Efficient Data Management Support for Virtualized Service Providers

Íñigo Goiri, Ferran Julià and Jordi Guitart
Barcelona Supercomputing Center - Technical University of Catalonia
Jordi Girona 31, 08034 Barcelona, Spain
{inigo.goiri, ferran.julia, jordi.guitart}@bsc.es

Abstract

Virtualization has been lately introduced for supporting and simplifying service providers management with promising results. Nevertheless, using virtualization introduces also new challenges that must be considered. One of them relates with the data management in the provider. This paper proposes an innovative approach for performing efficiently all the data-related processes in a virtualized service provider, namely VM creation, VM migration and data stage-in/out. Our solution provides a global repository where clients can upload the task input files and retrieve the output files. In addition, the provider implements a distributed file system (using NFS) in which each node can access its own local disk and the disk of the other nodes. As demonstrated in the evaluation, this allows efficient VM creation and task execution, but also task migration with minimum overhead, while keeping it accessible during the whole process.

1 Introduction

Nowadays, service providers offer software as a service over the Internet, allowing the user to forget about any problem related with software deployment or maintenance. Nevertheless, this approach moves all the management problems from the customer side to the provider side. Recently, the use of virtualization has been explored for cost reduction and easier management in service providers. Virtualization allows the consolidation of services, multiplexing them onto physical resources while supporting isolation from other services sharing the same physical resource, reducing in this way provider's costs and maintaining the integrity of the underlying resources and the other services.

Virtualization has other valuable features for service providers. It offers the image of a dedicated and customized machine to each user, decoupling them from the system software of the underlying resource. This reduces the management efforts in the provider, because this allows having a new VM just copying a previously created or moving the application from a node to another if needed. In addition, virtualization allows agile and fine-grain dynamic resource provisioning by providing a mechanism for carefully controlling how and when the resources are used, and primitives for migrating a running machine from resource to resource if needed.

However, using virtualization in service providers introduces new challenges that must be resolved in order to take the maximum benefit of the virtualization capabilities. One of these challenges relates to data management, since using virtualization involves dealing with a great amount of data, including the VM images or the input files of the applications, which must be accessible from all the nodes in the provider as fast as possible.

This paper presents an innovative data management solution for performing efficiently all the data-related processes in a virtualized service provider (VSP), namely VM creation, VM migration and data stage-in and stage-out. Our solution provides a global repository where clients can upload the input files needed by the tasks to be executed in the provider and retrieve the output files. Customized VM images can be also stored in the repository for being reused in the future.

In addition, the provider implements a distributed file system (using NFS) in which each node can access its own local disk and the disk of the other nodes. This allows each node creating VMs and executing tasks efficiently using the local disk. Furthermore, tasks can be also migrated with minimum overhead, since it is not necessary to transfer the VM image, while maintaining their accessibility during the whole process. Moreover, migrated tasks can access remotely their required data without noticeable performance penalty.

The rest of the paper is organized as follows: Section 2 describes the architecture of a VSP. Sections 3 and 4 present our solution for managing data inside a VSP. Section 5 describes the evaluation of the introduced approach. Section 6 presents some related work in this area. Finally, Section 7 exposes the conclusions of this paper and the future work.

2 Virtualized Service Providers

This section gives an overview of the architecture of a VSP, in order to contextualize our data management solution. Additional details can be found in [4]. As shown in Figure 1, a VSP consists of a single global *Scheduler* and one *Virtualization Manager (VtM)* per node.Virtualization is supported by using Xen [2].

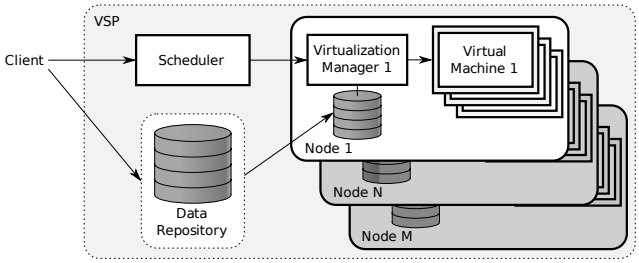

Figure 1. VSP architecture

The *Scheduler* periodically decides about the task placement in the nodes and the amount of allocated resources to each task in the corresponding node. The *Virtualization Manager (VtM)* is responsible of creating new customized VMs, executing tasks on these VMs, monitoring their execution and finally extracting the results and destroying the VMs.

An interaction starts when a client sends a task and its description to the *Scheduler* and uploads the needed data into a data repository. Then the *Scheduler* decides the node that will execute the task and sends it to the corresponding *VtM* that creates the VM where the task will be executed. Notice that creating a VM requires extensive disk usage in order to install the guest system and deploy the needed software. In addition, the input data of the task must be accessible to it inside the VM. Once the task has finished, the *Scheduler* asks the *VtM* to destroy the VM. Finally, the *VtM* copies the output data to the data repository in order to make it available to the user.

3 Support for Data Stage-in/out

Typically, tasks need some input data in order to be executed, and when they finish their execution, they generate some output. For example, a video encoder needs an input video file and it generates an encoded video file. When submitting a task, the client has to provide this input data to the provider. Similarly, when the task finishes, the client must be able to access the output data. Additionally, in a virtualized service provider this data management has to deal with having dedicated and isolated virtual environments for each task.

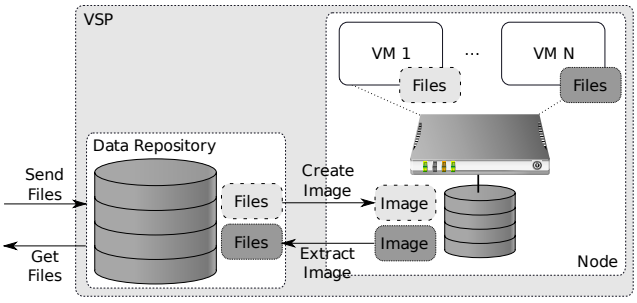

Figure 2. Data stage-in/out

We propose a global data repository in the provider which stores the information of every client taking into account who is the owner of each file. This repository provides services for performing the data stage-in and stage-out. As shown in Figure 2, before submitting a task to the provider, the client uses these services for uploading the required input data to the repository. This uploading process can be performed by FTP, SFTP or HTTP. When the VM that will execute a task is being created, the data management service gets the required information from the data repository, creates a disk image containing this information and mounts this image in the home space of the VM. In the same way, when the task finishes its execution, the data management system extracts the output information from the VM image and uploads it to the data repository, where it can be retrieved by the client using the protocols commented above.

Furthermore, this data repository allows also storing the VM images used by each customer. These images can be later reused for creating new VMs.

4 Support for Fast VM Creation and Migration

The creation of a VM image involves dealing with a large amount of data, including the installation of the guest operating system and the deployment of the required software. Optimizing the access to this information is mandatory to create the VM as fast as possible. Virtualization allows also moving VMs between nodes, that is migrating VMs. This implies creating a snapshot of the memory and the disk, and moving it between nodes. In addition, it is desirable that the VM remains always available to the clients in spite of being migrated (i.e. live migration). This requires the disk to be available on both nodes, and dealing with large (order of gigabytes) data images, which makes this a complex problem.

Works such as [6] and [12] propose solutions using a NFS server that provides the images to every node. However, this approach has some performance problems, since it implies working remotely in the VM creation phase. This

server also becomes a huge bottleneck since every node in the cluster will access to it. The alternative solution of using only a local file system on every node exhibits good performance for creating the VMs (it is done locally to the node) but implies transferring the whole VM image when the task is migrated. Taking into account that a minimal VM image has around 2 GB of data, this makes impossible to support live migration.

In contrast with this, our solution consists of a distributed data management system where every node generates and manages its own images. We propose having a NFS system distributed among the nodes. In other words, each node has a NFS server that can be accessed from all the other nodes, as shown in Figure 3. By using this approach, each node can work locally when creating the VMs and these images are locally and remotely available when needed (e.g. during task execution). This avoids explicit image transfers when migrating a VM.

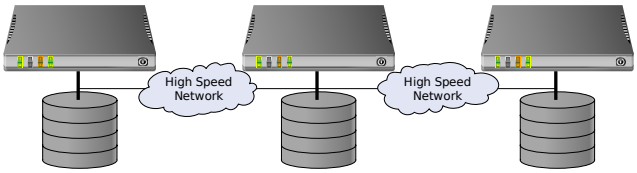

Figure 3. Distributed file system

5 Evaluation

This section evaluates our data management proposal comparing its performance with other data models. The evaluation focuses on the data-related processes that occur in a virtualized service provider, namely the VM creation, the data access during task execution and the VM migration.

5.1 Experimental environment

The applications used in the experiments include a batch application with high CPU consumption and an application server. The first application is a video file encoder, mencoder [9]. The second application, RUBiS (Rice University Bidding System) [10], simulates a auction service like eBay, and has been deployed in the Tomcat [1] server.

Our experimental testbed consists of two hosts, *Host A* which is a 64-bit architecture with 4 Intel Xeon CPUs at 3.0GHz and 10Gb of RAM memory, and *Host B* which is a 64-bit architecture with an Intel Xeon CPU at 2.6GHz and 4Gb of RAM memory. The hard disk features of both hosts (extracted with *hdparm*) are presented on Table 1. In addition, some other machines are used as clients for the application server. All the machines are connected through a Gigabit Ethernet.

Host	Cache read	Buffered disk read
A	3190.15 MB/s	62.01 MB/s
B	2848.53 MB/s	58.61 MB/s

Table 1. Hard drive features

5.2 VM creation performance

The creation of a VM has strong disk requirements, since it must create a disk image and install the guest operating system inside. Our experiments reveal that the whole creation of a VM (including the operating system installation and the deployment of the needed software) needs 6.4" at *Host A* and 2.8" at *Host B* if the local disk is used, while it takes 13.5" at *Host A* and 5.3" at *Host B* when using a remote disk accessed via NFS. Our approach benefits from this because it uses the local file system for creating the VMs.

5.3 Task execution performance

The influence of data management solutions on applications performance is variable, since the number of disk I/O operations depends on the type of application. For example, an application server will probably use less disk than a video encoding application or than a simple file copy. This section shows some experiments that evaluate the performance of different types of applications using both local and remote disks. The first experiment consists of stressing the disk by copying a file of 580 MB, while the second one consists of running a *mencoder* application.

Table 2 shows the time needed to execute both tasks in our two hosts when accessing the local disk and a remote disk via NFS. Notice that, using a local disk provides better performance for copying a file. It is also noticeable the influence of the disk speed on the obtained speedup. In addition, though *mencoder* makes extensive use of the disk, there is not any difference on executing remotely and locally. This occurs because the bottleneck of this application is the CPU and accessing the remote disk using a fast Ethernet is enough to fulfill the disk requirements of this task.

	file copy		memcoder	
Host	local	remote	local	remote
A	4.1"	11.4"	583"	584"
B	11.3"	21.0"	640"	640"

Table 2. Run times of file copy and memcoder

Our approach uses the local disk on each node for task execution, which has demonstrated to be the fastest solution for all the types of applications. Only tasks which have

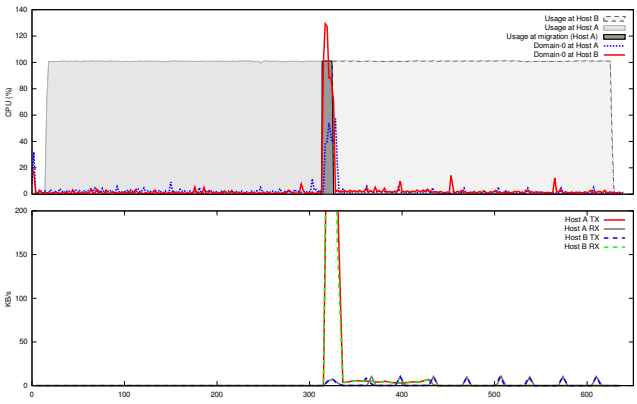

Figure 4. Migrated memcoder execution

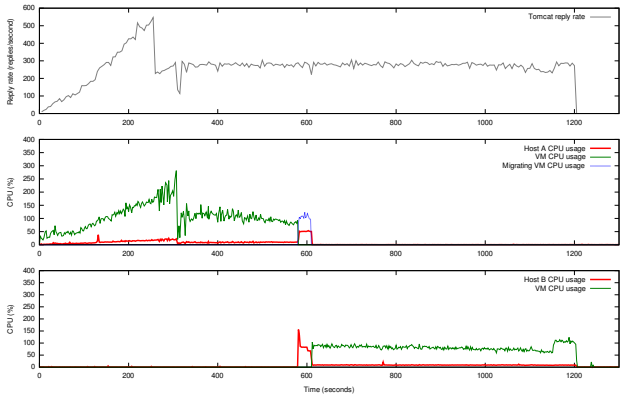

Figure 5. Migrated Tomcat execution

been migrated access remotely to their data. Nevertheless, this does not involve a performance problem, since we have shown that most applications (e.g. *memcoder*) exhibit similar performance when using local or remote disks. In fact, using remote disks will only have influence with applications with an extensive use of the disk (e.g. file copy).

5.4 VM migration performance

5.4.1 Overhead of task migration

First experiment measures the overhead introduced on the run time of a task which is migrated during its execution with respect to the local execution. Figure 4 shows the execution of a *mencoder* that is migrated from *Host A* to *Host B* when it has been executing for 300 seconds. The top graphic shows the CPU usage in both hosts of the task and the Xen Domain-0. The bottom graphic shows the network traffic of *Host A* and *Host B*. Notice that this graphic has been scaled in order to appreciate the transfer of data during the remote execution of the task, and not only the transfer during the migration.

In this figure, we can see how when the task is executing locally, there is not any network traffic. Nevertheless, when the VM is migrated, a big amount of data (around 1 GB), including the whole memory of the VM and a snapshot of the system, is transferred to the other host. In this phase, the CPU usage of the Xen Domain-0 is noticeable on both hosts, since it is working for migrating the VM. Finally, when the task is executing remotely, this is reflected in the network traffic.

	run time	expected time
Host A to B	611”	610.67”
Host B to A	613”	610.25”

Table 3. Migrated memcoder run time

This experiment provides also information about the total execution time of a task which has been migrated during its execution. This is important for measuring the migration overhead. Table 3 shows the execution time of *memcoder* if it is migrated at 300” from *Host A* to *B* and from *Host B* to *A*. In addition, the table also shows the expected execution time in these two situations, which has been calculated from the values in Table 2. Notice that the overhead introduced when migrating an application is negligible.

5.4.2 Accessibility of a migrated task

Another issue is the accessibility of a task when it is being migrated. This is especially important in tasks such as a web server attending clients’ requests. Next experiment evaluates the accessibility of such a task when it is being migrated. Figure 5 displays the execution of a Tomcat server stressed by several clients, showing its CPU usage and the Domain-0 CPU usage in both *Host A* (middle graphic) and *Host B* (bottom graphic). As expected, the Domain-0 CPU usage is only noticeable when the application is being migrated. In addition, the top graphic of Figure 5 shows the reply rate of the server (i.e. the throughput), demonstrating that it is always available to the clients that try to access the web page. Notice that the reply rate is maintained during the whole execution of the server including the migration phase, obtaining a 100% uptime in the service.

6 Related Work

Introducing virtualization for abstracting the nodes of a service provider, allocating tasks in a VM and consolidating and isolating them in the same physical machine has been widely investigated during the last years. Most of the related works include some solution to the data management problem, since this is one of the main challenges in VSPs. For instance, the In-VIGO project [3] and VMPlant [8] use

a simple local file system which only allows executing tasks in a single node without task migration.

In order to allow task migration, some other works send the required data among the different nodes. An example of this is VIOLIN [11], which uses diff files for transferring disk images in order to reduce the amount of sent data. Globus Virtual Workspaces [7] uses a global shared file system among all the nodes to store the VM images. In addition, in the same way than our proposal, this work can store VM images and data in the repository for future usage. SoftUDC [6] shares data between nodes by providing a smart storage virtualization service that allows any machine accessing any data in the pool. However, VM migration between nodes implies moving all required data by the VM from a node to another. In order to prevent the overhead resulting of sending the images to each node, [12] proposes a model for provisioning VM images that contains an on-demand environment efficiently. This model takes into account the overhead resulting from instantiating a remote virtual resource and introduces a method for efficiently manage virtual data transfer in order to reduce the overhead.

Another issue in virtualized environments is how to provide the applications with their required input data. Most of the previously described works (and others such as SODA [5]) delegate to the user the responsibility of putting this data inside the VM before the task starts executing. Our system simplifies this process by decoupling the user and the VM on which the task runs. The user just uploads the data in a data repository and the system will make it accessible to the VM.

7 Conclusions

This paper has presented a data management system for virtualized service providers that allows taking full advantage of the data-related virtualization capabilities. The proposed solution provides a global repository where clients can upload the input files needed by the tasks and retrieve the output files. In addition, the provider implements a distributed file system (using NFS) in which each node can access its own local disk and the disk of the other nodes.

Our evaluation demonstrates that nodes can create VMs and execute tasks efficiently using the local disk. Furthermore, tasks can be migrated with minimum overhead, since it is not necessary to transfer the VM image, while maintaining their accessibility during the whole process. In addition, migrated tasks can access remotely their required data via NFS. As a rule of thumb, this is done without noticeable performance penalty. Only applications with intensive disk usage (e.g. file copy) have a slight performance loss.

Probably, the weakest link in our system relates with the security, since providing data to an application and retrieving results is currently done with a simple check about the owner. For this reason, our future work consists of providing a most secure access to the data. In addition, we will perform extensive experimentation of the whole virtualized service provider using real task workloads.

8 Acknowledgments

This work is supported by the Ministry of Science and Technology of Spain under contract TIN2007-60625 and the European Commission under FP6 IST contract 034556 (BREIN).

References

[1] Apache Tomcat. http://tomcat.apache.org.

[2] Xen. http://www.xensource.com.

[3] S. Adabala, V. Chadha, P. Chawla, R. Figueiredo, J. Fortes, I. Krsul, A. Matsunaga, M. Tsugawa, J. Zhang, M. Zhao, L. Zhu, and X. Zhu. From virtualized Resources to Virtual Computing Grids: the In-VIGO system. *Future Generation Computing Systems*, 21(6):896–909, 2005.

[4] J. Ejarque, M. de Palol, I. Goiri, F. Juli, J. Guitart, R. Badia, and J. Torres. SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers. In *4th IEEE International Conference on e-Science (e-Science 2008)*, Indianapolis, USA, December 7–12 2008.

[5] X. Jiang and D. Xu. SODA: a Service-on-demand Architecture for Application Service Hosting Utility Platforms. In *12th IEEE International Symposium on High Performance Distributed Computing*, pages 174–183, Seattle, WA, USA, June 22–24 2003.

[6] M. Kallahalla, M. Uysal, R. Swaminathan, D. Lowell, M. Wray, T. Christian, N. Edwards, C. Dalton, and F. Gittler. SoftUDC: a Software-based Data Center for Utility Computing. *Computer*, 37(11):38–46, 2004.

[7] K. Keahey, I. Foster, T. Freeman, X. Zhang, and D. Galron. Virtual Workspaces in the Grid. *Lecture Notes in Computer Science*, 3648:421–431, 2005.

[8] I. Krsul, A. Ganguly, J. Zhang, J. A. B. Fortes, and R. J. Figueiredo. VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing. In *ACM/IEEE Conference on Supercomputing*, Washington, USA, November 06–12 2004.

[9] mencoder. http://www.mplayerhq.hu.

[10] RUBiS: Rice University Bidding System. http://rubis.objectweb.org/.

[11] P. Ruth, P. McGachey, and D. Xu. VioCluster: Virtualization for Dynamic Computational Domains. In *2005 IEEE International Conference on Cluster Computing*, page 7, Boston, MA, USA, September 27–30 2005.

[12] B. Sotomayor. A Resource Management Model for VM-based Virtual Workspaces (MSc Thesis). *University of Chicago, Department of Computer Science*, 2007.