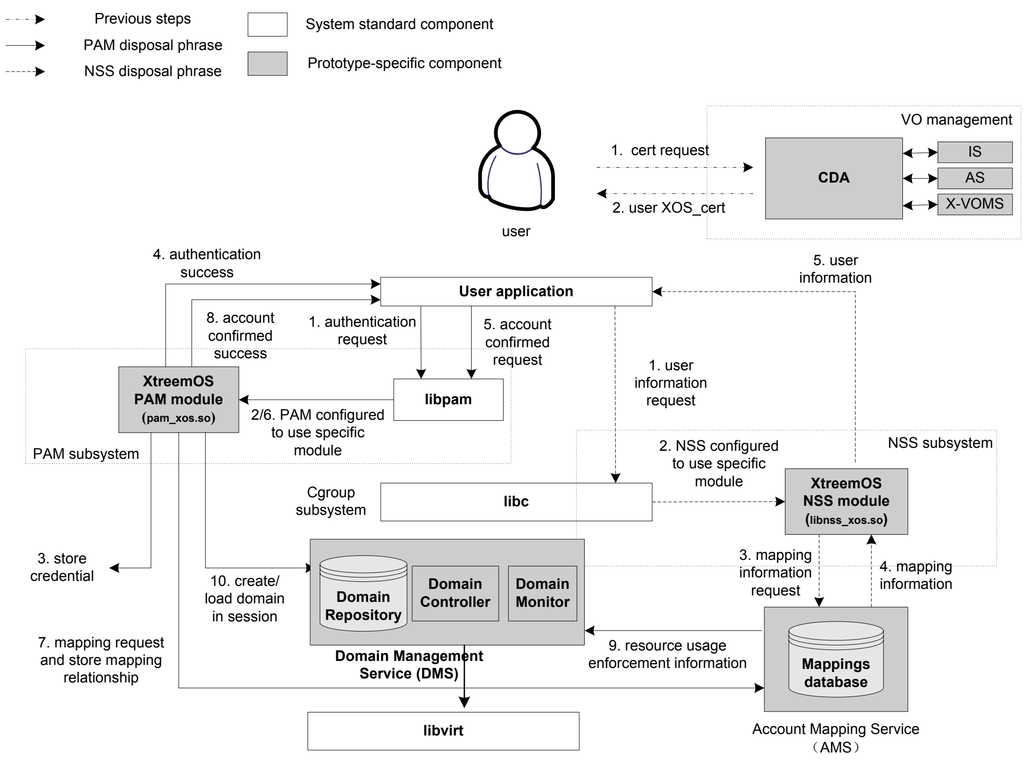
Previous steps
PAM disposal phrase
NSS disposal phrase
System standard component
Prototype-specific component
VO management
1. cert request
2. user XOS_cert
CDA
IS
AS
X-VOMS
user
4. authentication success
User application
5. user information
8. account confirmed success
1. authentication request
5. account confirmed request
1. user information request
XtreemOS PAM module (pam_xos.so)
libpam
2/6. PAM configured to use specific module
PAM subsystem
2. NSS configured to use specific module
NSS subsystem
XtreemOS NSS module (libnss_xos.so)
Cgroup subsystem
libc
3. store credential
10. create/ load domain in session
Domain Repository
Domain Controller
Domain Monitor
3. mapping information request
4. mapping information
7. mapping request and store mapping relationship
Domain Management Service (DMS)
9. resource usage enforcement information
Mappings database
libvirt
Account Mapping Service (AMS)

Figure 2: Overview of Architecture

From Linux kernel version 2.6.27, many subsystems such as PID namespace, memory/network usage subsystem, have been integrated in kernel container. We can make use of existing functionalities or develop custom function subsystem to achieve our goal. Figure 4 depicts the interaction between cgroup based DMS with other modules.

A detailed introduction of linux container support in `libvirt` is presented in [9]. Here we briefly introduce the implementation approach.

5.2.1 Isolation of resources

For PID isolation, each process can only see the process belonging to same container. The view of `ps` command only presents the processes in the same container. One process can not be allowed to access or communicate with other processes in other containers. The isolation of memory access is done by cgroup subsystem. Memory limitation can be done with containers (still need a `cgroup` patch). Each container can be allocated given memory. Currently, the memory limitation is only applied to physical memory rather than virtual memory. The limitation of network bandwidth can be archieved by blocking network traffic for

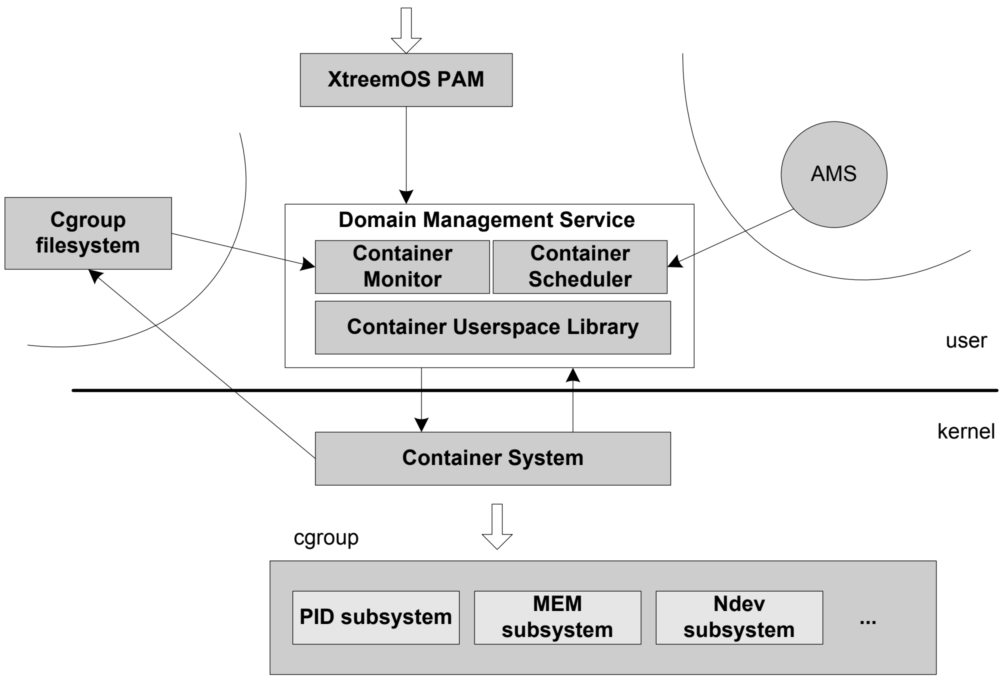

Figure 3: Default DMS implementation based on cgroup support

a given network namespace.

5.2.2 Container monitor and scheduler

The container monitor here is to provide QoS related information to scheduler. By inspecting the `/proc` file system of each container (i.e. cgroup file system), the container monitor could perform accounting of resource usage for each container. Container scheduler freeze (or unfreeze or execute) containers based on accumulated accounting information. Container scheduler is able to switch the status of multiple containers in a priority-based or time sharing-based policy. The container scheduler also provides the support for checkpoint/restart and migration of containers.

5.3 Network Issues

By default, domains are assigned internal IP address in virtual networks. That should not affect VO users submitting jobs as AEM hide the running details from users. But for VO users accessing domains directly from external networks (e.g. via `xos-ssh`), normally domains need public IP addresses as the physical node has. The alternative way is to use port forwarding on physical node (the limitation is that one port is only mapped to one domain). This issue can be also addressed in the application level. For example, `xos-ssh` client sends specific domain addresses to `xos-sshd`, and the `xos-sshd` relays traffic to domains according to target domain addresses.

5.4 Monitoring and Accounting

Generally speaking, large part of monitoring and accounting works could be done in higher level services like execution management services. With introducing virtualization and domains, monitoring and auditing works become complicate. For example, while VMM can allocate fixed shares of CPU among competing VMs, it is also necessary to account for work done on behalf of individual VMs in device drivers (e.g. multiple VMs share I/O paths in hypervisor or domain-0 and this part of overhead needs to be logged). In other words, the accurate measure of consumed resources by domains should include the I/O work done in the host.

For monitoring the physical nodes, there are many choices. Ganglia [21] is cluster-wide tool currently used to gather information of physical nodes though it may be not aware of virtualized nodes. Nagios [8] is another open source monitoring program for hosts, services and networks and generally used for alerting. It is possible to develop Nagios plugin for monitoring VMs.

Unlike the software that monitors the underlying hardware, the software that monitors the hypervisor depends on the type of hypervisor. For example the Argo project is a simple, extensible, framework for monitoring and controlling a host running multiple Xen instances. The Unix/Linux system management tool monit can also be used to watch hypervisor processes. The XenMon [18] tools make use of the existing Xen tracing feature to provide fine grained reporting of various domain related metrics. The xenbake daemon keeps a large amount of history in a shared memory area that may be accessed by tools such as xenmon. Monitoring applications running inside VMs is no different than monitoring applications running on a physical server.

6 Conclusion

In this report, we have investigated state-of-art virtualization techniques and chose a feasible approach to achieve advanced VO support in terms of strong isolation and enforcement of resource usage. For each VO user, a domain (VM) is allocated once a PAM session starts. VMs enable fault isolation and encapsulating of different applications in self-contained execution environments so that a failure in one virtual machine does not affect other VMs hosted on the same physical hardware. Individual VMs are often configured with performance guarantees and expectations, e.g., based on policies or service level agreements. Thus, the resource consumption of one VM should not impact the promised guarantees to other VMs on the same hardware. We also discussed using the default hypervisor based on cgroup support in recent Linux kernel.

Bibliography

[1] Amazon elastic compute cloud (amazon ec2). http://aws.amazon.com/ec2/.

[2] libvirt:virtualization api. http://libvirt.org/.

[3] Linux containers. http://lxc.sourceforge.net/.

[4] Openvz. http://openvz.org/.

[5] Virtuozzo. http://www.parallels.com/virtuozzo/.

[6] Vmware. http://www.vmware.com.

[7] G. Banga, P. Druschel, and J.C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. *OPERATING SYSTEMS REVIEW*, 33:45–58, 1998.

[8] W. Barth. *Nagios: System And Network Monitoring*. No Starch Press, Inc, 2006.

[9] Daniel P. Berrange. *An introduction to libvirt's LXC (LinuX Container) support*, 2008. Available at `https://lists.linux-foundation.org/pipermail/containers/2008-September/013237.html`.

[10] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall. Cloud Computing, 2007.

[11] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: an overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication Review*, 33(3):3–12, 2003.

[12] XtreemOS consortium. D2.1.2: Design and Implementation of Node-level VO Support. November 2007.

[13] XtreemOS consortium. D2.1.6: Evaluation of Linux native isolation mechanisms for XtreemOS flavours. November 2008.

[14] E. Courses and T. Surveys. A Comparison of Virtualization Technologies for HPC. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 861–868, 2008.

[15] B. des Ligneris. Virtualization of Linux Based Computers: The Linux-VServer Project. In *Proceedings of the 19th International Symposium on High Performance Computing Systems and Applications*, pages 340–346. IEEE Computer Society Washington, DC, USA, 2005.

[16] W. Emeneker, D. Stanzione, and H.P.C. Initiative. HPC Cluster Readiness of Xen and User Mode Linux. In *2006 IEEE International Conference on Cluster Computing*, 2006.

[17] A. Gavrilovska, S. Kumar, H. Raj, K. Schwan, R. Nathuji, V. Gupta, R. Niranjan, A. Randive, and P. Saraiya. High-Performance Hypervisor Architectures: Virtualization in HPC Systems. In *Workshop on High Performance Virtualization (HPCVirt) in conjunction with EuroSys*, 2007.

[18] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling Tool. *Technical ReportHPL-2005-187, Hewlett-Packard Development Company, LP*, 2005.

[19] M. Lageman and S.C. Solutions. Solaris ContainersąłWhat They Are and How to Use Them. *Sun BluePrints OnLine*, pages 819–2679, 2005.

[20] Barham P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[21] F.D. Sacerdoti, M.J. Katz, M.L. Massie, and D.E. Culler. Wide Area Cluster Monitoring with Ganglia. In *Proceedings of the IEEE Cluster 2003 Conference*, 2003.

[22] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux Security Module. *NAI Labs Report# 01*, 43, 2001.

[23] S. Soltesz, H. Pötzl, M.E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2007 conference on EuroSys*, pages 275–287. ACM Press New York, NY, USA, 2007.

[24] L. Youseff, R. Wolski, B. Gorda, and C. Krintz. Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. In *Proc. of*

International Workshop on Virtualization Technologies in Distributed Computing (VTDC), 2006.