Xen Management with SmartFrog
On-Demand Supply of Heterogeneous, Synchronized Execution Environments

Xavier Gréhant[1,2], Olivier Pernet[3], Sverre Jarp[1],
Isabelle Demeure[2], and Peter Toft[4]

[1] CERN openlab, Geneva, Switzerland
[2] École Nationale Supérieure des Télécommunications, Paris, France
[3] ENSIMAG, Grenoble, France
[4] Hewlett-Packard Laboratories, Bristol, UK
xag@cern.ch

Abstract. Applications to be executed on multipurpose Grids frequently have very specific resource requirements (platform, kernel, operating systems, libraries, memory, CPU, etc.) and need to be delegated part of the resource control. Typical Grid sites offer a limited range of resource types, inhibiting the range of applications that can be supported; and Grid node managers are bound to maintain their servers according to users requirements. To address these problems, we introduce SmartDomains, which combines the high performance virtual machine technology provided by Xen, with automatic deployment of Xen virtual machines using the SmartFrog configuration and deployment framework. SmartDomains automatically deploys distributed, synchronized pools of custom-configured Xen virtual machines and manages them through their lifecycle as a single coherent distributed execution environment. SmartDomains uses a representation of the complete distributed resources specifications, including information about how to sequence their creation and removal. We discuss SmartDomains test cases at CERN for distributed testbeds and Grid execution nodes.

1 Introduction

Although virtual machine (VM) technology has been around for four decades or more [1,2], there has been a resurgence of interest in recent years as virtualization has become practical on commodity hardware [3]. It follows that grid resource management tools will evolve to embrace support for virtual resources and those that do not will risk obsolescence. Our contribution to this evolution is a tool called SmartDomains (SD), which automatically supplies custom-configured, distributed virtual execution environments targeted at running batch Grid jobs or conducting system tests. In these contexts, it is important to keep independent the activity on the utilized resources and the maintenance of the backing hardware. SD provides fast, integrated VM control and a representation of resources to decouple administration and usage. A comparable *resource control plane* was developed in COD [4] for nodes provisioning in Beowulf clusters, and in PlanetLab [5] for managing networked applications' points of presence. Scientific production Grids (e.g. LCG, EGEE, OSG) do not yet optimise back-end resource usage with platform virtualization [6], which would add extra dimensions of flexibility in terms of resource configuration and fractional physical resource allocation (as illustrated with Tycoon [7]) although *Globus Virtual Workspaces* [8] already make interactions with the VM Monitors (VMMs) a Grid service.

The goal of SD is to provide a simple-to-use yet powerful mechanism for describing a required set of virtual machine resources, and a fully-automated deployment system to create VMs according to the supplied description. The automated deployment engine is a peer-to-peer distributed layer that takes in resource request descriptions and realises them with the requested sequencing. The same deployment engine is used to deactivate resource requests and release their resources, again according to the specified sequencing. Our experience with this approach leads us to believe that it makes it easier for resource administrators to prepare and control virtual resources, and for developers to create new functionality.

In this paper we discuss SD from three perspectives: In section 2 we explain the component technologies used in the SD system. SD usage is explained in section 3 along with a comparison of other systems using virtualization for resource management, and performance measurements are presented. In section 4 we illustrate the benefits of SD for resource administration, and its contributions to research and development in resource management systems.

2 SmartDomains, a Novel Approach

SD builds on Xen [9] for virtualization, and on SmartFrog [10,11] for the resource description and deployment mechanisms. Using virtualization in batch execution environments offers resource consumers and resource providers the following benefits:

Software compatibility: By creating a library of customized VM images, VMs can easily replicate a very wide range of resource configurations, satisfying the specific needs of a wide range of applications.

Resource sharing and performance isolation: By running multiple VMs on the same physical machine, fractional resources can be allocated, with fine-grained control over the resource consumption of each VM.

Failure isolation: VM failures do not affect the physical node nor other VMs.

We chose the Xen virtualization technology [9] for its high performance, openness, advanced features (live VM migration, for example) and growing popularity. Xen allows VMs to run at near native speed, which is critical for high-performance computing applications. However, the SD approach could be applied to other virtualization technologies such as VMWare [12], or in-kernel virtualization approaches such as KVM [13]. The resource description and deployment mechanism in SD is built using SmartFrog (SF), a Java-based framework for the configuration, deployment and management of distributed software systems, developed by HP Labs. In the domain of utility computing, the SoftUDC project [14] illustrates the use of SF along with VMMs for centralized management features. SF encourages the separation of the functionality of a software component from its configuration details. This allows the development of configuration-driven systems, the behaviour of which, including deployment choices, can be determined by configuration data. SF provides:

– A rich description language to express the configuration of software components (pieces of software to be placed on distributed hosts), and to express their orchestration at run-time using various composition components.
– A deployment engine formed from a distributed, peer-to-peer network of SF daemons. The deployment engine interprets the description language, dispatches software component deployment and management actions to local or remote daemons, checks liveness and maintains dependencies and references for attribute lookup and remote method invocation.

A domain is a running Xen virtual machine. SD defines a set of SF components that configure, monitor, deploy and manage Xen virtual machines. A user submits a description of the distributed environment he requires, specifying with complete freedom the kernels, filesystem images (distribution, libraries), computing resources (memory, CPU, hard drive size, etc), and orchestration details (e.g., the deployment sequence). SD automatically deploys the description and manages the deployment process until the resources are no longer required. Direct access to the physical nodes or to the Xen VMMs is not necessary. SD has been used in production since early in its development to create distributed virtual testbeds for the task of integrating and certifying gLite, a major grid middleware software distribution. This caused SD to support fast deployment of complex distributed configurations. We are now experiencing its integration in EGEE production Grid. In this context, SF dynamically boots appropriate execution environments upon request from a VO[1], and thus improves the compromise between resource delegation to the user, and control by the node manager.

3 Usage: A Comparative Overview

SD requires users to create descriptions for their virtual resource pools. There are simple extension and composition mechanisms that make it easy to create a library of different virtual resource pools, and to share these amongst users. It is simple to deploy a description, to un-deploy it, and to do so repeatedly. We compare this approach with other tools.

3.1 Launching a Virtual Pool

To start a deployment and launch a virtual pool, users submit a description with a single command line, and use another command line to un-deploy (remove)

[1] Virtual Organization, a federation of Grid users.

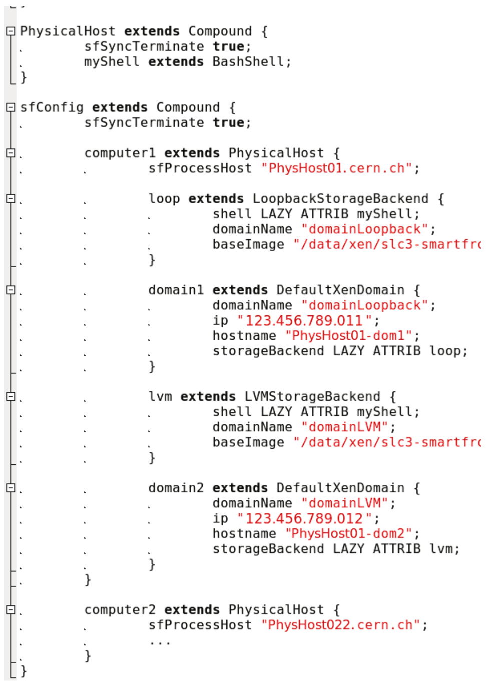
the virtual pool: `> sfStart localhost pool vp.sf`. The user specifies where deployment is to be initiated (*localhost* here), provides a name for identification at runtime (*pool*), and provides the description (*vp.sf*) of resource requirements: `> sfTerminate localhost pool`. Behavior on termination is defined in the description as well; by default it shuts down all virtual domains and cleans up the physical machines to return them to their initial state.

SD is distinctive in that it reduces the virtual resource management burden to the simplicity of an "on/off" button. It can automatically deploy the appropriate virtual resources on grid sites to handle incoming jobs or job workflows, or it can be used for repetitive, varied testbed setups for quality assurance tasks. A number of advanced enterprise resource management systems (Platform VM Orchestrator [15], Cassat Collage [16], OpenQRM [17], DynamicOE [18]) also leverage virtualization, but generally for a different purpose: they let the site administrator flexibly allocate computer center resources across long-lived applications, typically addressing resource utilization and high-availability concerns. Open source virtualization management projects (Virtual Workspaces, GPE, Enomalism) focus on presenting the VMM control via a variety of different interfaces. All these systems makes VMMs remotely accessible. SD development began with many of these ideas in mind, but its evolution was driven by specific requirements emerging from the CERN computing environment. Hence we focused on building a highly-configurable, highly-automated system that minimizes user interaction.

3.2 Describing a Virtual Pool

A single logical description specifies the actions that the distributed deployment system will take in order to deploy, and un-deploy, the requested virtual resource pool. The only current, known limitation to the run-time, distributed scope of a SD description is the presence of firewalls that block Java RMI communication, which is used for peer-to-peer communication by the SF daemons. A virtual resource pool deployment can vary in scale from one host to a complete data center. The RMI limitation will be overcome in future implementations by substituting RMI for a more firewall-friendly protocol, such as REST-based communication over HTTP. SD descriptions permit all possible configurations allowed by the Xen VMM, expressed as attribute values[2]. It differs in this respect from Amazon EC2, which provides fixed virtual machine resource configurations, and is elastic only in terms of the number of machines. However, like Xenoservers [19], EC2 does let users upload their own VM images. In Tycoon, it is the resource provider that decides what image is used. SD is linked to OSFarm[20] for custom image generation. In addition to configuring a domain, attributes define other behaviors, such as saving the image after use, compressing / uncompressing it. For ease of use, however, almost all attributes have default values. As a consequence, describing a single virtual domain with SD requires less knowledge than

[2] For brevity, they do not appear on figure 1. For a full list, please see the tutorial: `www.cern.ch/smartdomains`.



```
PhysicalHost extends Compound {
        sfSyncTerminate true;
        myShell extends BashShell;
}

sfConfig extends Compound {
        sfSyncTerminate true;

        computer1 extends PhysicalHost {
                sfProcessHost "PhysHost01.cern.ch";

                loop extends LoopbackStorageBackend {
                        shell LAZY ATTRIB myShell;
                        domainName "domainLoopback";
                        baseImage "/data/xen/slc3-smartfr
                }

                domain1 extends DefaultXenDomain {
                        domainName "domainLoopback";
                        ip "123.456.789.011";
                        hostname "PhysHost01-dom1";
                        storageBackend LAZY ATTRIB loop;
                }

                lvm extends LVMStorageBackend {
                        shell LAZY ATTRIB myShell;
                        domainName "domainLVM";
                        baseImage "/data/xen/slc3-smartfr
                }

                domain2 extends DefaultXenDomain {
                        domainName "domainLVM";
                        ip "123.456.789.012";
                        hostname "PhysHost01-dom2";
                        storageBackend LAZY ATTRIB lvm;
                }
        }

        computer2 extends PhysicalHost {
                sfProcessHost "PhysHost022.cern.ch";
                ...
        }
}
```

Fig. 1. Description with minimal configuration

writing a Xen configuration file. The description language lets the user configure how virtual machines should be synchronized (on figure 1 *Compound* is an example of synchronization type). It addresses the need for lifecycle management mechanisms cited on the Xen road-map [21], and is necessary for distributed batch jobs and tests. A management console provides a view of deployed virtual resource pools at runtime, and lets users update their configuration. For example, to change the frequency of virtual domains liveness checking, change memory allocation, or alter whether the image will be saved after shutdown.

3.3 Measurements and Future Work on Performance

Our measurements show that SD overhead is negligible on modern systems, for a considerable gain in resource flexibility. Our first tests with 48 CPU-intensive benchmark runs did not show any significant overhead introduced by SD. We found only a 0.25% difference in minimum elapsed times between these configurations: no liveness checking and liveness checking every 2 seconds, and between checks every 2 and 10 seconds. We monitored memory consumption in different scenarios (figure 2): a single deployment shows that an idle SD daemon needs about 20MB, and booting and terminating a first VM requires about 3 more MB. (We did identify a memory leak: each additional VM requires an additional 300KB which is never surrendered. For the moment the daemon can just

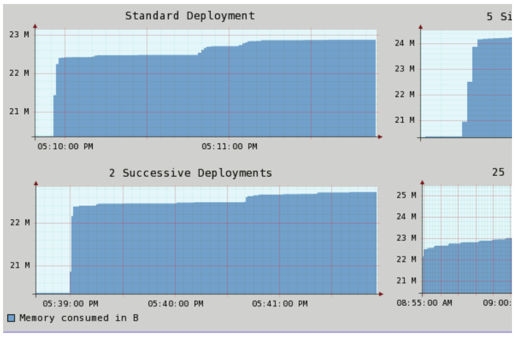


Fig. 2. Memory measurements

be restarted after every few hundred deployments.) For successive, cumulative deployments, we have been restricted by disk space so far. For simultaneous, multithreaded deployments, currently 8 VMs can be booted together on the same host; this limitation will be removed in the future. As shown on figure 2, booting 5 VMs simultaneously requires about 24 MB total, and all the VMs are booted after about 40 seconds. The time to change an environment is the same as VMPlants'[22] best case, a complementary work that minimizes software installation time on VMs.

4 Extending SmartDomains

SD is easy to extend for both resource administrators and developers. Administrators can easily provide and extend resource pool descriptions, that can be offered to their users. Developers can easily extend the SD framework through the addition of new control components.

4.1 Administration: Specialize by Composing and Pre-configuring

The first prerequisite is to install the Xen hypervisor on every physical machine. Installing SD is then just a matter of downloading the SD distribution and running an Ant command. It is therefore a simple process to install SD in large computer installations. And, while SD is targeted at large deployments, the simplicity of installation makes it quite usable on a single laptop for local VM deployments. Any computer can trigger the deployment of new virtual resource on the whole pool of physical hosts of which it is part, and where the virtual machines actually reside does not matter. There are no *a priori* privileged hosts and every node can act as the root for the whole site's computing power. SD distributes the work across the distributed computing environment and avoids bottlenecks and single points of failure. SD makes resource control simple in cases where it is delegated to a "trusted community" (e.g. physicists working on the same CERN experiment) or operated inside the same organization. In other cases, an additional admission control component may restrict deployment possibilities, and a user interface will regulate access to this component to restrict deployments. The extension mechanism (example of *domain1 extends XenDomainDefault* on figure 1) adds or overwrites component attributes that will be resolved at run-time as configuration data available to the corresponding class. This lets administrators create partly-configured descriptions which can be easily extended by users with all the necessary data. The description language allows attributes to be flexibly linked upwards (*ATTRIB*, *PARENT:*), or downwards (*name: childName: attribute*) in the hierarchy. Used in conjunction with LAZY keyword, this indicates a reference to be used for remote method invocation; without LAZY, it copies the target of the link. This allows administrators to define specialized components that populate low-level base components with user-level attributes. In our use-cases, these mechanisms improved convenience while using pre-defined templates.

4.2 Development: Enriching SD by Plugging in New Logic and Composite Structures

The basic SF approach is to describe a hierarchical organisation of software components and their configuration data; when deployed, each component interprets its configuration data, which drives the component's behavior. This proceeds in a hierarchy from the root component downwards. We mentioned how important it is for SD to not require any interaction with the user in order to accept batch requests for non-interactive jobs or software quality assurance tasks (section 3.1). Hence automation is vital, and new forms of automation can be developed by extending the set of SF/SD components. As an example, to dispatch virtual machines across physical hosts, a `Scheduler` component chooses the next suitable host, and a `Schedulee` wraps in the description the components to be placed.

For fine-grained resource sharing via a bidding system, a definitive advantage of Tycoon over previous works was the *best response algorithm* that bids in place of the user to avoids the need for frequent interaction. Indeed, we contend that computing resource management should be transparent and automated without restricting functionality. Combining the ability to manipulate descriptions with the ability to easily add new components allows us to balance functionality with automation and transparency. The mechanisms provided in SF for flexibly composing components at runtime allow developers to easily implement new behaviors (high availability, scheduling and load balancing mechanisms) and to apply these to the management of SD virtual resource pools. There is no restriction in the types of components that can be created, and they can easily fit into the framework and descriptions. Following the lessons of PlanetLab experience[5] SD favors evolutionary design more than *clean slate* design. In most other systems from enterprise resource management systems to research prototypes, adding new functionality can have significant implications for the whole system structure, thus restricted to some usage policies or available hardware (e.g. the two hard-coded availability classes in DynamicOE). SD allows the implementation of distributed algorithms involving VMs located on the whole peer-to-peer network of trusted daemons, to contrast with Tycoon where real-time auction for resources is limited to a competition between virtual machines on a per-physical-host basis [7].

5 Conclusion

SD deploys virtual resource pools for batch jobs or tests. Our use cases at CERN drove us towards high configurability, and high automation. Interaction is still possible, but the composable, component-based structure of the system allows automation functionality to be easily added. Generality is preserved when writing a description or specializing components, because the extension mechanisms allow descriptions to be prepared in advance and then customized for every deployment. Configurability, composition and lifecycle management are provided to the user or administrator through the description language. These are novel characteristics in a resource management system. they provide the necessary flexibility to decouple administration and usage requirements on the Grid *resource control plane*.

References

1. Goldberg, R.P.: Survey of virtual machine research. Computer, 34–45 (June 1974)
2. Denning, P.: Performance analysis: Experimental computer science at its best. Communications of the ACM 24(11), 725–727 (1981)
3. Rosenblum, M.: The reincarnation of virtual machines. ACM Queue 2(5), 34–40 (2004)
4. Chase, J.S., Irwin, D.E., Grit, L.E., Moore, J.D., Sprenkle, S.E.: Dynamic virtual clusters in a grid site manager. In: HPDC 2003: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, p. 90. IEEE Computer Society, Washington (2003)
5. Peterson, L.L., Roscoe, T.: The design principles of planetlab. Operating Systems Review 40(1), 11–16 (2006)
6. Nabrzyski, J., Schopf, J.M., Weglarz, J.: Grid Resource Management: State of the Art and Future Trends, 1st edn. Springer, Heidelberg (2003)
7. Feldman, M., Lai, K., Zhang, L.: A price-anticipating resource allocation mechanism for distributed shared clusters. In: EC 2005: Proceedings of the 6th ACM conference on Electronic commerce, pp. 127–136. ACM Press, New York (2005)
8. Foster, I., Freeman, T., Keahy, K., Scheftner, D., Sotomayer, B., Zhang, X.: Virtual clusters for grid communities. In: CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid, pp. 513–520. IEEE Computer Society, Washington (2006)
9. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164–177. ACM Press, New York (2003)
10. Goldsack, P., Guijarro, J., Lain, A., Mecheneau, G., Murray, P., Toft, P.: Smartfrog: Configuration and automatic ignition of distributed applications. Technical report, HP (2003)

11. Pathak, J., Treadwell, J., Kumar, R., Vitale, P., Fraticelli, F.: A framework for dynamic resource management on the grid. Technical report, HPL (2005)
12. Waldspurger, C.A.: Memory resource management in vmware esx server. In: OSDI 2002: Proceedings of the 5th symposium on Operating systems design and implementation, pp. 181–194. ACM Press, New York (2002)
13. Qumranet: Kvm: Kernel-based virtualization driver. Qumranet White-Paper (2006)
14. Kallahalla, M., Uysal, M., Swaminathan, R., Lowell, D.E., Wray, M., Christian, T., Edwards, N., Dalton, C.I., Gittler, F.: Softudc: A software-based data center for utility computing. Computer 37(11), 38–46 (2004)
15. Computing, P.: Platform VM orchestrator., `http://www.platform.com/`
16. Cassatt: Cassatt collage., `http://www.cassatt.com/prod_virtualization.htm`
17. Qlusters Inc. 1841 Page Mill Road, G2, Palo Alto, CA 94304: openQRM Technical Overview: Open Source - Data Center Management Software (November 2006)
18. FusionDynamics: Fusiondynamics - `http://www.fusiondynamics.com`
19. Kotsovinos, E., Moreton, T., Pratt, I., Ross, R., Fraser, K., Hand, S., Harris, T.: Global-scale service deployment in the xenoserver platform. In: Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS 2004), San Francisco (December 2004)
20. Bjerke, H., Rattu, D., Habib, I.: OSFarm
21. XenSource: Xen roadmap, `http://wiki.xensource.com/xenwiki/XenRoadMap`
22. Krsul, I., Ganguly, A., Zhang, J., Fortes, J.A.B., Figueiredo, R.J.O.: Vmplants: Providing and managing virtual machine execution environments for grid computing. In: SC 2004: Proceedings of the ACM/IEEE SC 2004 Conference on High Performance Networking and Computing, Pittsburgh, PA, USA, p. 7. IEEE Computer Society, Los Alamitos (2004)