A Multiple Dimension Slotting Approach for Virtualized Resource Management

Fernando Rodríguez, Felix Freitag, Leandro Navarro

Department of Computer Architecture
Technical University of Catalonia, Jordi Girona 1-3
08034 Barcelona, Spain
{frodrigu, felix, leandro}@ac.upc.edu

Abstract

Recent advances in virtualization technologies can be applied to different research areas, such as virtualized resource management in High Performance Computing (HPC) Grid. On the one hand, the HPC Grid hides to users the complexity of the virtual cluster management, but on the other hand, the Virtual Machines (VM) are managed in the execution domain of the node's resource manager. Hence, local decisions must be done to meet the quality of service required by each VM. Moreover, a user should be able to define a Service Level Agreement (SLA) with strong and soft requirements for a given resource (CPU, disk, network, etc) without detailed knowledge of the hardware configuration of each node. In this work, we present a fine-grain approach for virtualized resource management in HPC nodes. Our approach improves the resource usage in nodes enabled with virtualization by efficiently partitioning and fine-grain management through multiple dimension slotting. We show in preliminary results that the management of single physical resources gives accurate usage information to a local resource manager (LRM). The assignment of resources is improved by adding a virtual partitioning layer for the subsets of physical resources. Relocating and resizing assigned single resources in the LRM scope improves internal migration mechanisms that could potentially reduce the overhead of external migrations.

Categories and Subject Descriptors D.3.3 C.2.4 [**Distributed Systems**]: Distributed applications; C.4. [**Performance of Systems**]: Measurement techniques.

General Terms Management, Measurement, Performance, Design, Experimentation.

Keywords Local resource manager, Virtualization.

1. Introduction

Recent advances in virtualization technologies are lead by the efforts of projects like Xen [1][3], VMware[14], and VNET [13]. In virtualized nodes, the physical resources are multiplexed among Virtual Machines (VMs). The Virtual Machine Monitor (VMM) is the software that manages the physical node. Thus, the guest-OS running inside the VM receives a slice of the physical node. Research challenges for virtualization have being studied for HPC [11] and Grid [4].

Today, nodes that act as resource providers show heterogeneous hardware with more than one resource of each type (CPU, NICs attached to different networks, disks, etc). We can see this phenomenon as usual in these days with, for instance, laptops that are bundled with dual core processors and two NICs (for wireless and wired networks). For this reasons we claim that local resource managers should take advantage of a fine-grain approach to better manage every single piece of hardware that can be virtualized. Rather than multiplexing a physical resource, we should choose and assign subsets of single physical resources and assign them to one or more than one (by multiplexing) (VM).

Virtual machine monitors must offer primitives to assign single resources to each VM according to demand. These mechanisms must be exposed to standardized interfaces through a pinning framework for managing resources in VM environments. Similar to the pinning mechanism of Xen to assign processors in Symmetric Multiprocessing (SMP) architectures to VMs. Additionally, the mechanism should be extended to every type of resource that could be virtualized by VMM or other virtualization approach.

A coarse-grain resource management of virtualized resources could potentially have more drawbacks than benefits when sharing an under-utilized physical node between VMs. For instance, an inaccurate slotting mechanism could provide to upper layers incomplete or erroneous information resulting in a waste of resources. This could happen in nodes with SMP architectures, redundant NICs with different speed capacities (10Mb, 100Mb, 1Gb), and redundant storage capacity (IDE, SATA, SCSI, etc).

Another potential problem is the unbalance of the workload. In SMP nodes, for instance, Xen virtual machines are created in a round robin fashion assigning processors one by one. Thus, we could have applications with different behaviour (burst, batch, parallel, intensive, etc.) running in these VMs interfering with each others. One solution is to separate the virtual environments considering the application profiles and assign these profiles to different processors. The proposed fine-grain assignment mechanisms will help to balance the workload in VMs.

Moreover, we can take advance of the pinning mechanisms to reduce external migration of VMs. One way to do this is by relocating the assigned single physical resource and adjusting the percent of share (an example of this is the migration from one NIC of 100Mbs to 1Gbs). From now, we refer to this mechanism as internal migration.

In this work, we present multiple dimension slotting (MuDiS), an approach to improve the resource usage in nodes enabled with virtualization. Rather than managing local resources extending VMMs, our goal is to enable local resource managers to better manage single physical resources. We envision a flexible standard API to assign atomic physical resources entities to VMs.

MuDiS has mainly two benefits. First, it allows HPC or Grid Global Resource Manager (GRM) to make better scheduling decisions by receiving accurate information of the internal usage in

every resource provider. And second, the providers are able to maximize the use of resources in a node. The MuDiS approach will enable the implementation of autonomic or reconfiguration mechanisms to adapt physical hardware in a fine-grain fashion to different type of applications.

Our approach dynamically optimizes the workload balance on each resource. In consequence, we expect a reduction in the overhead caused by external migration mechanisms. A learning mechanism with heuristics could infer resource usage patterns of running virtual workspace. This will enhance the LRMs since profiles could be created which allow application specific resource assignment. Additionally, it could be achieved that the interference between running applications becomes reduced.

The rest of the paper is organized as follows. Section 2 reviews the related work. Section 3 discuss the MuDiS approach and outlines the prototype. Section 4 shows and compares the results of experiments with and without MuDiS. Finally, Section 5 presents our conclusions.

2. Related Work

In this section we present the work related to virtualization-enabled systems.

Lamia Youseff et al [16] investigate the efficiency of HPC applications in paravirtualized nodes. The experiments include a set of micro-benchmarks, macro-benchmarks, and real HPC applications. The results show that Xen paravirtualization system poses no statistically significant overhead over other OS configurations.

K. Keahey et al [8] introduce the concept virtual workspace (VW), an approach which aims the provisioning of a customized and controllable remote execution environment for Grid resource isolation purposes. The underlying technology that supports the prototype are virtual machines for the hardware virtualization (Xen and VMware) and Globus Toolkit (GT). The interactions of resource provisioning follow VW descriptions. The descriptions define the VM state (running, shutdown, paused, etc) and the main properties that a required VM must complain such RAM size, disk size, network, etc. Currently, there is not discussion about the performance impact of multiple VM running at the same time and the consequence when SLAs are not fulfilled.

Mark F. Mergen et al [11] discuss open issues of hardware virtualization for HPC environments such as the agreement on the appropriate minimal abstractions and services that a hypervisor should implement, and the selection of the appropriate core primitives and capabilities that a hypervisor has to support for HPC, among others research questions.

Nadyr Kiyanclar et al [9] present Maestro-VC, a set of system software which uses virtualization (Xen) to multiplex the resources of a computing cluster for client use. The provisioning of resources is achieved by a scheduling mechanism which is split into two levels. An upper level scheduler (GS) which manages VMs inside the virtual cluster, and an optimal low-level scheduler (LS) per VM. The purpose is to incorporate information exchange between virtualized and native environments to coordinate resource assignment. The LS, however, is an optional mechanism and if desired it must be explicitly supplied by the user.

Wei Huang et al [6] present a VM based framework for HPC. The I/O virtualization overhead is reduced with a technique called VMM-bypass; this technique allows time-critical I/O operations to be carried out directly in guest VMs without involvement of the VMM and/or a privileged VM. The results show that HPC applications can achieve almost the same performance as those running in a native, non-virtualized environment. However, the efficient management is an open issue that needs to be addressed.

David Irwin et al [7] present Shirako, a system for on-demand leasing of shared networked resources in federated clusters. Shirako uses an implementation of COD [2] for the cluster site manager component and SHARP [5] for the leasing mechanism. Leases are used as a mechanism for resource provisioning, thus intra LRM adaptation is not independent and only possible at long terms.

Our work is in the direction of Dongyan Xu et al [15]. The authors propose the support of autonomic adaptation of virtual distributed environments (VDE) with a prototype of adaptive VIOLIN [12] based on Xen 3.0. They address challenges of live adaptation mechanisms and adaptation decision making. Inter LRMs adaptation is based on live cross-domain migration capability, and intra LRM adaptation by adjusting resources shares of physical nodes according to VM usage. Our proposal differs in that it attempts to improve the intra LRM mechanisms; for instance, our approach uses low level management to assign physical processors to VMs.

3. MULTIPLE DIMENSION SLOTTING

The Multiple Dimension Slotting approach aims to enhance the multiplexing mechanisms offered by VMMs by managing resources in a fine-grain fashion. At the same time, it allows that LRMs to manage the pinning mechanisms according to user requirements.

3.1 MuDiS Management Component

The Multiple Dimension Slotting management component runs in the first domain, known as Domain0 (Dom0). Xen is a Type I VMM –also known as 'hypervisor'– that runs directly on top of the hardware and boots Dom0. This domain is a privileged host OS that has access to the real hardware. The MuDiS carries out the account usage of every single resource and groups single physical resources into subsets. These subsets (which can be dynamically changed) are seen by the LRM as pseudo-physical machines with minimal interfering I/O interruptions. Thus, we isolate physical resources with different characteristics that can be exploited by the LRM.

The subsets of resources can be assigned to VMs with different application profiles (e.g. CPU-intensive, Network-intensive, SMPs requirement, or I/O-intensive). This virtual division allows VMs behaving according to the limitations of the assigned resources, and hence the running applications.

Figure 1 show and example of the partitioning of physical resources. We can see in Figure 1a the partition in 3 subsets, each one with different characteristics. The subsets are exposed to the LRM and each one can be sliced with traditional approaches (e.g. VM1 and VM2 could have each one 50% of each resource of subset 1 respectively). The MuDiS management component maps the subset shares and assigns them to virtualized resources through the VMM. In Figure 1b we can see an example for a new configuration after time n. The observed applications behaviour could have shown that some resources were under- or over-utilized. Finally, the LRM regroups VMs to improve performance and load balancing.

The major overhead during the internal migration of VMs occurs when one VM needs to be moved to another physical disk. The reassignment of processors or migration to better network links is expected to be done in this context in considerably less time. Since memory resource has homogeneous characteristics, the management is done by increasing or decreasing the amount assigned.

With our approach we gain in performance by adding an extra layer of isolation. The services to manage and monitor subsets of

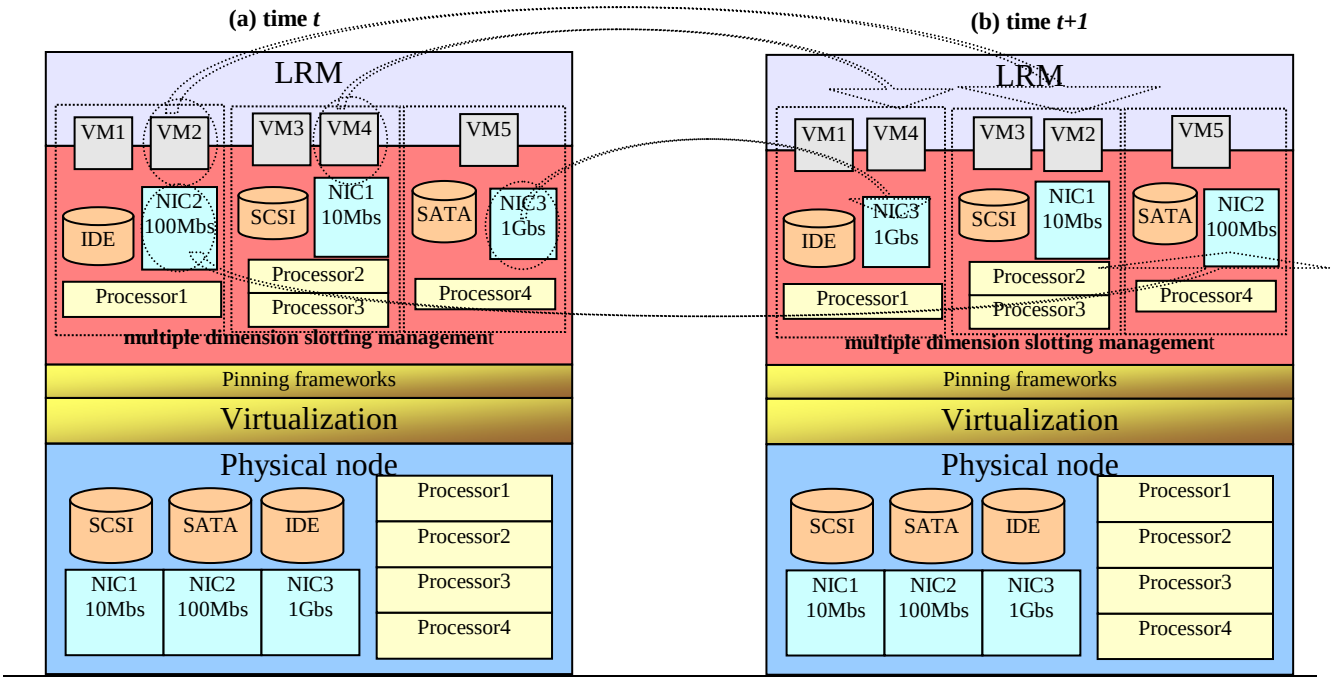

Figure 1. Example of the assignment mechanism of single physical resources at time *t* (a), and after reconfiguring at time *t+1* (b).

physical resources (and VMs associated) are exposed through a standard API interface to the upper layers of the LRM.

3.2 Architecture

For evaluating the proposed approach we have developed some of the components that are part of our GRM of LRMs enabled with adaptive resource mechanisms for virtual workspaces. Our LRM prototype aims to provide to the GRM with the infrastructure for accessing virtualized resources in a transparent way. It exposes a simple standard service interface for jobs execution by hiding the complexity of the managed local resources. The implementation of the proposed architecture leverages technologies that are currently in continuous development. Those technologies are virtualization (which is based on Xen) and Tycoon [10], a market-based system for managing compute resources in distributed clusters.

3.3 Prototype

The current organization of the global current architecture can be seen in Figure 2. In the following the components and their interrelations are explained in detail.

The *Local Resource Manager* (LRM) component offers public services via XML-RPC interfaces. Part of the actions that can be performed by LRM are currently encapsulated in a TycoonAPI component which interfaces with Tycoon. The current prototype of the proposed architecture uses Tycoon for managing the creation, deletion, boot and shutdown of virtual machines, as well as weighting the resources with its bidding mechanism.

TycoonAPI component is part of the LRM architecture. It has been designed to allow migration to other high level VM resource managers. Even though it is not intended to substitute Tycoon, the development of a high level resource manager is an open issue.

The design of the LRM includes other components that interfaces directly with Xen to perform tasks such as *monitoring* and *CPU capacity* management. The first is necessary to trace VMs performance in order to compute costs, and the second which implements the contributions of this paper achieves offering fine-grain CPU specifications for VMs in SMP architectures.

CPU capacity management is part of the *Multiple Dimension Slotting* Management (MuDiSM) and interfaces with the pinning framework offered by the virtualization layer (Xen). The *disk capacity management* and *network capacity management* are also part of the MuDiSM and complement each other to abstract subsets of physical resources.

The current architecture has a *txtLRM component* which is a client that requests actions to LRMs. A LRM must previously be located by the *discovery service* (or *directory component*). This service is part of the *Global Resource Manager* (GRM leverages txtLRM functionality and eventually substitute it); this component is currently designed but not implemented.

A *security component* is part of the architecture. It has the functionality to authenticate txtLRM requests in LRM via credentials and access lists. A *PKI subcomponent* enables the infrastructure to offer secure connections. Its main purpose is to upload sensitive job information like data or application (source or binaries) to the workspace.

In order to accomplish job management, the architecture has a *scheduler* component. Its functionality is to make a plan for requested job submissions satisfying user SLAs taking into acount available resources. This component is part of the GRM. The LRM also has a *job deployment component* to manage job executions in the workspace.

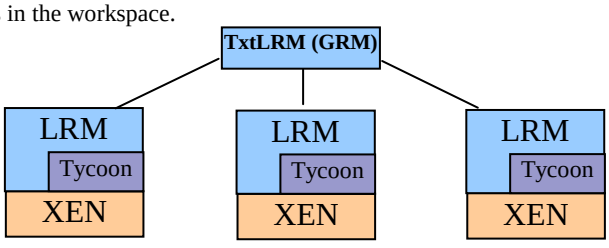

Figure 2. Baseline prototype.

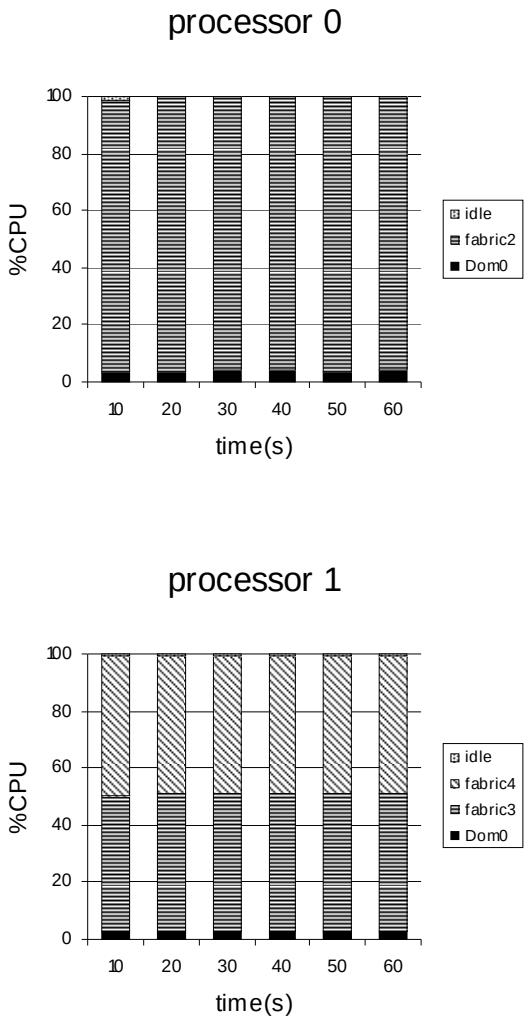

Figure 3. Three requests for VMs with 1GHz assigned without MuDiS. One VM is assigned to processor 0, and two VMs to processor 1.

Besides the standardized interfaces, another feature for the GRM is a *user interface component* for the final user. Its function is to allow job management submission and monitoring. Also, a *register component* allows final client (users or agents) obtaining credentials in order to perform requests to the GRM.

The information relative to workspaces (fabrics) state, users, and executed jobs, among others must be managed. This is done by the *information management component* of the LRM. The GRM must also have a similar component for clients, job submissions, and registered LRMs (directory component).

Finally, part of the LRM is the *invoicing component*, which computes a cost-function for the executed job plans. We have selected a first formula for investigation; however, we do not have a real justification for that choice. The current design includes the calculation of the invoice by: Min(CPU consumed, CPU assigned) * basic price, CPU assigned * basic price and CPU assigned * offered price (client budget).

The architecture is implemented in Python. It uses XML messages to define actions and also to interchange information between LRM servers and txtLRM clients.

4. Experiments

In our experiments the hardware of the resource provider node is a Pentium D 3.00GHz (two processors), 1GB of RAM, one hard disk (160GB, 7200 rpm, SATA), and one network interface card (Broadcom NetXtreme Gigabit). The operating system is Fedora Core 4 and the system-level virtualization solution is Xen 3.0.2. The Tycoon client and auctioneer are 0.4.1p133-1 version.

The prototype implements multiple dimension slotting by creating subsets of processors. In our experiments, two subsets, each one with one processor are created. The Xen scheduler is sEDF (scheduler Earliest Deadline First) which provides weighted CPU sharing per virtual machine. We also use *vcpu-pin* to restrict a VM to be executed in a specific physical CPU.

We evaluate our approach with four experiments. The first three experiments using a SMP node, and the fourth using a uniprocessor node.

- In first experiment we allocate three VMs without MuDiS.
- In second experiment we evaluate MuDiS through two configurations.
- In third experiment we use MuDiS to allocate four VMs.
- In fourth experiment we allocate one VM in a uniprocessor node.

In first and fourth experiment we allocate four requests for the creation of virtual workspaces. The required processing power of each VM is 1GHz, 1GHz, 1GHz, and 3GHz, respectively. We evaluate MuDiS in the second experiment, but with different configurations.

In each experiment we launch an application to stress the CPU by repeating a task formed by four sets of different mathematical operations. We measure the start and end time of each set of operations and report it as time consumed per transaction. In each experiment we run the benchmark during 60 seconds on each virtual workspace. The output file reports the total time, the number of tasks, and the number of transactions.

In the first experiment we offer processor power as a whole reporting the capacity of the node as 3Ghz. In this experiment we instruct LRM to allocate three VMs that request 1GHz and then 1 VM that requests 3GHz. Using the Xen mechanisms each VM is created with equal share by weighting each one with 1/3 of the total share.

Figure 3 shows the CPU usage of each virtual workspace in the first experiment. Additionally, we report idle processor. Table 1 shows mean values of tasks, time, and transactions per second (*tps*); it also shows the standard deviation of *tps*. Tables in this work are obtained for 15 runs of each experiment. We observed that VMs of 1 GHz are assigned, however the fourth VM with requirement of 3GHz could not be allocated.

Table 1. Transactions per second achieved in each VM, all with equal share

VM	Tasks	Time	tps	σ
fabric2	137.53	60.20	9.14	0.26
fabric3	70.40	60.29	4.67	0.12
fabric4	70.60	60.28	4.68	0.10

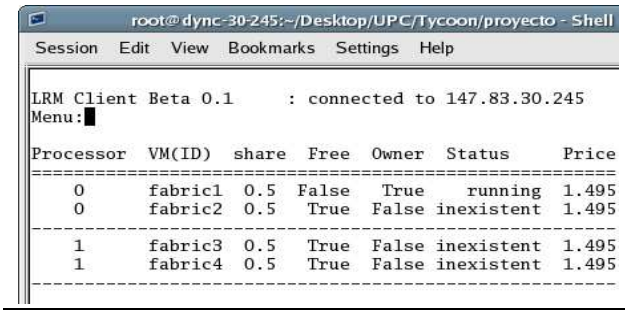

Figure 4. Example of partitioning through LRM client interface.

In the experiment one, the clients and GRM would expect equal performance in each VM, but the number of *tps* shows (Table 1) that fabric2 VM outperforms the rest. In fact, all of them receive more resources than the original requirements. We can argue that Xen is fair enough (in architectures with two processors) when we have an even number of VMs, but not in the case of an odd number as shown above. Nevertheless, we can not be tempted to generalize this approach when we need accurate information about the expected performance.

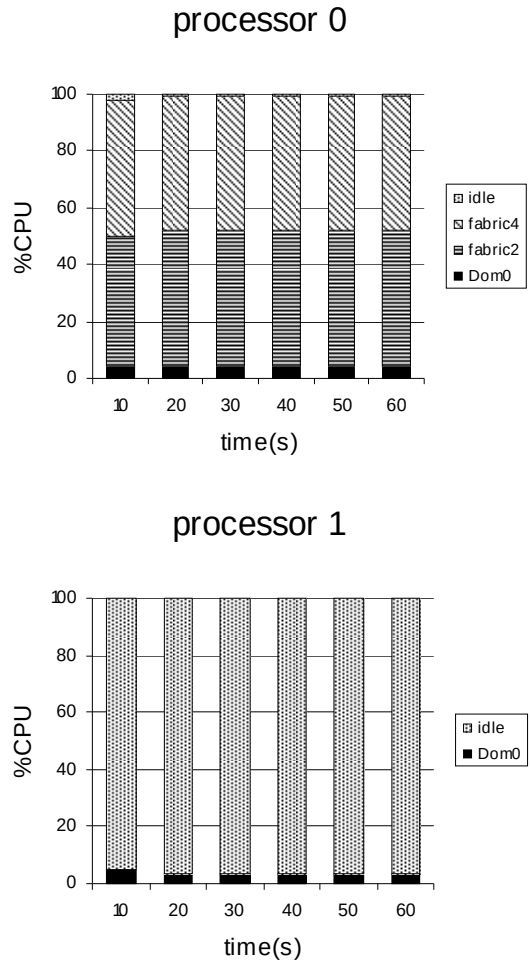

Figure 5. LRM allocates two requests for VMs of 1.5Ghz with MuDiS. Both VMs (fabric2, fabric4) are assigned to processor 0. Processor 1 has no assignment.

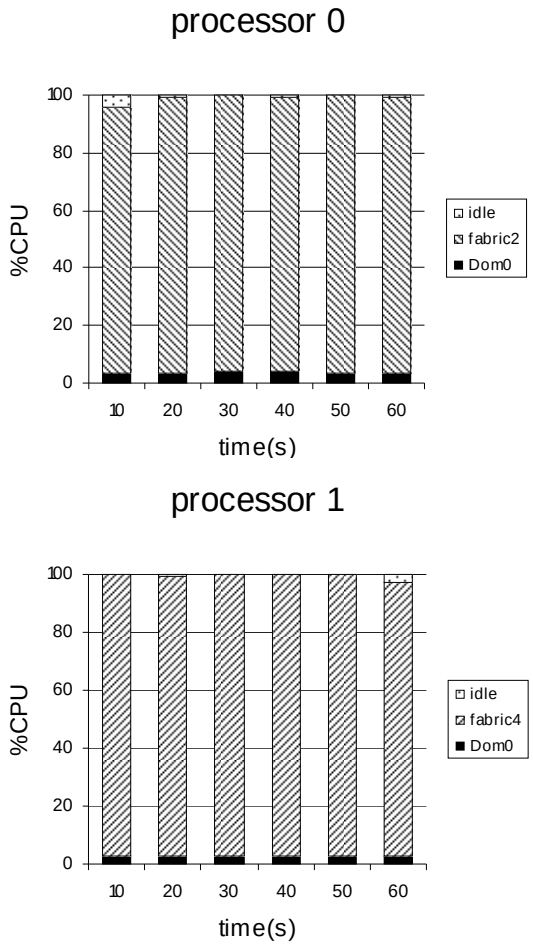

Figure 6. LRM allocates two requests for VMs of 3Ghz with MuDiS. First VM (fabric2) is assigned to processor 0, and the second VM (fabric4) is assigned to processor 1.

As we pointed out, the main problem is how to optimize local resources by allocating the maximum possible number of VMs with different constraints (CPU intensive, net intensive, or disk intensive).

Figure 4 shows a screenshot of the LRM client interface, as an example for partitioning the capacity of the node provider. Notice that we can manage the percent of share assigned to each VM by processor. Our current LRM prototype slices node statically, that is why the "Status" column shows VMs as "inexistent". We expect to offer dynamic slicing in a future prototype.

In the second experiment, instead of offering processor power as a whole, we offer it in terms of subsets of CPU using the MuDiS we propose. We set two configurations: a) we ask the LRM for the creation of two virtual workspaces and execute the benchmark, each one requesting 1.5GHz of processing power, and b), we ask the LRM for the creation of two virtual workspaces, each one requesting 3Ghz of processing power (see Figure 6).

In the first case, our LRM creates both VMs in processor 0 as shown in Figure 5 and leaves room for future allocations in processor 1.

In the second case we can see that with this approach the LRM can provide to clients (users or agents) detailed information about free resources and also meet the requested SLAs for virtual work-

spaces. Table 2 and Table 3 summarize the *tps* for the two configurations.

Table 2. Benchmark results allocating 1.5GHz in each VM.

VM	Tasks	Time	tps	σ
fabric2	69.47	60.41	4.60	0.13
fabric4	68.93	60.34	4.57	0.11

Table 3. Benchmark results allocating 3.0GHz in each VM.

VM	Tasks	Time	tps	σ
fabric2	135.80	60.22	9.02	0.20
fabric4	138.87	60.22	9.22	0.26

Following the same approach, we conducted a third experiment which consists of requesting to the LRM the allocation of four VMs meeting the requirements of 1GHz, 1GHz, 1GHz, and 3GHz (like in the first experiment). Figure 7 shows the CPU utilization for each VMs. We can see in the graphs that the benchmark application launched in each VM achieves the full utilization of assigned CPU shares.

In Table 4 we can see that the transactions per second achieved in third experiment are consistent with requested SLAs. We can see that the fine grain LRM has been able to allocate the 4 requested VMs, different to the baseline system used in the first experiment.

Table 4. Benchmark results for third experiment.

VM	Tasks	Time	tps	σ
fabric1	133.93	60.20	8.90	0.42
fabric2	45.13	60.57	2.98	0.14
fabric3	45.00	60.47	2.98	0.16
fabric4	44.93	60.54	2.97	0.13

Finally, in a fourth experiment we deploy for comparative purposes our prototype in a laptop with one Pentium IV M 3.06GHz processor and ask the LRM to create a VM with 3GHz. Table 5 shows the measured *tps* results from running same benchmark.

In the uniprocessor architecture we do not see a significant difference in the number of *tps* compared with the results of our former SMP architecture testbed. However, we could obtain a small deviation caused by two factors: by the internal processor architecture and second because of the overhead generated by the privileged Xen domain-0 which runs on only one processor. Nevertheless, it reports and meets the expected performance when running our benchmark.

Table 5. Benchmark results allocating 3.0GHz in a uniprocessor node.

VM	Tasks	Time	tps	σ
fabric1	136.67	60.18	9.08	0.12

From the second experiment we observe that MuDiS allows accurate assignment of requested CPU. In both configurations of the second experiment the SLAs are meet. Moreover, in its first configuration only one processor is used for the two 1.5GHz VMs, leaving the second processor for future allocations. Comparing the allocation in the first and third experiment we observe that MuDiS gives us a better assignment of VMs to processors by performing the creation, assignment and monitoring of the VMs. The advantage of the LRMs and interest to GRM is that the MuDiS is able to better map SLAs to resources while having the capacity to adapt resource assignment to the required performance of the user application.

5. Outlook and conclusions

We have presented a multiple dimension slotting approach for local resource managers (LRMs) to manage in a fine grain way the virtualized resources.

The approach adds another layer for the management of virtualized resources. We target three different features: fast responses, light footprint and no overhead. The implemented algorithms to adjust subsets, re-locate resources (or VMs) and adjust shares must not compromise the global performance of the entire node. The algorithms must not interfere with the hosted VMs and its applications. The multiple dimension slotting must be precise when managing the share assigned to each VM on each resource.

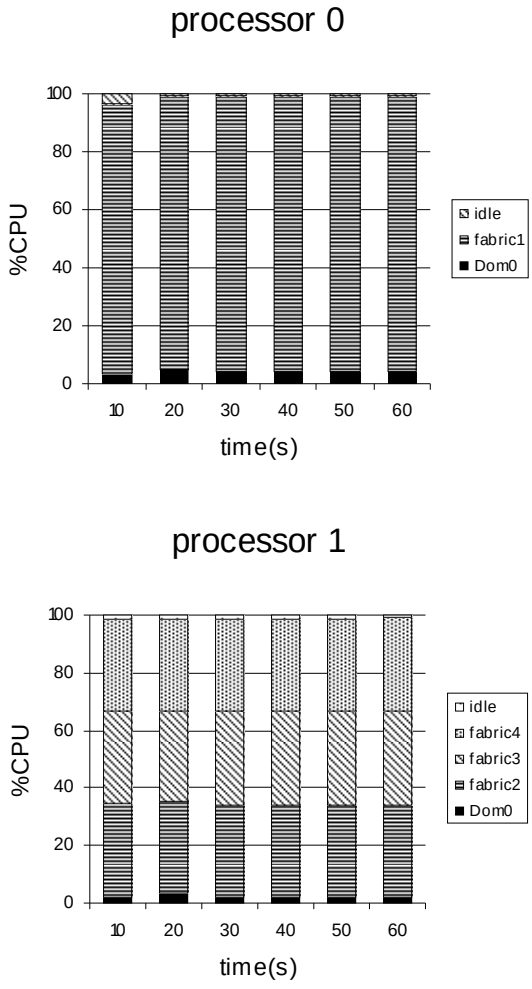

Figure 7. LRM allocating one VM in processor 0 and three VMs in processor 1.

The LRM has been implemented taking advantage of Xen virtualization and Tycoon. Our preliminary results obtained from experiments showed that the local physical resources were properly assigned, such that the performance measured in transactions per second was accurate and corresponded to agreed SLAs. We made four experiments running an application that behaves as a CPU intensive job. In the first experiment which represents the baseline system, we used a coarse-grain approach offering resources with the global share mechanism. In the second and third experiment we used the proposed MuDiS management to partition resources in their singles entities. We observed that the fine grain approach implemented in the MuDiS-LRM achieved a better allocation of resources compared with the original system. Our approach could be further enhanced by dynamic adaptation of assigned resources to application behaviour.

Acknowledgments

This work was supported in part by the European Union under Contract SORMA EU IST-FP6-034286, the Ministry of Education and Science of Spain under Contract TIN2006-5614-C03-01, and the Mexican Ministry of Education under program PROMEP (PROgrama de MEjoramiento del Profesorado). We thank the review committee for their time and insightful comments.

References

[1] Barham, P. Dragovic, B. Fraser, K. Hand, S. Harris, T. Ho, A. Neugebauer, R. Pratt, I. and Warfield A., "Xen and the art of virtualization," in Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp. 164–177, October 2003.

[2] Chase, J. Irwin, D. Grit, L. Moore, J. and Sprenkle, S. Dynamic Virtual Clusters in a Grid Site Manager. Twelfth IEEE Symposium on High Performance Distributed Computing (HPDC), June 2003, Seattle, Washington.

[3] Clark, C. Fraser, K. Hand, S. Hansen, J. G. Jul, E. Limpach, C. Pratt, I. and Warfield. A. “Live Migration of Virtual Machines”. In Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), Boston, MA, May 2005.

[4] Figueiredo, R.J. Dinda, P.A. Fortes, J.A.B. “A case for grid computing on virtual machines”. In Proceedings of Int. Conf. on Distributed Computing Systems (ICDCS), 2003.

[5] Fu, Y. Chase, J. Chun, B. Schwab, S. and Vahdat, A. “SHARP: An Architecture for Secure Resource Peering”. In Proceedings of the 19th ACM Symposium on Operating System Principles, October 2003

[6] Huang, W. Liu, J. Abali., B. and Panda, D.. “A Case for High Performance Computing with Virtual Machines”. The 20th ACM International Conference on Supercomputing (ICS '06), Cairns, Queensland, Australia, June 2006

[7] Irwin, D. Chase, J. Grit, L. Yumerefendi, A. Becker, D. and Yocum, K. “Sharing Networked Resources with Brokered Leases”, USENIX Annual Technical Conference (USENIX), June 2006, Boston, Massachusetts.

[8] Keahey, K. Foster, I. Freeman, T. Zhang, X. Galron, D. “Virtual Workspaces in the Grid”, Europar 2005, Lisbon, Portugal, September, 2005.

[9] Kiyanclar, N. Koenig, G.A. and Yurcik, W. “Maestro-VC: On-Demand Secure Cluster Computing Using Virtualization”. 7th LCI International Conference on Linux Clusters, May 2006.

[10] Lai, K. Rasmusson, L. Adar, E. Sorkin, S. Zhang, L. and Huberman, B. "Tycoon: an Implementation of a Distributed Market-Based Resource Allocation System", HP Technical Report, December 8, 2004

[11] Mergen, M. F., Uhlig, V., Krieger, O., and Xenidis, J. 2006. Virtualization for high-performance computing. SIGOPS Oper. Syst. Rev. 40, 2 (Apr. 2006), 8-11.

[12] Ruth, P. Rhee, J. Xu, D. Kennell, R. and Goasguen, S. “Autonomic Live Adaptation of Virtual Computational Environments in a Multi-Domain Infrastructure”. IEEE Int’l Conf. on Autonomic Computing (ICAC’06), June 2006

[13] Sundararaj, A. and Dinda, P. A. “Towards Virtual Networks for Virtual Machine Grid Computing”. 3rd USENIX Virtual Machine Research and Technology Symp., May 2004.

[14] VMware. http://www.vmware.com.

[15] Xu, D. Ruth, P. Rhee, J. Kennell, R. Goasguen, S."Short Paper: Autonomic Adaptation of Virtual Distributed Environments in a Multi-Domain Infrastructure", Proceedings of The 15th IEEE International Symposium on High Performance Distributed Computing (HPDC'06), Paris, France, June 2006

[16] Youseff, L. Wolski, R. Gorda, B. Krintz, C. “Paravirtualization for HPC Systems”. ISPA Workshops 2006: 474-486