The Virtual Cluster: a Dynamic Environment for Exploitation of Idle Network Resources

C. De Rose
PUCRS - Brazil
derose@inf.pucrs.br

F. Blanco
CPAD - PUCRS/HP - Brazil
blanco@cpad.pucrs.br

N. Maillard
CPAD - PUCRS/HP - Brazil
nicolas@cpad.pucrs.br

K. Saikoski
PUCRS - Brazil
saikoski@inf.pucrs.br

R. Novaes
HP Brazil
reynaldo_novaes@non.hp.com

O. Richard
ID-IMAG, Grenoble - France
Olivier.Richard@imag.fr

B. Richard
HP Labs, Grenoble - France
bruno_richard@hp.com

Abstract

Standard environments for exploiting idle time of workstations are based on some kind of spying process that detects low CPU usage and informs to a scheduler so that work can be dispatched. This approach generates local interference and, since the same local environment is used, could lead to security problems. We are investigating the exploitation of idle times in network resources based on a complete mode change in a candidate node. After the detection that some node is idle, a Mode-Switcher boots a new operating system that will work over a separate disk partition. After the boot phase the node is linked to a logical network topology and is available to receive jobs. Users can allocate nodes from this virtual cluster through a standard front-end as they would do in a "conventional" cluster. Because nodes may leave and join this virtual machine we use a distributed processor management to allow user applications to cope with this dynamic resource behavior. In this paper we describe the architecture of the virtual cluster and present the results obtained with a Mode-Switcher and a prototype application under real use conditions.

1. Introduction

One decade ago the idea appeared to harness the global computing and storage power of computers all around the world through the Internet network. From Cluster Computing, which aimed at adapting distributed computing to a loose coupled network of workstations or standard PCs, the community moved to Grid Computing [3] to use computing resources from different, possibly virtual, organizations. For example the Condor and the Globus project are intending to provide a set of tools to match its needs of computing power, regarding a certain application, and the available Grid resources.

Recently, some Global Computing projects appeared that explicitly intend to use idle cycles of machines connected to the Internet. A typical example is the SETI@Home project which provides a task scattered computation, whose elementary task can be downloaded from a server and will be executed locally whenever the computer turns unused (e.g., when the screen-saver turns on).

The key points of these approaches are that the computing nodes are scattered across the Internet: some may be part of an existing cluster, some may be isolated. Besides, the nodes may suddenly switch off, for instance because the local user went back to work or because a modem turned off. And least, since the execution of a distributed computation will involve distinct unknown nodes, security concerns must be tackled to prevent a malign code to infest the host nodes and to prevent ill-intended nodes providers to spoil the result of the computation. The scope of this article is the design of an environment, V Cluster[1] that could deal with these three points: usage of idle cycles, handling of dynamic behavior of the nodes, and safe remote ex-

[1] Project supported by HP Brazil.

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE

ecution.

This paper is organized as follows: the first section presents the key ideas of three classical environments for distributed Grid computing (Mosix, Condor and Globus). We then describe our virtual cluster project and its components: the Mode-Switcher that deals with the mode change between local processing and cluster mode, the DPM resource manager that allows the dynamic handling of the nodes, and the applications that we intend to develop on top of vCLUSTER. In the last section we present some measurements in order to validate our approach. This work is the latest step in an on-going work at the CPAD-PUCRS. The main contribution is the integration of the Mode Switcher with DPM in order to provide an effective (virtual) cluster adapted to distributed efficient computations.

2. State of the art

In this section we briefly present the functionalities offered by some of the classical environments for exploitation of idle time in workstation networks.

Mosix [1] is one of the oldest projects aiming at handling the job execution and the load balancing on a cluster. This environment provides a preemptive process migration mechanism at the operating system level, in order to generate a SMP- like vision of the computational resources. When a certain amount of the resource has been used (e.g., at memory exhaustion), it is up to the Mosix system to migrate some or all of the processes constituting the job to some other nodes of the cluster, thus achieving load balancing. The nodes of the Mosix architecture exchange some knowledge about the other resources through gossiping, *i.e.* each node regularly sends information about himself to a randomly chosen subset of its neighbors. Thus, there is no centralization of the resource's state such as in Condor. Mosix was not intended to be used for the exploitation of workstation idle time but some of its mechanisms for information discovery and process migration where used as a reference for the development of other systems in this field.

Condor [6], developed at the Wisconsin University, has been developed to use idle cycles of independent workstations. The centralized manager of a Condor pool of machines handles a list of the nodes to determine if they are available or not. When idle nodes are detected, a matching system between the available resources and the user's needs is done to allocate the jobs. This is transparent to the user. Condor uses a remote system call mechanism and a checkpoint technique to migrate and (re)start the jobs on the available nodes, similar to Mosix. Some restrictions exist, such as the impossibility to launch multi-process jobs, the lack of secure protocols or sandboxing systems for remote execution.

The Globus toolkit [3] provides a Grid environment with components dedicated to information discovery, communication protocols (among secured ones), resource allocation and management, and data management. Globus has been used on a variety of existing Grids and several applications are being developed on top of it (e.g, climate simulation and tomography). The Globus3 toolkit [5] intends to provide web based services to access the software components. The component in charge of the resource discovery and handling uses the Metacomputing Directory Services (MDS) which organizes the information in a tree structure and acts as a server when client jobs request resources. The information about the resources is handled on each node by local agents and centralized by the MDS service. This one may, in turn, be implemented in a distributed way ([4]) but to our knowledge no result has been published with a distributed service yet.

3. The virtual cluster approach

3.1 Motivation and idea

Based on our experience in the field of cluster architectures, (developing middleware and implementing parallel applications), we wanted to explore some concepts of grid computing. Our first experiments where in the area of cluster of clusters, evaluating how our middleware had to be expanded so that a single parallel application could use resources allocated in more than one cluster. Going further in this direction we tried to think of a simple way to explore the idle potential of the workstations in our lab and surroundings.

The idea was not to throw away everything we did in the last five years but to use this knowledge as a foundation for the development of a system that would incorporate some elements of grid computing. The user should be able to allocate these idle workstations in the same way it allocates and uses the available nodes in the "conventional" clusters. So we incorporated in our cluster management software an additional cluster, a virtual one. The vCLUSTER aggregates the idle workstations from our lab so that a user may allocate nodes and run applications using the same scripts available for the other clusters. To detect which nodes are joining and leaving the vCLUSTER we used a mode switcher module developed for another HP project, the iCluster [8]. This Mode-Switcher detects idle resources and

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE

boots a new operating system which runs on a different disk partition. If the local user needs the workstation again the node switches back to the original operating system without any data loss (see section 4.1). With this concept we guarantee no interference in the performance of the workstation when it is in use and that, when in cluster mode, no local data may be accessed.

3.2 The vCluster architecture

The three possible states of a vCluster node are the following (see figure 1):

Node in standard mode: represented in figure 1 by the white circles. These are nodes that are not idle and therefore are not part of the vCluster.

Free node in cluster mode: represented in figure 1 by the light gray circles. Because these nodes where idle, the Mode-Switcher booted the cluster operating system and connected them into a logical topology (tree based). This logical topology will be used to find free nodes when a request for the vCluster arrives. No distributed processing is running in these nodes because they where not allocated yet.

Allocated node in cluster mode: represented in figure 1 by the dark gray circles. These are nodes in cluster mode that are already allocated and executing a distributed application. The thicker connection lines represent the links to the other allocated nodes running the same application.

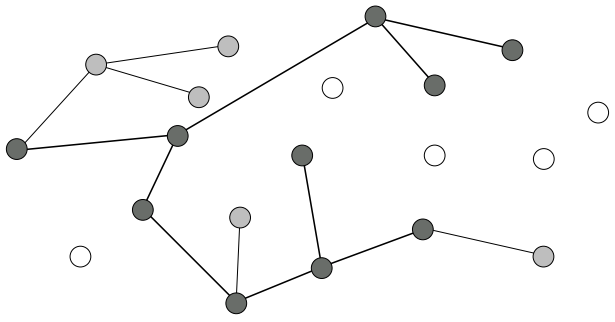

Figure 1. Possible states of a node and the vCluster logical topology.

Figure 2 shows a high-level representation of the vCluster architecture. After a node changes to the cluster mode and boots the cluster operating system, it runsthe Distributed Processor Management (DPM) daemon [2]. DPM connects the node to the logical cluster topology and implements a dynamic processor management ensuring that a distributed application may request additional nodes or partially release the nodesit is using. We implemented an additional service for vCluster so that DPM can cope with nodes that abruptly leave or join the vCluster (see 4.2).

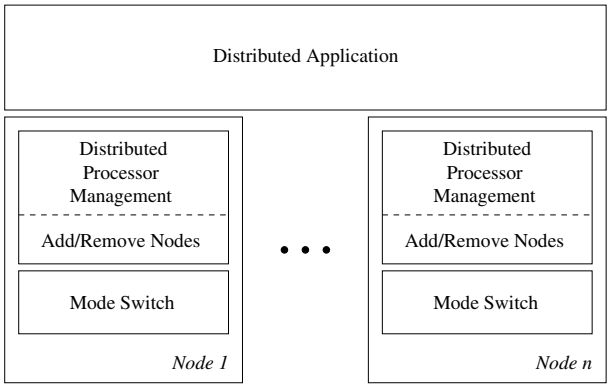

Figure 2. vCluster architecture and it's layers.

On top of DPM a distributed application may run as in a conventional cluster. It has to be aware of two situations:

1. nodes may abruptly leave the vCluster (change back to standard mode);
2. nodes may join the vCluster and could be used to improve performance.

We treat both situations with a very simple approach. We prepared a basic application framework to run in the vCluster. It is based in a client/server model where clients ask for work and the server is responsible to manage the work that is already done. Regarding the situation 1, the server processes has to implement some fault-tolerance so that it can detect if a client is not responding anymore. To avoid the problem of the server node going down we put the server in a special fixed node. This node never changes to standard mode and acts like a host machine for the vCluster. Together with the server we execute also a client process, so that, even in the minimal case (with one node), the distributed application makes progress. Regarding the situation 2, we implemented a service in our application framework to test periodically for new nodes in the vCluster and automatically dispatch new clients.

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE

4 The vCluster layers

4.1 Mode-Switcher

The Mode-Switcher [8], developed by HP Brazil, is the module responsible for alternating between the user operating system (standard mode), and the cluster operating system (cluster mode).

The main Mode-Switcher's role is to keep track of the usage of the machines where it is installed and identify which periods these machines are idle and could be exploited as cluster nodes. Based on this knowledge the Mode-Switcher initiates the transition to the cluster mode without user intervention. Its design has the following characteristics:

Efficient mode switching: switching between modes is implemented in a very efficient way resulting in switching times in less than one minute.

Localise has priority: no context save or management operations in cluster mode should delay the switching to the standard mode when a user decides to use the machine.

Zero local interference: the Mode-Switcher installation does not have any impact on the target system performance when in standard mode.

No special hardware requirements: No special hardware is required. The Mode-Switcher is implemented completely in software. No specific hardware is needed other than a network connection.

A state diagram of the Mode-Switcher is presented in figure 3.

During normal operation, a local user might not use his/her workstation all the time, leaving some idle time that could be exploited. The Mode-Switcher tracks this time and tries to find a usage pattern for this workstation. When the detected idle time matches the pattern, the Mode-Switcher performs the context switching. Both context saving and restoring operations are done using system calls. So, since these context switching tasks are provided by the operating system API, they must be theoretically trustworthy and we assume that they occur without any data loss. The switching from the user mode to cluster mode is presented in the figure 3 as the transition labeled as "Idle time detected". After the context switching the machine will boot the cluster operating system and then it can be used as a cluster resource.

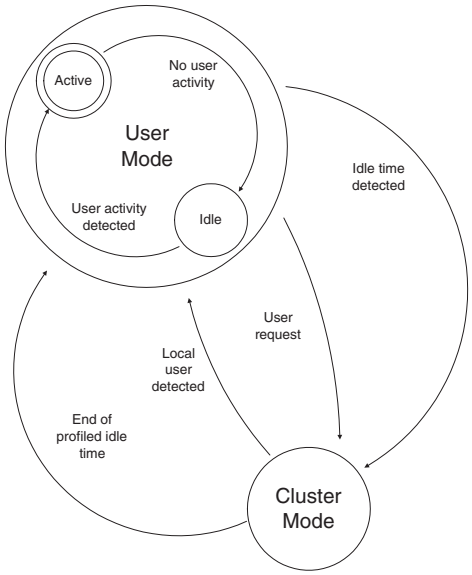

Figure 3. State diagram of the Mode-Switcher

Two events will force the machine to return to standard mode: the detection of a local user presence or the end of the profiled idle time. In the first case, any mouse movement or keyboard hit is used to detect the presence of the local user. The second case will not require any user intervention because the Mode-Switcher will switch back to the user mode based on its knowledge about the workstation usage. In both cases the standard mode will be restored to the state it has before switching.

In the standard mode the Mode-Switcher will present itself as a small icon in the users task bar to indicate it is running. As said before, it does not interfere with the normal operation of the machine and the user does not lose the control over the machine because the cluster mode activation may be postponed, immediately activated or conditioned by specific events (after a defined idle period).

4.2 Distributed Processor Management

DPM is a distributed processor manager that allows dynamic addition and removal of nodes to be managed. It is totally distributed, this means that there is not any central module or manager that is responsible for the inclusion or exclusion of nodes.

Users can allocate their jobs, and use the API provided by DPM to communicate with the system. So, user's applications can allocate more nodes at execu-

tion time and spawn their jobs among nodes that became free after the initial allocation.

There is a daemon running on each node of the cluster to control local information about the node. To minimize network interference, DPM uses a hierarchical logical topology when communicating with the nodes. To DPM, the nodes are logically organized in a tree hierarchy and the messages exchanged between each node respects this hierarchy.

4.2.1 Logical hierarchy

DPM mounts its logical hierarchy when it starts. There are three ways to do this: static, dynamic with broadcast or dynamic with unicasts. Each of these has different network bandwidth requirements. In the VCLUSTER architecture we set DPM to work with the dynamic unicast mechanism because of the dynamic characteristics of the environment. The information stored by each daemon corresponds only to its local state (free or allocated) and its neighbors (the nodes it has to communicate with). Therefore, it stores info about its father and a list of its children.

A configuration file indicates how the nodes are organized in groups, and a search on the network will only consider the nodes that are already running the DPM daemon. The network search is made using unicasts. A message is sent for each DPM daemon and the node is considered online, if it replies, or offline, if a timeout occurs.

The format of the configuration file used is described below:

```
##############################################
#Format of this file :
#STARTUP_HOST <HOSTNAME>
#GROUPS
#<GROUP NAME> <NODE 1> <NODE 2> ... <NODE N1>
#<GROUP NAME> <NODE 1> <NODE 2> ... <NODE N2>
#<GROUP NAME> <NODE 1> <NODE 2> ... <NODE N3>
#HIERARCHY
#<GROUP NAME> <NAME OF CHILD GROUP>
#<GROUP NAME> <NAME OF CHILD GROUP>
#
#Example:
STARTUP_HOST marfim
GROUPS
switch1 ipe pinus jacaranda
switch2 jequitiba carvalho
switch3 araucaria mogno mangueira quaruba
HIERARCHY
switch1 switch2
switch2 switch3
```

4.2.2 Node inclusion

Due to the fact that DPM does not use any central module, the daemons must have a way to find which nodes are online or offline when they start, to properly configure the logical hierarchy.

Considering the following configuration file:

```
STARTUP_HOST marfim
   GROUPS
      group_1 A B C
      group_2 D E
      group_3 F G H I
   HIERARCHY
      group_1 group_2 group_3
```

If the nodes are turned on in the following order: F, I, H, A, B, D, the configuration process would occur as in figure 4.

The nodes, when turned on, search for already online nodes and decide which nodes will be their father and which will be their children. In figure 4, when t=4 and node A is entering in the hierarchy it decides that it will have no father and node F will be its child. So, node A will send a message to node F to change its father to node A.

Node A does not enter as child of F because it is in group group_1, and group_1 is the father of group_2, as we can see in the configuration file above. Thus, node A decides that it will be the father of the node F because all nodes that were online when node A tried to enter are on group group_2.

4.2.3 Node exclusion

When a node is switched to standard mode, it has to leave the hierarchy in a consistent state. This is done by calling the internal function `reconfigureNeighborhood()`. This function analyzes the local information and decides which changes have to be done.

For example, if node F wants to get out of the hierarchy shown above, it will send messages to nodes A, H and I (figure 5). Nodes H and I will receive messages to change their father, and node A will receive messages to add nodes H and I as its children and remove child F.

4.2.4 DPM API

DPM provides an API to allow the system administrator to write their own queue managers programs and to allow users applications to grow and shrink dynamically.

The DPM API is subdivided in three categories: Administration, Application and Common.

Administration functions

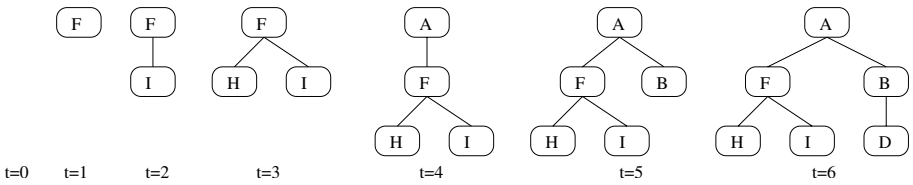

Figure 4. Node inclusion in DPM.

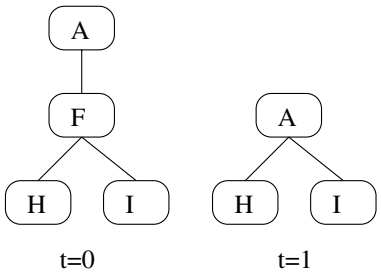

Figure 5. Node exclusion in DPM.

The administration functions can only be used by the system administrator. They are responsible for job allocation, job removal and logical hierarchy reconfiguration. The DPM API functions in this category are:

dpm_alloc_job This function starts the allocation of nodes to a job that was not previously known by the system.

dpm_remove_job This function is used by the system to completely remove a job partition.

dpm_job_nodes This function is used to retrieve the list of nodes of a specific job.

dpm_add_child This function is used to add a node as a child of an already running DPM daemon.

dpm_remove_child This function is used to remove a node that is the child of an already running DPM daemon.

dpm_change_father This function is used to change the father of an already running DPM daemon.

dpm_reconfigure This function is used to force a daemon to redo the configuration of the logical hierarchy used by it.

dpm_exit_hierarchy This function is used to cause a daemon to be invisible to the actual logical hierarchy. It will tell to its neighbors to remove any references to it.

Application functions

The application functions are to be used by the user's application to communicate with the DPM system. These functions allow the users to add more nodes to its allocated partition, release allocated nodes or retrieve a list of current allocated nodes. The DPM API functions of this category are:

dpm_alloc_node This function is used to allocate additional nodes to the current application's partition.

dpm_release_nodes This function is used to release nodes from the current application's partition.

dpm_release_all_nodes This function is used to remove all nodes from the current application's partition.

dpm_my_nodes This function retrieves the list of nodes from the current application's partition.

Common functions

The Common functions can be used either by the user's application or by the administrator. The DPM API functions of this category are:

dpm_init This function should be called before any other function of the DPM API. It is responsible for contacting the DPM daemon and establishing a connection between the user program and the daemon.

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE

dpm_finalize This function should be called after DPM use.

dpm_perror This function prints a status message about the latest executed function.

dpm_run_program This function runs a program on a remote host.

dpm_insert This function is used to insert the nodes in a dpm_iplist_t variable.

dpm_node This function is used to access the nodes contained in a dpm_iplist_t variable.

4.3 Distributed application

To validate the vCLUSTER architecture we implemented a prototype application based on an initial version of our application framework. Since communication costs may be very high among idle workstations spread in a university campus or different floors of a company building, the application should be coarse grained to allow gaining performance in such an environment. It's also important that partial results may be easily rescheduled to other nodes in the case that some machine abruptly leaves the vCLUSTER. Based on these characteristics we have chosen a distributed Ray-Tracer application for the first prototype.

POV-Ray (Persistence Of Vision Raytracer [7]) is a three-dimensional rendering engine. The program derives information from a file containing the description of a scene (objects, textures, lights and point of view) simulating the way the light interacts with the objects in the scene to obtain a three-dimensional realistic image (procedure known as ray tracing). Each pixel of the resulting image may be calculated from the scene description without knowledge of neighbor points. MPIPovray 3.01 is a parallel implementation of the above application that divides the image to be calculated in horizontal slices, mapping each slice to one slave process. A master process is dedicated to the image partitioning, slices distribution and screen drawing. Slave processes do not wait for the calculation of the whole slice and send ready screen lines to the master to allow real time drawing of the already calculated points.

We have changed the distributed implementation of the MPIPovray so that the dynamic vCLUSTER nodes have the initiative to request work, like in a client/server relation. The clients (running on the vCLUSTER nodes) connect to a server to request a section of the image, render the received section and then send the rendered lines back to the server. Thus, each node renders a section of the image and the complete image is drawn only on the server.

The use of the client-server architecture was chosen because all the initiative must start from the client side. As such, if a node is not in the cluster mode, it is simply ignored. Also, the server does not store any information about the active clients. This is particularly important because the nodes of the virtual cluster may change to user mode at any time, and this characteristic allows the clients to abort their execution at any point of the code, without the need to do any extra work after the user activity detection.

Therefore, the server must guarantee that the entire image will be rendered independently if a section was delegated to a node that changed back to user mode during the execution or no. This is done by re-sending the section that was lost when all other sections were received. So, when a client has to change back to the user mode, it does not have to report this to the server, resulting in a more efficient switching operation.

5. Performance evaluation

5.1 Mode-Switcher

In table 1 we present the switching times achieved by the Mode-Switcher in our test node (mean time of 5 measurements for each case). The node is a PC with an Intel Pentium 4 1.6 GHz processor and 128MB of RAM memory. The operating system used in the cluster mode was Linux. In standard mode the node runs Windows 2000 (or whatever other more recent version). These measures show that if a student tries to use a workstation in cluster mode it has to wait only around 35 seconds.

Mode-Switching	Time(s)
Linux to Windows	34.88
Windows to Linux	49.04

Table 1. Switching times in a HP E-PC node.

5.2 Distributed Ray-tracer application

The performance evaluation of the distributed version of POV-Ray was done rendering an image with 640x480 pixels (chess2.pov). The image has been partitioned in slices of 15 lines. This means that when a client requests a work to the server, it will receive a slice with 15 lines to render. The client will request a new section only after its previous 15 lines have been

rendered. Each client node used was a PC with an Intel Pentium 4 1.6 GHz processor and 128MB of RAM memory. The operating system used in the cluster mode was Linux. These computers are part of the laboratory used by the undergraduate students of PUCRS.

The time that the application needs to render the image decreases as new clients are included in the calculation. The time that the sequential version needs to render the image is 23 minutes and 52 seconds. Using the distributed version, running with 16 clients, the same image is rendered in 1 minute and 54 seconds (i.e. the efficiency is 78.5%).

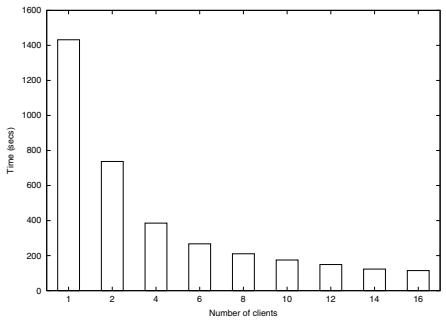

Figure 6. Performance of the distributed pov-ray application over the VCLUSTER.

In order to have a more accurate idea of how faster the distributed version is, the execution times are shown in figure 6. It's easy to see that the speedup is increasing with the addition of new nodes. Also, the performance speedup is still growing with 16 clients. This means that the maximum speedup acquired in our test, 12.49, is not the maximum possible. With more clients, the speedup would be greater. This is expected because of the coarse granularity of the problem.

Thus the POV-Ray example shows that our preliminary implementation of the VCluster environment can yield significant parallel gains for master/slave based applications.

6. Conclusions

In this paper we have presented the VCLUSTER environment. It proposes a virtual architecture to exploit idle workstation power based on a Mode-Switcher module. A Distributed Processor Management layer is used to handle the dynamic behavior of the nodes in the virtual cluster by using a distributed tree structure in which the nodes are listed. We have also presented the results of an experiment with a distributed version of a Ray-Tracing application in this environment.

Our preliminary experiments rely on a client/server programming model to test our environment with coarse-grained parallel applications. Research remains to be done in order to add some dynamical execution capabilities to the VCLUSTER that could deal with fine-grained computations, possibly with data-dependencies that would prevent from using a client/server approach. In turn, this would imply some considerations on the scheduling strategies that could use some information on the nodes, collected by the DPM layer.

We believe that the virtual cluster is an interesting alternative to traditional clusters for coarse-grained parallel applications. It has to be investigated if this virtual architecture is scalable and if it could also be interesting for other kinds of applications.

References

[1] A. Barak, S. Guday, and R. G. Wheeler. *The MOSIX Distributed Operating System*. Springer, Berlin, 1993.

[2] C. De Rose and H.-U. Heiss. Dynamic processor allocation in large mesh-connected multicomputers. EURO-PAR 2001, 2001, Manchester, Inglaterra. Published in Lecture Notes in Computer Science (LNCS), 2001.

[3] I. Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science*, 2150:1–??, 2001.

[4] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of International Workshop on Quality of Service*, 1999.

[5] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed systems applications. *IEEE Computer*, 35(6):37–46, June 2002.

[6] M. Livny, J. Basney, R. Rajesh, and T. Tannenbaum. Mechanisms for high throughput computing. *Speedup Journal*, 11(1), June 1997.

[7] Persistence of vision(tm) ray tracer, 2001. Available at http://www.povray.org.

[8] B. Richard and P. Augerat. I-cluster: Intense computing with untapped resources. MPCS'02, Ischia, Italy, April 2002.

Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'02)
0-7695-1772-2/02 $17.00 © 2002 IEEE