Designing Virtualization-Based Infrastructure for Providing On-demand HPC Software-based Service

Rodrigue Chakode, Jean-François Méhaut

Designing Virtualization-Based Infrastructure for Providing On-demand HPC Software-based Service

Rodrigue Chakode, Jean-François Méhaut
INRIA Mescal Research Project - LIG Laboratory
University of Grenoble - France
Email: {Rodrigue.Chakode, Jean-Francois.Mehaut}@imag.fr

***Abstract*—The emerging of Internet-based computing, namely cloud computing, has increased the possibility of sharing remote resource. On the other hand, while the usefulness and how to use grid computing for sharing hardware resource have been well studied, such studies do not seem to be available concerning software. We propose in this paper an on-demand service model to share software within grids. Targeting performance, flexibility and resource use efficiency, we discuss virtualization-enabled features to suggest an infrastructure model, and an implementation approach based on service execution within virtual machines (VM). The study show that, in spite of the potential performance overhead, on which conventional wisdom has based its unsuitability for HPC, virtualization can enable relevant features concerning on-demand cloud-like service. The possibility to build flexible and highly-reconfigurable infrastructure, and the possibility to easily achieve efficiency-driven resource scheduling are examples. Based on an OpenNebula infrastructure along with Xen as VM Monitor backend, we implement a prototype, and carry out experiments concerning performance using the parsec benchmark. Our results show that, with suitable tuning, VM can achieve near native performance, even when applications run onto single node-hosted concurrent VMs.**

***Keywords*-Cloud Computing; On-demand Scheduling; Performance of Virtual Machines; Reconfigurability of Virtual Clusters; Software-based Service;**

I. Introduction

In grid virtual-organization, members usually share resource among them. Such resource can be hardware resource, or software. While how to share hardware resource has been well studied, this does not seem to be the case concerning software. The emerging of cloud computing has enabled the possibility of remotely accessing software as a service. This approach would be suitable to share specialized software within a grid. The service can be accessed following a subscription, and/or an on-demand model, while the latter would be suitable for service providers. Indeed, they would like having an online control on the access and the service use. In addition, the use can be charged to users. In this case, it could be interesting to automatically adapt the charging to service requirements, and the resource availability.

Virtualization were originally led by the aim of consolidating servers within data centers. Its main drawback has been performance overhead, especially caused by virtual machine management process, and mostly by I/O operations. Naturally, this performance overhead has been a constraint for the using of virtualization-based solutions within an HPC context. Due to conventional wisdom, the constraint still persists in spite of the work done these last years to reduce this overhead. Virtualization enables relevant features. Performance isolation among virtual machines (VM), live migration of VM, and the possibility to suspend and resume VM are examples. They enable node consolidation, preemption and process migration, which can be useful for achieving load balancing and/or energy-driven scheduling, and for dynamically adapting the infrastructure configuration. Besides, VMs can be automatically and easily tailored to fill the needs of applications.

In this paper, we discuss an on-demand service model for sharing software within grid virtual organization. Targeting performance, flexibility and resource use efficiency, we discuss features enabled by the virtualization to suggest an infrastructure model, and an implementation approach based on execution within virtual machine. In order to minimizing potential performance overhead due to virtualization, our design relies on two main concepts. A network-based application repository called *High-Speed Network Data Repository* (HSNDR), and *Smart Virtual Machines* (SVM). An HSNDR is repository through which applications within VMs perform I/O, while a SVM is a lightweight VM capable of automatically building service execution environment (along with an HSNDR), and scheduling the execution. We have implemented a prototype over an OpenNebula infrastructure along with Xen as VM Monitor backend. From this prototype, we carry out experiments concerning performance using parsec benchmark. Basing on several VM hardware configurations (number of cores, memory size, I/O devices, etc.), we particularly investigate the performance of concurrent VMs-based execution.

We will discuss the suitability of an on-demand service model for sharing software (Section II). We will then design an infrastructure model (Section III), followed by a prototype implementation (Section IV). From which, we evaluate the performance of applications running onto SVMs (Section V). Finally, we will present our conclusion and future works.

II. Sharing High Performance Software with On-demand Access

The usefulness of grid computing to aggregate and share computional resource, including hardware, as well as software has been well discussed [1], [2], [3]. However, almost of these studies target the sharing of hardware resource. We propose here an approach for sharing software.

A. On-demand Service Model for Sharing Software

A simple way to share software can consist in distributing software binaries. However, this process could be an heavy task since it has to be repeated at each software release. Likewise, provider would like preserving software integrity. Besises, specialized architecture (such as supercomputer, or computers with specialized chips) could be required to run software. It would be suitable in this case to provide the software as a on-demand remote service [4].

In such a case, service consists in processing user-provided data. The processing is scheduled onto the provider's infrastructure. An on-demand model requires a prompt service activation after user request. The service use can be charged to users. It has been showed that incentives for sharing resource within grid rise when resource providers can derive earnings [18], [19]. To be conformed with modern grid infrastructure, the service can be requested through web service, thereby being handleable by widespread used grid scheduling tools, such as WSGRMS [5]. Since the service providing relies on the provider's infrastructure, resource management can raise tricky issues as introduced in [1].

B. Resource Management

Several situations can happen when schedule the user service requests. The arrival of simultaneous requests which can not be scheduled at once. The arrival of requests when resources are not longer available. Assuming that the service use is charged, users need to be guaranteed with a well-defined service quality level, which can be defined in term of service delivery time, service cost, etc. We assume therefore that the service provider's jobs have less prioritized than user's jobs. Consequently, provider's jobs should be only scheduled onto idle resources, and have to be suspended when users' requests arrive (best effort scheduling). It would be suitable to avoid the wasting of already-done computation, by implementing preemption and process migration for instance.

Likewise, it is usual that computational power delivered by a single node in today HPC clusters over-passes the power required by a single application instance, according to data and parallelism level. In order to maximizing node throughput, consolidation should be enabled in order to allow the execution of several instances onto a same node, while guaranteeing isolation among the instances. Concerning software-based service providing, this consolidation can assist to optimizing the number of handleable requests. In order to be efficient, the mechanisms of suspending and restarting applications, as well as the mechanisms of consolidating nodes have to be easily achievable, and enable easy automation (easy reconfigurability). What does not seem possible with the existing tools within HPC environments.

1) Preemption and Process Migration in HPC context: In order to schedule prioritized application, it can be necessary to implement interrupt other applications. To avoid the restarting of the latter at their beginnings, preemption and process migration tools, such as BLCR [6], Condor [7], Libckpt [8], and cocheck [9], enable the suspension and the restarting of sequential, as well as distributed applications. Nevertheless, the correctness of apllications can not be guaranteed when the associated processes has open pipes, open files, and/or open sockets. In addition, these tools require the gathering of low-level information, such as process identifier, process session identifier, and/or process group identifier.

2) Consolidation Tools within HPC Context: Computing nodes in most of today clusters have more than four CPU/cores along with a large amount of memory (can be greater than 64GB). As previously mentioned, their delivered computational powers often overpass the requirement of a single instance of application. Consolidation tools, such as Linux Control Group (cgroups) [10] along with cpusets, or Solaris Containers [11] enable the correct running of several application instances onto a single node, thereby optimizing throughput and resource use. For safety and coherency purposes, these instances are mutually isolated. That is to say that each instance has its dedicated resource frame, and can not access to resource located beyond its frame. Although the aforementioned tools preserve isolation, they require low level tuning of operating system, thereby being suitable for permanent and/or long-term configuration.

C. Towards an Infrastructure for Sharing Software

In all, existing checkpointing and consolidation tools seem to be not suitable for designing highly configurable environment. Such features can be enable by the virtualization[12] in spite of its potential performance overhead. This point of view is consolidated by researchers in [13], [14]. While consolidation can be easily achieved by running applications within virtual machines (VM), checkpointing is also achievable easily (typically, by saving the entire VM state). Moreover, VMs can be migrated easily towards less or more computing nodes in order to achieve load-balancing and/or energy-efficient computing, which are relevant requirements to modern data centers.

Assuming that VMs will be stood permanently, a VM-based infrastructure will look like transitional cluster. Resource scheduling can thus be achieved through a traditional resource manager such as OAR [15] and PBS [16]. In such a case, the VMs' configurations (hardware and/or, software) have to be known in advance. However, using on-demand model, VM configurations can vary according to execution

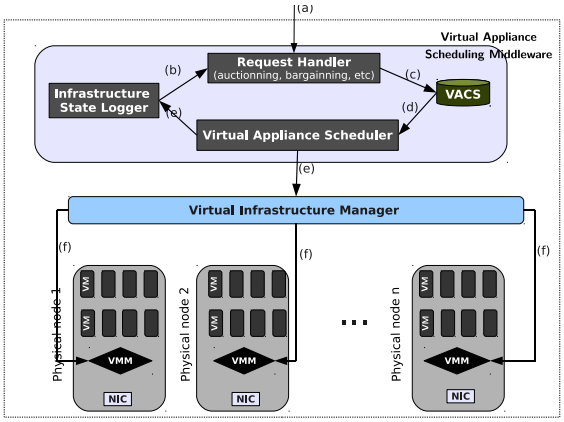

Figure 1. A middleware architecture and workflow description for providing software-based service

data, for instance. Furthermore, as previously discussed in [17], scheduling resource within such a context needs to be flexible, and enables reconfigurability for guaranteeing the efficiency of resource use. This implies challenge to dynamically configure the suitable VMs for running services, and to dynamically schedule these VMs onto physical nodes.

III. Infrastructure Model for Scheduling Parallel Service On-demand

Functionally, the service execution is scheduled onto VM which is then scheduled onto physical node. Contrary to traditional resource managers, the cluster frontend does not have to be directly accessed by users. Direct-access to cluster can be unsafe, while the resource use can be unfair. Indeed, several accesses have to be granted to users whom can be malicious and/or selfish users. Deal with such users can become increasingly complex according to their number. Therefore, we assume that service is requested through web service interface along with execution parameters, that service provider enables mechanisms allowing users to upload and/or download input and output data, where required.

Our design focuses three main points, (i) requests handling, (i) resource scheduling, (iii) the minimization of the potential overhead due to the virtualization.

A. Handling User Request

One of the most important things to deal with is to handle user requests. We assume that software providers charge the use of software. The service access can be auctioned or bargained through a market-oriented grid system, such as proposed in [20], integrated within the *Request Handler* (RH) (see Figure 1), after a service request (a). This bargaining or auctioning can take the state of available resource (provided by the *Infrastructure State Logger* (b)) into account. Successful bargaining implies that there are sufficient resources to schedule the service. This process constitutes the *on-demand access-model* to service. For scheduling purpose, the RH retrieves parameters from service request to build the configurations of the virtual appliance (VA) associated to the service, and stores them into the *Virtual Appliance Configuration Store* (VACS) (c).

B. Request Scheduling

After handling request, two things have to be achieved for scheduling the service: (i) create VAs with suitable configuration and data, (ii) schedule these VAs onto physical nodes. In order to guarantee an efficient use of the whole system, this creation has to be achieved easily and fastly. It has therefore to be achieved automatically and dynamically, thereby avoiding human interactions, where possible. For scheduling these VAs, the *Virtual Appliance Scheduler* (VAS) relies on a virtual infrastructure (VI) manager [21] (f). After successfully scheduling service request, the state of available resources is updated into the *Infrastructure State Logger* (e).

Basically, avoiding human interactions assumes however mechanisms for automatically providing input to applications within VMs, for automatically retrieving output, while preserving performance.

C. Dealing With Performance

Potentially, such an infrastructure can have performance issues, mainly consisting of time to setup VM, overhead due to VM Monitor (VMM), and overhead due to I/O operations performed within VMs.

1) Dealing with I/O: Most of existing data centers use manually-built VM software stacks also known as virtual appliances (VA). VMs disks are often virtual block devices, including raw hard-drive partitions, LVM partitions, and file-backed VBD. The last is most used since it enables easy customization. Indeed, it is easily duplicable using basic file copy tools, and can also be mounted as directory within file system. Its content can thus be modified easily. Such a task can be automated when configuration and data are known and available. According to the architecture on Figure 1, such information can be retrieve from the VACS database. However, although file-backed VBDs are suitable for automating the setting of VAs, they can have drawbacks. The main problem is performance overhead [22], more significant when VM host data-intensive application. Furthermore, VM setup time grows according to application data size. Without choosing suitable disk size, it can be filled during execution, thereby causing crash. Without knowledge on the size of data, it seems difficult to avoid such a crash, unless to always use large disks, and often waste unused space. Furthermore, copying data to and/or from VM requires disk- and network accesses, which consume time. The scheduling of distributed applications can require multiple copies of same data within VMs depending to the number of coopering VMs.

To avoid such issues, we propose to store application data into a network-based data repository, hereafter referred by *Application Data Repository* (ADR). An ADR is linked to VM's file system as a network-attached storage. Although basically, this configuration avoids duplication, the performance of an ADR-based application also depends on the underlying network. In [23], researchers suggest that by performing I/O operations through high-speed networks such as InfiniBand and Giga Ethernet-based TCP, the performance of VM-based data-intensive applications can be significantly improved, while better performance are achieved using InfiniBand. Although this result was achieved with a specific visualization tool, it can be applied with other tools. Indeed, it basically relies on a generic mechanism, the Virtual Machine Monitor-Bypass or OS-Bypass mechanism. It can be implemented in several ways. PCI passthrough [24] which is implementation independent, is another example of the bypass mechanism implementation.

2) Setting Up Virtual Machine: Instead of packing all application data into VM image, we propose to use *Smart Virtual Appliances* (SVA). A SVA is a lightweight VM image along with minimal software stacks, capable of setting custom execution environment up, from configurations. Relying on an ADR Repository, SVA's Init Scripts and configuration files are altered to achieve this customization at startup of SVM (Smart Virtual Machine). Such customization consists in configuring access to ADR, setting user and files accesses, scheduling commands, etc. By using SVA, multiple data copies can be avoided, thereby reducing time required to set VM up. A SVM is configured so to automatically shuts down as soon as scheduled tasks are ended, thereby freeing physical node.

IV. Implementation

The original context of this work is CILOE[1], an industrial project supported by the business cluster Minalogic[2]. The project aims at fostering access to HPC technologies, including methodologies, softwares, and security mechanisms, to small and medium enterprises. It is a pilot project aiming at developing open components for providing on demand software-based service over clusters of multi-core machines. CILOE involves nine partners (four small and medium software editors, three research institutions, one computer manufacturer, and one consulting company). The current implementation is a footstep following a theoretical-preliminary study [17].

A. Break in Virtualization Technologies for Making Choices

Virtualization is an essential solution for allocating computational resources tailored to the needs of an application. Virtual machines (VMs) enable the specialization of operating systems towards particular tasks, with hardware resources safely and transparently multiplexed by the VM monitor (VMM). Virtualization allows the decoupling of the physical computational infrastructure management from the management of the user-visible virtual computing facility.

1) About Virtualization and Service-Oriented Computing: Virtualization has fostered a new paradigm of computing, namely service computing which is also known as cloud computing [4]. Cloud services can be grouped in three categories, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). They are located at different levels in an infrastructural and service hierarchy as illustrated on Figure 2. At the physical node level (low level), VMMs including Xen [25], KVM [26], and VMware [27] enable features, consolidation, (live-)migration of VM, as well as the possibility of suspending and restarting VMs. As previously mentioned (first section), these features can be interesting within an HPC context, although virtualization suffers of performance overhead. At the infrastructure level (middle level), virtual infrastructure managers, such as OpenNebula [28], Eucalyptus [28], Globus Nimbus [29], Amazon EC2 [30], and VMware vSphere rely on VMMs as backend to provide virtual clusters (cluster of network-interconnected VMs). According to VI managers, several algorithms and/or policies can be provided to schedule VMs onto physical nodes (immediate provisioning, best-effort, advance reservation, etc). Some of VI managers (e.g. OpenNebula) can enable the possibility of using custom scheduling algorithms [31]. IaaS clouds rely basically on VI managers. At the service level (top level), tools can be built over VI managers to provide PaaS, as well as SaaS clouds.

2) Technology Choices: Formally, a *Virtual Appliance Scheduling Middleware* (VASM) (see Figure 1) encapsulates three layers. The software service layer relies on the platform layer, which is deployed on an infrastructure layer. Concerning implementation, we would like building open framework. It would be therefore based on open components. VM technology choices are restricted to non-proprietary solutions. Other selection criteria are respectively performance, useful features, and flexibility. At the node level, there are studies concerning VMMs performance. [25], and [32] move Xen and KVM in forward, while none of them do not really differentiate KVM and Xen concerning all kind of applications. [33] suggests however that the Xen paravirtualization approach versus the KVM full virtualization approach, allow the building of lightweight VM images (without hardware drivers). Paravirtualized VMs seem to be more customizable than fully virtualized ones. At the infrastructure level, [31] provides a survey concerning existing VI managers. OpenNebula differentiates itself by enabling relevant features, such as an easy customization of scheduling algorithm, and the possibility of contextualizing VMs.

[1]http://ciloe.minalogic.net

[2]http://www.minalogic.com. This work is supported by Minalogic through the project CILOE. We express our thankfulness to the CILOE partners for their collaborations.

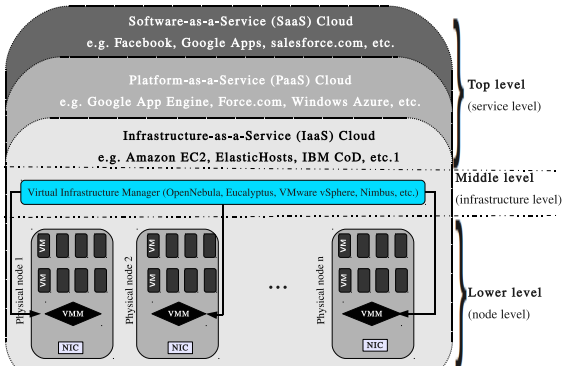

Figure 2. Hierarchy of Virtualization-based Services

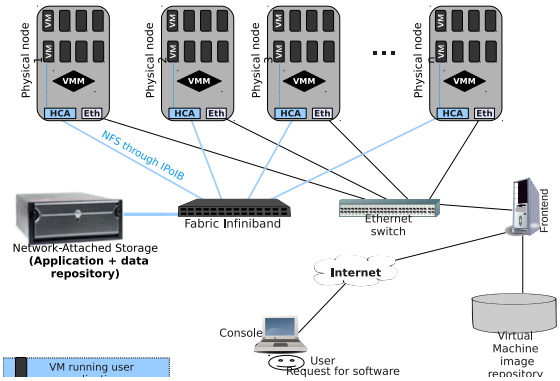

Figure 3. A physical architecture for running software-based service

```
SERVICE_ALIAS_LIST = "service1   service2   service3f"
SERVICE_APPLICATION_PATH = "/app1_repository/bin/app1   /app2_repository/bin/app2/app2_repository/bin/app3"
APPLICATION_USER_MAP = "app1_adm app2_adm       app3_adm"
ADR_REPOSITORY_MAP= "/app1_repository          /app1_repository          /app1_repository"
ADR_SERVER = server_ip | server_name
VACS_DIR = /var/vasm/vacs
```

Figure 4. Overview of Service Configuration File

B. Implementation of the VASM Components

The implementation of the VASM is at prototype stage. This prototype does not implements the negotiation within the RH, yet ongoing. It assumes that resources are available for scheduling service as requested. Otherwise, if a VM associated to service request is scheduled when there are not sufficient resources available on physical node, it will fail at startup. This behavior is inherent to the backend VMM, namely Xen. The prototype is implemented as a command line program allowing arguments. It already enables the handling and the scheduling of requests from command line. The VASM arguments consist of a service name (program alias) along with execution parameters which consist of program arguments, and hardware requirements (number of cores, memory size). Further details about the implementation of each component are given hereafter.

1) Physical Architecture: For the implementation, we consider a typical cluster with a frontend and computing nodes interconnected through a network as illustrated on Figure 3. Such a network, usually a Ethernet network, is use for operating tasks. In addition, computing nodes communicate through an HPC-dedicated high speed network, for instance an InfiniBand (IB) fabric. In such a configuration, the VASM stack is hosted onto the frontend, from which service requests are submitted. The Ethernet network is used for scheduling and other miscellaneous tasks, such the migration of VMs. Images of VMs are stored into a Ethernet-based NFS Repository, enabling live migration, while centralizing images management tasks.

2) ADR Repository: It is implemented as a NFS file system over IB. However, instead of implementing IB drivers within VMs, the support of IPoIB (IP over IB) is enabled within privileged domains (Dom0). Indeed, their kernels are compiled by enabling IB support, and IB drivers including IPoIB stack are installed from a OFED stack [34]. On Figure 3, bold lines are associated to the IPoIB network, which are dedicated to computation. Each Dom0 is configured as an ADR router for its guest domains (DomU).

3) Request Handler: Request Handling consists in parsing request command line, and automatically building the associated virtual appliances (VA). This task is achieved from configuration files, and relies on template-based contextualization feature enabled by OpenNebula. For understanding purpose, an OpenNebula VM template includes various directives. Some of these directives are used to specify hardware configuration (number of CPU/cores, memory size), as well as *context files*. Such files are archived within an ISO image, and then attached into VM's file system as a block device.

Our implementation defines a VA configuration as a set of files consisting of a VM template, context scripts (to automatically schedule service execution onto VM), as well as scripts (to schedule VM onto computing node). For creating this configuration, a generic template tailored for this purpose, is duplicated and altered basing on request parameters. Scripts necessary for scheduling service execution, as well as other contextualization files are generated from request parameters and the services configuration file. As illustrated on Figure 4, this file lists all available services along with parameters necessary to setting execution environment. For instance, the configuration provides for each service, directives for indicating Executable Files, an ADR Repository, and an User with suitable rights on data. All generated files are stored into the VACS directory, whose its path is also specified in the services configuration file.

4) Request Scheduler: As mentioned in the previous section, scheduling virtual machine onto physical node is a fairly easy task consisting in running the RH-generated

scripts. Other tasks are scheduled within smart virtual appliances.

5) Smart Virtual Appliance: It is implemented from a basic image of VM, whose Init Scripts are altered to achieve the following algorithm when the associated virtual machine starts. (i) Mount the context files ISO image into a temporary directory, (ii) Move to this directory, (iii) Read the virtual appliance-configuration file. (iv) Configure access to the associated ADR Repository and setting file accesses, (v) Switch to suitable User, (vi) Schedule service execution, (vii) Waiting for the service execution end, (viii) Save the execution exit status into the ADR, (ix) Unconf access to ADR, (x) Shut down virtual machine.

V. Evaluation

In this section we evaluate the performance of VASM's Components in order to validate our design. We evaluate the performance of HSNDR-based SVMs, focusing first on time required to configure and to start up virtual machines according to application data size. We compare these time with those required when application data (binaries and data) are copied within VMs. To evaluation the performance of applications running onto HSNDR-based SVMs, we use a representative benchmark concerning emerging applications, namely [35]. Indeed, benchmarks, such as NPB [36], used in prior works [25], [23], [32] seem to be old regarding the new generations of software. Parsec focuses on emerging workloads and was designed to be representative of next-generation shared-memory programs for multi-core chips. This evaluation brings out three other contributions. We evaluate the overhead due to virtualization when applications are running onto multicore VMs. We investigate the performance of applications running onto single node-hosted concurrent VMs. We finally evaluate the impact of I/O operations (performed within VMs) on virtual machine monitor.

A. Experimental Environment

We use Bull R422-E1 servers as physical computing nodes. Each node consists of 8 GB of memory, 2 quad cores Intel Xeon CPU (2.5Ghz, 6MB of cache size), 1 SATA disk, and 2 network interface cards (NIC). The first NIC is connected to a Gigabit Ethernet network, and the other to an InfiniBand fabric through 20G HCA (Mellanox ConnectX IB 4X DDR MT26418). The VASM middleware is deployed over OpenNebula 1.4, while the HSNDR and the VM image repositories are configured as NFS v3 file systems over SATA disks. The NFS servers are managed by native Linux, and are respectively accessible through the IPoIB and the Ethernet network. Each physical node includes Xen 3.4.2 and OFED 1.4.2, and is configured to work as OpenNebula node. DomO and DomU kernels was compiled separately.

B. Performance and Benefits of HSNDR-based SVMs

We evaluate the three parsec's OpenMP applications, Bodytrack (image tracking), Blackscholes (option simulation), and Freqmine (word frequence mining). While the weight of disk I/O is fairly high in the first, it is slight in the second, and low in the third. Before evaluating their performances, we evaluate the raw performance of I/O device using the iozone benchmark [37]. Such an evaluation aims at validating the choice of I/O device.

Running iozone, the maximum of measured read speed over VBD or HSNDR is about 45% less than the maximum over native SATA disk (Figure 5-(b)). However, Due to NFS write cache, the maximum of measured write speed over HSNDR is about 192% greater than the maximum over native SATA or VBD (c). Concerning execution time, while their performances are slightly equal when running single instance (first cluster on Figure 5-(c)), HSNDR shows well scalability than VBD devices. This result can be justified by looking the behavior of Dom0s when one of their hosted VMs computes iozone. The response time of a Dom0 hosting VBD-based DomUs is unpredictable. As shown on Figure 7, its often behaves as its load has been multiplied by about 150. These response time consist of the time to compute remote *ls* commands onto each Dom0, using *ssh* from another node. Concurrent instances over raw SATA cause crash along with SIGFAULT signal.

Benchmarks from parsec applications show that, isolated mono- and multi-core VMs achieve near native performance (less than 3% of overhead). However, significant overhead can appear with single node-hosted concurrent VMs-based executions (over 50% with 8 concurrent 1-threaded instances of Bodytrack), see Figure 6. Due to swapping, concurrent instances of fremine achieve poor performance onto native system (over 700% of overhead), while instances based on concurrent VMs perform near native performance (0.01% of overhead). It can be observed that performance overhead varies considerably according to the number of concurrent DomUs (inducing high a load on Dom0), and the weight of I/O operations within applications (inducing high load on the underlying disk). This overhead can be significantly reduced, for instance, by tuning the underlying network and/or disks, or by choosing a suitable number of concurrent VMs along with suitable hardware configuration, etc.

Concerning time to get VMs ready to compute, HSNDR-based SVMs require a low and constant time. Using 500MB image, about 18 seconds are required for duplication, and setup (see on Figure 8). This represents less than 10% of compute time for applications requiring more than 3 minutes of execution time. Contrariwise, packed data-based VMs require linear time according to data size. Up to 10 minutes are required for packing 10GB of data. Ten Gigabytes is not surrealist. For example, JivaroD [38], an application within the CILOE context can process more than 15GB of input.

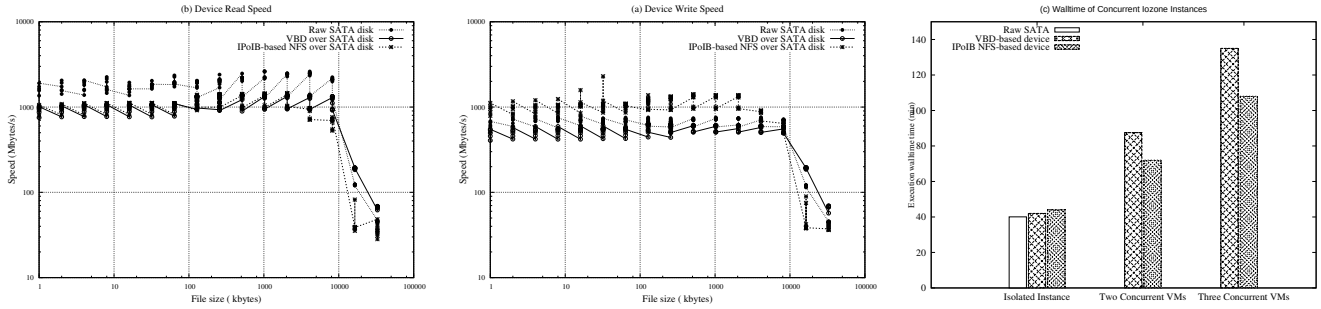

Figure 5. Device Bandwidth and Scalability

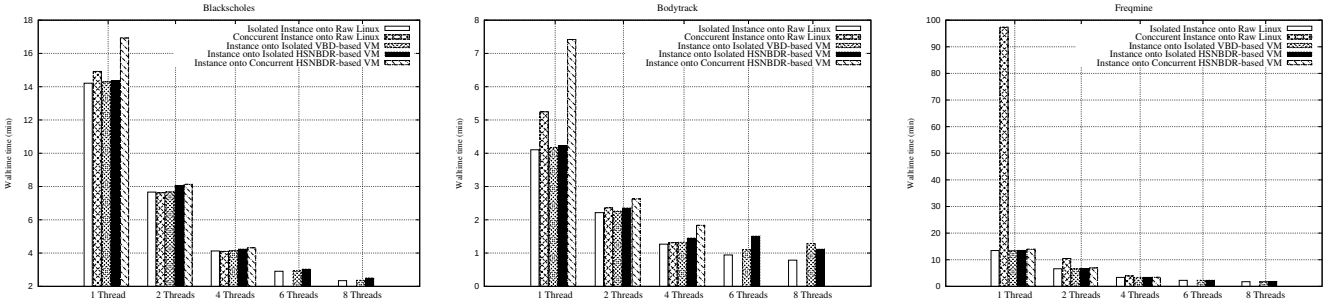

Figure 6. Performance of HSNDR-based VMs running Parsec Applications

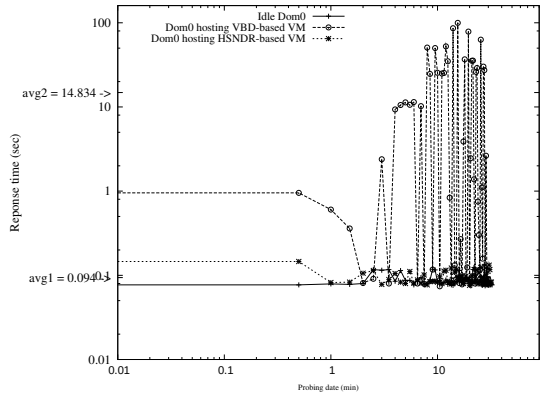

Figure 7. Dom0 Response time when I/O are performed within DomU

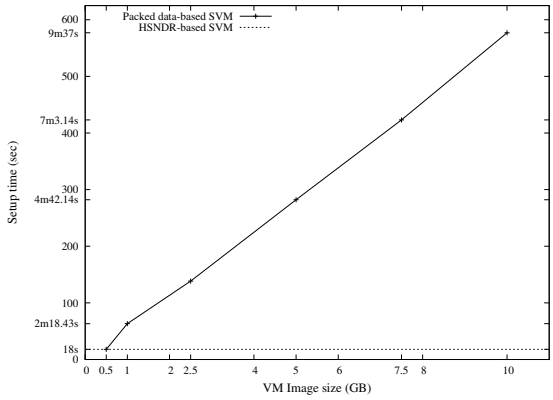

Figure 8. VM setup time. SVM achieves low and constant setup time

VI. Conclusion

We have proposed an infrastructure model for implementing on-demand software-based service, aiming at enabling software sharing within grids. Targeting performance, flexibility, and resource use efficiency, the model relies on *smart virtual machines* for on-the-fly scheduling service execution onto virtual machines (scheduling on-demand), and an *high-speed network-based data repository* through which applications within virtual machine (VM) perform I/O. Relying on an OpenNebula infrastructure along with Xen as VM Monitor backend, we implemented a prototype, from which virtualization overhead have been evaluated using parsec-benchmark's applications. Our results show that VMs can achieve near native performance, while otherwise, the overhead can be significantly reduced with suitable tuning. Besides, this model can also be suitable for enabling the implementation of highly reconfigurable on-demand PaaS Clouds. As future work, we plan to complete the implementation, including the components for load balancing (considering energy consummation and/or server load), and invoicing.

References

[1] C. Kesselman and I. Foster, *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[2] G. Fox, A. J.G. Hey, and F. Berman, *Grid Computing: Making The Global Infrastructure a Reality*, pp. 9–47. Wiley, 2003.

[3] S. Mukherjee, J. Mustafi, and A. Chaudhuri, *Grid Computing: The Future of Distributed Computing for High Performance Scientific and Business Applications*, vol. 2571/2002. Springer.

[4] L. M. Vaquero, L. Rodero-M., J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2009.

[5] W. Guan and G. Sun, "Web service grid resource management system," in *CSIE '09: Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering*, pp. 424–427, IEEE Computer Society, 2009.

[6] J. Duell, "The design and implementation of Berkeley Lab's linux Checkpoint/Restart," tech. rep., 2003.

[7] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System," Tech. Rep. UW-CS-TR-1346, University of Wisconsin - Madison Computer Sciences Department, April 1997.

[8] J. Plank, J. S. Plank, M. Beck, M. Beck, G. Kingsley, G. Kingsley, K. Li, and K. Li, "Libckpt: Transparent checkpointing under unix," pp. 213–223, 1995.

[9] G. Stellner, "Cocheck: Checkpointing and process migration for mpi," in *in Proceedings of the 10th International Parallel Processing Symposium (IPPS'96*, pp. 526–531, 1996.

[10] http://www.kernel.org/doc/Documentation/cgroups/.

[11] "System Administration Guide: Solaris Containers-Resource Management and Solaris Zones." http://dlc.sun.com/pdf/817-1592/817-1592.pdf.

[12] http://www.vmware.com/pdf/virtualization.pdf.

[13] M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis, "Virtualization for high-performance computing," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 2, pp. 8–11, 2006.

[14] A. B. Nagarajan, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive fault tolerance for HPC with Xen virtualization," in *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pp. 23–32, ACM, 2007.

[15] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mouni, P. Neyron, and O. Richard, "A batch scheduler with high level components," in *Cluster computing and Grid 2005 (CCGrid05)*, 2005.

[16] V. M. H. Feng and D. Rubenstein, "PBS: a unified priority-based scheduler," in *SIGMETRICS*, pp. 203–214, 2007.

[17] R. Chakode, J.-F. Méhaut, and F. Charlet, "High Performance Computing on Demand: Sharing and Mutualization of Clusters," in *AINA '10: Proceedings of the 24th IEEE International conference on Advanced Information Networking and Applications*, pp. 126–133, 2010.

[18] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic Models for Resource Management and Scheduling in Grid Computing," *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 2002.

[19] "Grid Economics and Business Models," in *GECON* (J. Altmann and D. Veit, eds.), vol. 4685 of *Lecture Notes in Computer Science*, Springer, 2007.

[20] P. Chacin, X. León, R. Brunner, F. Freitag, and L. Navarro, "Core services for grid markets," in *The CoreGRID Symposium (CGSYMP 2008)*, August 2008.

[21] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Computing*, vol. 13, pp. 14–22, 2009.

[22] A. Asosheh and M. H. Danesh, "Comparison of OS level and hypervisor server virtualization," in *Proceedings of the 8th conference on Systems theory and scientific computation*, pp. 241–246, World Scientific and Engineering Academy and Society (WSEAS), 2008.

[23] W. Yu and J. S. Vetter, "Xen-Based HPC: A Parallel I/O Perspective," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 154–161, 2008.

[24] "Linux virtualization and pci passthrough." http://www.ibm.-com/developerworks/linux/library/l-pci-passthrough/.

[25] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pp. 164–177, ACM, 2003.

[26] "Kernel based virtual machine." http://www.linux-kvm.org/.

[27] http://www.vmware.com/.

[28] http://www.eucalyptus.com/.

[29] http://www.nimbusproject.org/.

[30] http://aws.amazon.com/ec2/.

[31] B. Sotomayor, R. S. Montero, and I. Foster, "An Open Source Solution for Virtual Infrastructure Management in Private and Hybrid Clouds," *Preprint ANL/MCS-P1649-0709*, vol. 13, 2009.

[32] T. Deshane, Z. Shepherd, J. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao, "Quantitative comparison of xen and kvm," in *Xen summit*, USENIX association, June 2008.

[33] A. Whitaker, M. Shaw, and S. D. Gribble, "Denali: Lightweight Virtual Machines for Distributed and Networked Applications," in *In Proceedings of the USENIX Annual Technical Conference*, 2002.

[34] http://www.openfabrics.org/.

[35] http://parsec.cs.princeton.edu/.

[36] http://www.nas.nasa.gov/Resources/Software/npb.html.

[37] http://www.iozone.org/.

[38] "JivaroD Home." hhttp://www.edxact.com/prod_jivd.html.