Scalable Computing: Practice and Experience
Volume 6, Number 3, pp. 107–115. http://www.scpe.org
ISSN 1895-1767

NON-DEDICATED DISTRIBUTED ENVIRONMENT: A SOLUTION FOR SAFE AND CONTINUOUS EXPLOITATION OF IDLE CYCLES

R. C. NOVAES[*] , P. ROISENBERG[*] , R. SCHEER[*] , C. NORTHFLEET[*] , J. H. JORNADA[*], AND W. CIRNE[†]

Abstract. The Non-Dedicated Distributed Environment (NDDE) aims to muster the idle processing power of interactive computers (workstations or PCs) into a virtual resource for parallel applications and grid computing. NDDE is novel in the sense that it allows for safe and continuous use of idle cycles. Differently from existing solutions, NDDE applications run inside a virtual machine rather than on the user environment. Besides safe and continuous cycle exploitation, this approach enables NDDE applications to run on an operating system other than that used interactively. Our preliminary results suggest that NDDE can in fact harvests most of the idle cycles and has almost no impact on the interactive user.

Key words. Grid Computing, Virtual Machines, Idle Cycles.

1. Introduction. Modern desktop computers and workstations have powerful computational capabilities that are used primarily to provide short response times to the user's daily activities like word processing, spreadsheet calculations or web page rendering. Most of the time, however, this processing power is idle, waiting for occasional user inputs or requests. During this unused periods that can range from fractions of a second (e.g. between user keystrokes) to hours, the operating system normally executes an "idle" process, which is a dummy process with the lowest priority on the system, so it runs only when there is no other process or service needing to be executed. The processing capacity used to run this idle process is in fact being wasted.

The Non-Dedicated Distributed Environment (NDDE) aims to potentially use all of this fragmented idle time from most or all the machines connected to a network. This will create a very low cost virtual resource with only minimal interference on the normal operation of the interactive users. Such a virtual resource can be directly used to run parallel application or can be a component of a large computational grid.

Of course, this is not a new idea. Systems like Condor [1] and SETI@home [2] are classic examples of successful exploitation of idle cycles to do useful computation. NDDE differs from these in the sense that it allows for safe and continuous use of idle cycles. It is safe because it is much harder for a malicious guest application to tamper with user data and environment. It is continuous because it avoids the "interactive versus idle" resources dichotomy. That is, NDDE enables both environments to run concurrently so the workstation does not need to be totally idling to make its resources available. It can exploit idle resources in a much finer grain.

Safe and continuous idle exploitation is possible because NDDE applications run inside a virtual machine rather than on the user environment. Note that, additionally, this approach enables NDDE applications to run on operating system other than that used interactively.

We have conducted some initial experiments to (i) gauge how much of the idle cycles NDDE can in fact deliver for a parallel or grid application, and (ii) measure its impact on the interactive users. In a nutshell, NDDE can in fact harvests most of the idle cycles and has almost no impact on the interactive user. However, it displays a noticeable overhead for I/O intensive applications.

The reminder of this paper is organized as follows. The next section surveys the state of the art in exploring idle cycles. Then, we introduce NDDE, presenting its features and architecture. Finally, we give a performance overview of this environment and conclude with an outlook on future work.

2. Exploiting Idle Cycles. The use of many resources to tackle a single problem dates back (at least) to the 1970's. The conventional approach since then has been to use dedicated platforms for running parallel applications. These platforms are generally assembled as parallel supercomputers (such as IBM SP2 and Cray T3E) or dedicated Beowulf clusters [3].

On the other hand, there are also applications that can use non-dedicated resources, running opportunistically when resources are idle. Since non-dedicated resources are much cheaper than dedicated resources, much effort has been spent to ease using such idle resources. Therefore, we have seen in the 1980's the introduction of systems such as Condor, which enabled parallel applications to effectively benefit from cycles that

[*]Hewlett-Packard Brazil {reynaldo.novaes, paulo.roisenberg, roque.scheer, caio.northfleet, joao.jornada}@hp.com.
[†]Federal University of Campina Grande. walfredo@dsc.ufcg.edu.br.

would otherwise be wasted. More recently, SETI@home showed that this approach could scale up to planetary proportions.

However, in traditional idle harvesting systems as can be seen in Condor and SETI@home, the guest parallel application runs in the user environment (i.e., as a process in the user operating system). This creates a security concern. Since the parallel application runs as a normal process inside the user's environment, it may be able to exploit some security breaches and cause damage. There are two possible solutions for this problem. The first is to execute the guest application in an emulated platform, like Java. The second is to reboot the machine and run a completely independent operating system from where the guest application has no access to the user environment.

Systems like HP's I-Cluster [4] and vCluster [5] implement a solution based on reboot motivated by security concerns. These systems, upon detecting that there is no user activity, reboot the machine, entering in a different, separated operating system, in which the guest application runs. This approach requires a separated partition to hold the parallel environment and it addresses the security concerns providing a separated operating system and file system, preserving user data. As an extra advantage, the parallel application can run on an operating system different from the one that serves the interactive user. For example, in I-Cluster and vCluster, Windows caters for interactive users, while parallel applications run on Linux.

One drawback of this approach is that it requires a reboot to switch between the two operating systems and this operation has an impact on the interactive user. This is because switching between operating systems is not instantaneous. It takes tens of seconds, in the best case. In order to minimize such an impact, I-Cluster and vCluster keep track of the usage of the machine to try to predict when the interactive user will need it again. This prediction is used to avoid rebooting the machine into cluster mode when the user is expected to go in activity soon, as well as to reboot back into interactive mode in anticipation of the user's need. Of course, any user activity also prompts the switch back to the interactive operating system.

Other systems, which run guest applications concurrently with local user applications like SETI@home, use a different approach for harvesting idle cycles. They monitor user activity using operating system features, like a screensaver.

However, no matter which approach the system uses, it will always try to minimize the impact in the interactive user. Therefore, the prediction of the user idleness is crucial for switching and concurrent approaches. In a perfect world, the user should not notice the exploitation of idle cycles. An issue that complicates matters is that sometimes the user is not interacting with the machine but she is waiting for a task to be completed, like a download. This means that idleness detection mechanism must monitor many of the system's parameters to correctly detect user activity.

Unfortunately, the above approaches impose a limit on how many idle cycles one can harvest. First, for idleness prediction based systems, idle cycles will be really wasted because in order to cause minimal impact on the user, the system has to be somewhat conservative, keeping the system in interactive mode. Secondly, for screensaver-based systems, the user might be interacting with the computer, but using only a small fraction of its processing power (e.g., when the user is typing text) but the system will be seen as active. In both cases idle cycles will be being wasted.

In short, using the screensaver or rebooting the machine to safely exploit idle cycles seems to be effective when there are big chunks of idle time. Such schemes are not effective at harvesting fragmented idle time.

The NDDE addresses these problems. It allows for the safe exploitation of idle cycles, just as I-Cluster and vCluster, but is also able to harvest fragmented idle time, unlike I-Cluster and vCluster. Another feature that distinguishes our approach from the implementations listed above is that, being based on a virtual machine, it can provide a more homogenous execution environment.

3. NDDE. The NDDE is part of a group of projects hosted by HP Brazil that aim to provide simple solutions for exploiting unused computational resources for grid or cluster usage. The target research subjects are non-dedicated computers in corporations and educational institutions. This research includes the development of environment switching processes using reboot or in concurrent mode, like the solution presented here.

NDDE improves upon original I-Cluster and vCluster projects. It presents a different approach to explore idle time, based on the premise that there are unused cycles even when the user is interacting with the computer. The NDDE implements a virtual machine inside the user's system, running a separated operating system that has its own address space and file system. The parallel applications run concurrently with the user's applications. Although the core of this idea (grid computing using virtual machines) is not new, as we can see in paper of

Figueiredo et al. [6], our contribution is that we propose to increase the availability of a non-dedicated machine to the grid or cluster using as many idle cycle as possible, with minor impact on the interactive user's activities.

Using this approach the system does not require any special action, like rebooting, to become an active cluster resource. The guest operating system runs in a user-mode virtual machine, which has restricted access to user's system resources, thus parallel applications can be safely executed. In order to isolate the user's environment, the virtual machine can only access data inside its file system that is entirely contained inside a single file on user's machine. The main advantages on this approach are:

1. The guest environment is isolated from the user environment. The applications running on the guest OS have their own address and storage space and the access to system resource is made through a software layer provided by the virtual machine.
2. There is no noticeable switching time between the two different environments (user and parallel).
3. There is no instruction set conversion, only system calls conversion. So the overhead for CPU intensive applications is minimum.
4. The user does not need to be aware about exploitation of idle cycles. The only requirement may be that the user should leave the machine always turned on.
5. It increases the availability of the node to be exploited as a cluster resource. Any idle time, no matter how small it is, can be used to perform cluster tasks.

Fig. 3.1 shows the basic architecture model for the solution. The virtual machine acts like a native application and runs concurrently with other applications on the user's machine. The virtual machine can be implemented using open source tools like Plex86 [7] or commercial products like VMware [8]. Another option is to use User-Mode Linux for Windows (Umlwin32) [9], but it lacks the security offered by the virtual machine implementations. In all cases, the user machine's resources are shared between the native applications and the virtual machine.

The virtual machine runs its own instance of an operating system, called 'guest system', that provides access to the virtual machine's emulated storage space and controls the use of other resources like virtual memory space and network access. All parallel application accesses to system resources are made through the Host System Call converter, which converts the virtual machine system calls to equivalent host operating system calls. CPU intensive applications (the typical application on parallel environments) run near native machine speed since there is no machine instruction emulation. The parallel applications are loaded in the virtual machine address space and feel as if they are on a dedicated machine. Note also that parallel applications compatible with the guest operating system do not need to be changed or recompiled to run on this environment.

To improve the security, we could also restrict or completely eliminate the virtual machine's ability to access the network. A trusted application on the host OS would be responsible for transferring code and data in and out from the guest file system. This solution is very similar to Entropia [10] that offers a "sandboxed" environment for safe task execution.

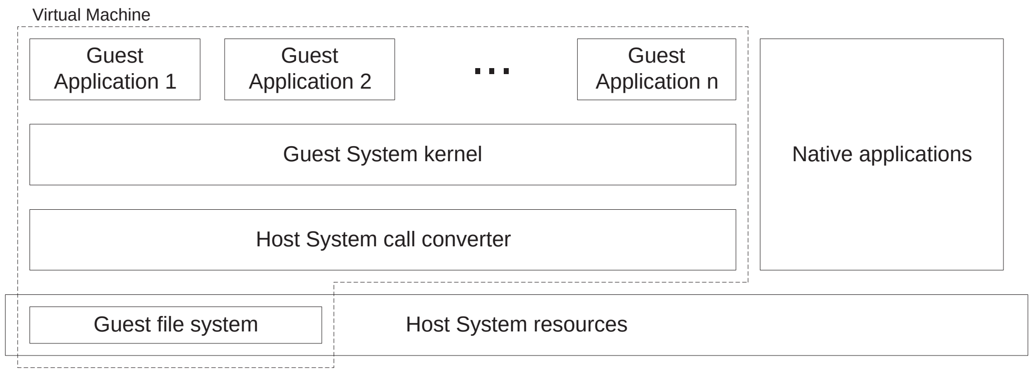

Fig. 3.1. *NDDE basic architecture*

To not interfere with the regular users of the computers, the virtual machines will be made to run as the process with one of the lowest priorities on the system, having only slightly higher priority than the operating system's own "idle" process. This way the virtual machine will be executed by the operating system only when there is no other process or service able to run, but will be chosen to be run by the operating system instead of the operating system's own "idle" process. When there are regular applications running, the NDDE environment, composed by the virtual machine and its own applications, will be automatically preempted and maintained by the host operating system in a "ready-to-run" state, so it can continue to run as soon as there is no regular processes or services running.

4. Performance Evaluation. In order to verify the usability of the NDDE two sets of tests were performed. The first one gauges the performance an application can attain via NDDE. The second one measures the impact on the regular interactive usage of the machine.

4.1. Benchmark Environment. The test environment consisted of a pair of HP e-PC 42, a Pentium 4 1.7 GHz machine with 256MB of memory. The host operating system was Microsoft Windows 2000 Professional. The guest operating system was Linux Red Hat 8.0 and used OpenSSI version 0.9.6r3 [11] as the basic parallel processing environment.

At the beginning two implementations for running the guest environment were considered: Umlwin32 and VMware. The VMware was chosen due to the UMLWin32's early development stage. In the tests described here, the guest operating system runs under VMware Workstation version 3.2.0 configured with 128 MB of memory.

4.2. Performance of Parallel Applications. The tests aim to measure the overall performance NDDE makes available for the guest applications. LLCBench, which is a combined set of synthetic benchmarks, was used to make these tests. It is the combination of BLASBench [12], MPBench [13], and CacheBench [14]. MPBench is used to measure the communication performance of MPI [15]. CacheBench has been chosen to determine the virtual machine's memory subsystem performance. Finally, BLASBench is used to measure the performance of a CPU-bound application.

In order to evaluate the performance impact, a baseline test was performed. These tests, referred as 'Native Linux' in the graphs, use machines executing OpenSSI in native mode, without any emulation.

The idea behind these tests was to verify the performance penalties imposed by this approach, that is, an execution environment running concurrently with a completely different operating system. For sure these tests are generic and only basic usability issues are addressed.

The following tests were performed: one group of tests for memory access simulation, shown in Fig. 4.1, Fig. 4.2 and Fig. 4.3, one test for simulating CPU intensive applications, shown in Fig. 4.4 and, finally, a test regarding the network bandwidth, shown in Fig. 4.5.

These graphs show the average results of each test after several runs.

The Fig. 4.1, Fig. 4.2 and Fig. 4.3 show that VMware has some influence on cache operation that is almost constant for all cache size. We speculate that this performance loss is probably due to page fault handling in the virtual machine but further investigation is required to confirm this theory.

The BLAS performance test, shown in Fig. 4.4, also shows that VMware adds little overhead to the guest environment for CPU intensive applications.

The Fig. 4.5 shows that the network operations suffer noticeable losses imposed by the I/O hardware emulation implemented by virtual machine. This happens because the guest application sees a "double OS" on every access to network devices, that needs to be handled first by the guest OS and later again by the host OS. The same situation happens for all file access, as described by Sugerman et al. [16].

These tests show that the virtual machine solution is best suited for CPU-intensive applications but may not be suited for network or I/O intensive tasks.

4.3. Impact on the User. In order to evaluate the impact of NDDE on normal interactive usage, we elected editing a huge file with Microsoft Word 2000 as archetypical representative of a machine's interactive usage. This huge file had 151MB in size, with 2,623,919 words in 14,211 pages. Considering that we were using a machine with 256 MB of memory and VMware was configured to emulate a machine with 128 MB of memory, this file size (151 MB) is expected to cause Microsoft Word Processor to generate some swapping activity.

In the guest operating system, two test applications were developed. One is a CPU consuming application, which executes a continuous loop. The other application is both CPU and memory consuming. This application allocates 100 MB and executes a continuous loop touching every page by changing the contents of a few bytes on each of them to force the pages to be marked as dirty. So, the guest operating system needs to save their contents to the swap file in case it needs to release pages to make room for user applications.

The tests were grouped in four distinct scenarios:

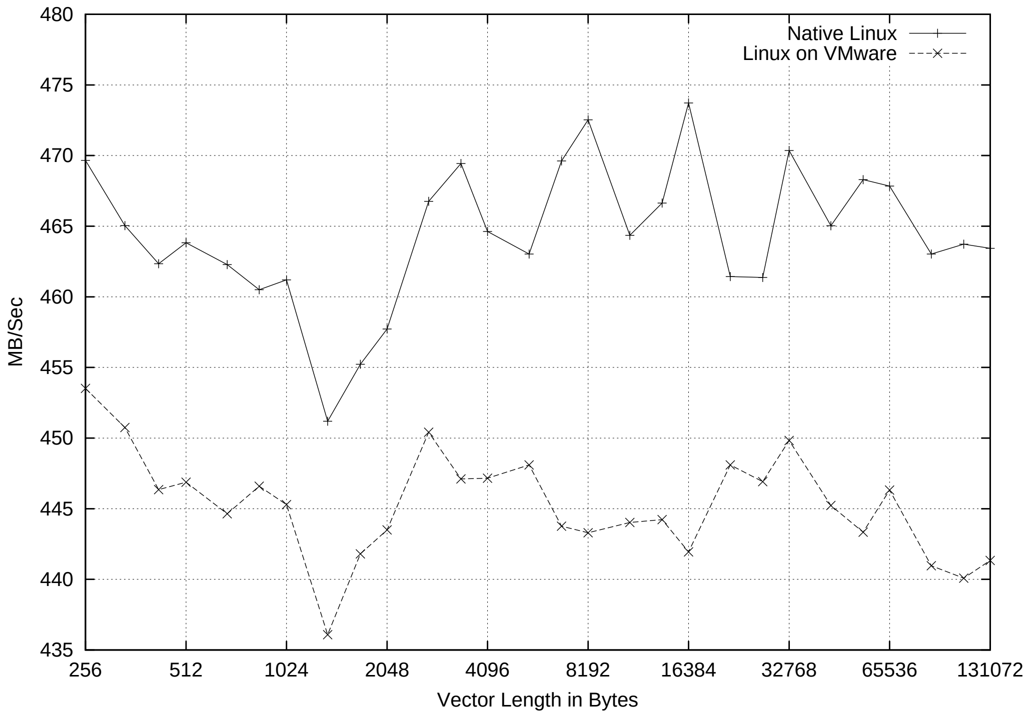

FIG. 4.1. *CacheBench Read Performance*

1. A baseline machine just with the user application (Microsoft Word 2000 editing the 151 MB file)
2. A machine with the same user application and just the guest operating system executing in VMware (no application executing on it)
3. Same as scenario 2, but executing a CPU bound application in the virtual machine
4. Same as scenario 2, but executing a CPU and memory bound application in the virtual machine

In each scenario, four operations were executed:

1. Starting of the application (Microsoft Word) and huge document load
2. Go to the end of the document
3. Select the "statistics" tab option in document properties
4. Replace a character at the end of document and measuring the time to save it

The completion time for each operation was measured, according to Table 4.1.

TABLE 4.1
Average resulting time (min:sec)

	Baseline	VMware only	CPU bound	CPU + memory bound
Load	0:39	0:41	0:41	0:48
Go to end	5:47	5:56	5:57	6:02
Properties	4:18	4:25	4:26	4:26
Save	3:00	3:10	3:12	3:26

The results shown in Table 4.1 are the mean value of several executions. There is only minor impact on the regular user operation for most scenarios, and even the impact of the concurrency for memory was acceptable in this test. Considering the gain of allowing the machine to act as a cluster node concurrently with normal machine operation, the impact on the regular user side seems to be acceptable.

It is interesting to point out that literature reports cases where the competition for memory introduced by guest application cause serious problems for the interactive use of the machine [17]. This would occur when the interactive applications are sleeping and thus can get swapped out to disk when the guest application needs to allocate more memory. We could not reproduce such a behavior. We conjecture that this is due to the

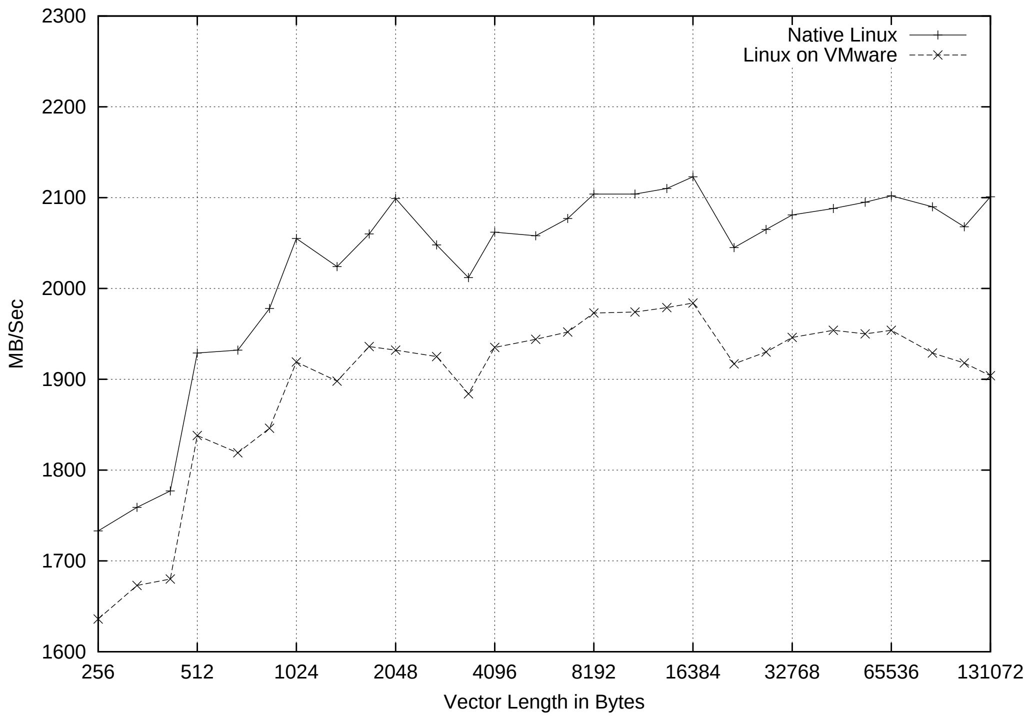

Fig. 4.2. *CacheBench Write Performance*

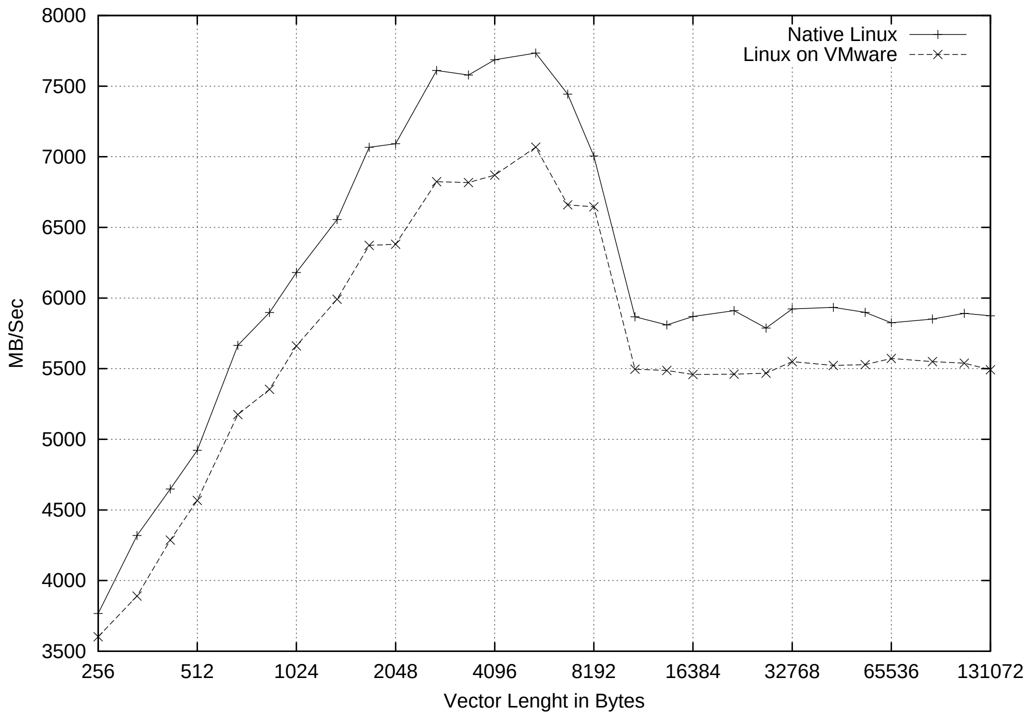

Fig. 4.3. *CacheBench Read/Modify/Write Performance*

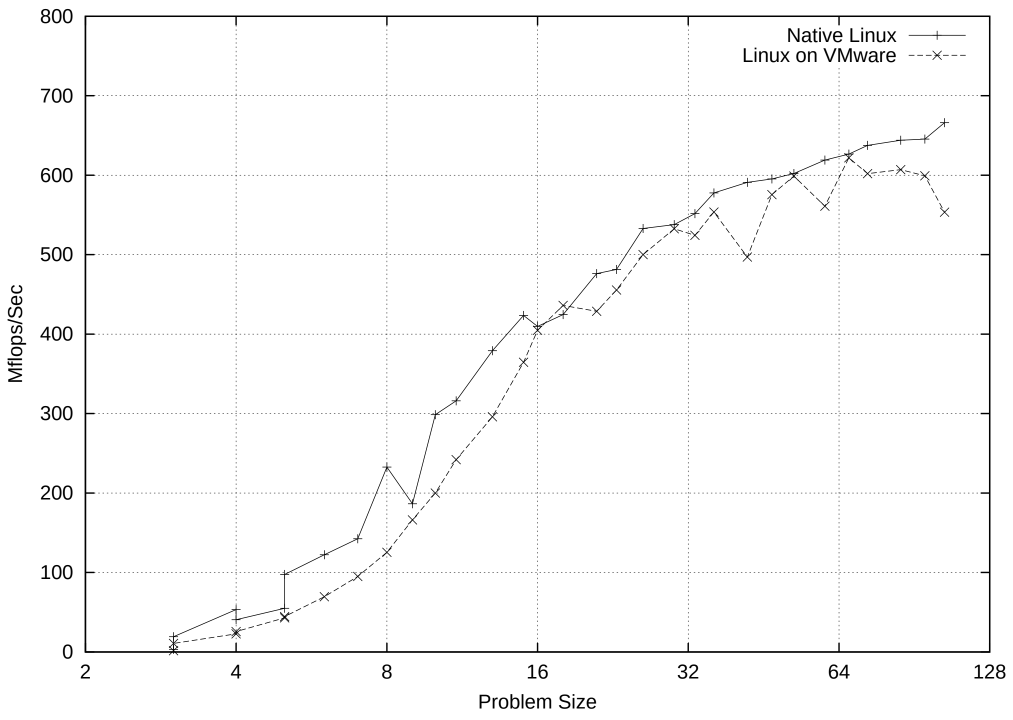

FIG. 4.4. *BLASBench Performance*

fact VMware's memory was limited to 128 MB. In previous experiments reported in the literature, the guest applications had no explicit memory limit.

5. Conclusions. In this article we described NDDE, an alternative way to explore the idle time of interactive computers, turning a set of such computers into a virtual resource for parallel applications and grid computing. NDDE is novel because it allows for safe and continuous use of idle cycles. It is safe in the sense that it is much harder for a malicious guest application to tamper with user data and environment. It is continuous because it can also harvest fragmented idle time. Moreover, since NDDE applications run inside a virtual machine rather than on the user environment, this approach enables NDDE applications to run on an operating system other than that used interactively.

An analysis was carried out to establish the performance of applications that run on NDDE. The results show that NDDE is best indicated when the parallel applications are computationally intensive. Applications that are I/O-intensive may be impacted by the intrinsic limitations of the implementation of virtual machines. The impact on the normal usage of the machine was also measured. The mechanism of using low priority on the virtual machine keeps the impact on the user to a minimum.

The next steps in this work will be going in three directions. First, we intend to evaluate NDDE more thoroughly, refining it where necessary. This includes (i) further investigating the apparent

resilience of NDDE to memory competition with the host applications, (ii) evaluation of the percentage of idle time that is available to be harvested on a typical enterprise or academic network, and (iii) reduce the overhead for parallel applications that are heavily based on internode communications. Some real world data is being collected in order to compare the total of cycles harvested in this solution with a screensaver-based or reboot-based solution.

One relevant result in this work is the performance loss observed in network-bounded applications. This issue motivates us to perform some measurements to determine the application granularity that the guest application should have to make the transference times be acceptable.

All the investigation topics described above will help us to see if this solution is profitable when compared to dedicated and switched environments. Second, it might be worthwhile to combine NDDE's and I-Cluster's approach into a hybrid scheme. For example, most machines are totally idle during the night. We could thus

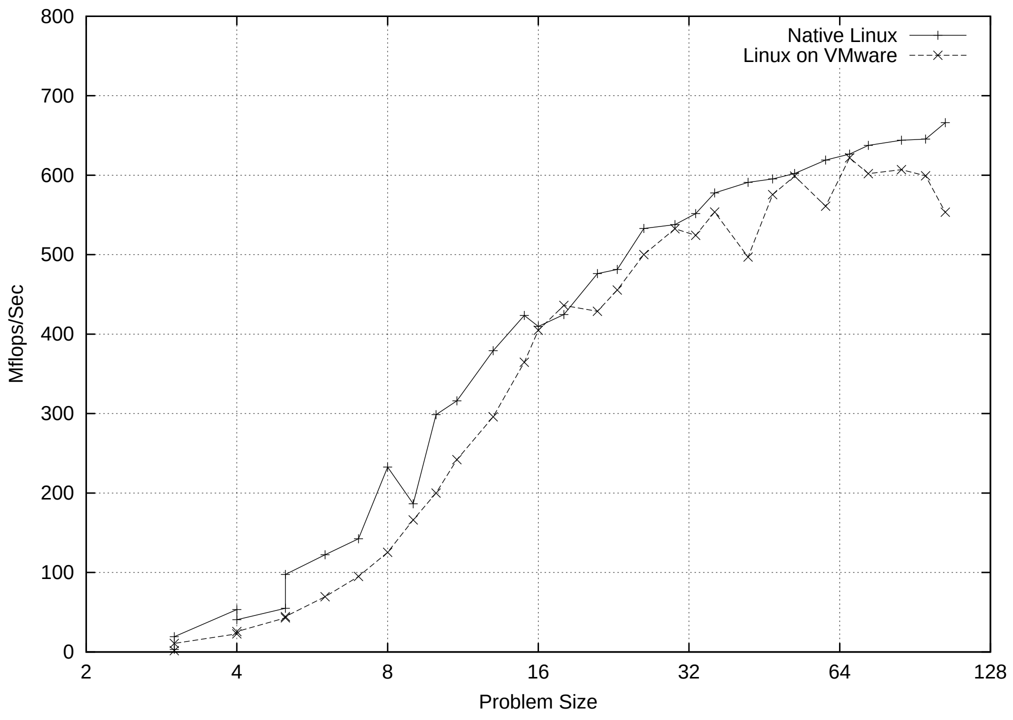

FIG. 4.5. *MPBench Bandwidth*

think of using I-Cluster during the night and NDDE during the day. Third, we intend to explore NDDE as a sandboxing platform for MyGrid [18], enabling grid applications that cross administrative boundaries. Note that such grid applications raise especially serious security issues, making the use of NDDE technology particularly relevant.

Acknowledgements. The authors wish to express their sincere thanks to the AGridM'03 reviewers for their insightful comments.

REFERENCES

[1] M. LITZKOW, M. LIVNY AND M. MUTKA, *A Hunter of Idle Workstations*, Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104–111, June 1988.
[2] D. ANDERSON, J. COBB AND E. KORPELA, *SETI@home: An Experiment in Public-Resource Computing*, Communication of the ACM, vol. 45, no. 11, pp. 56–61, November 2002.
[3] D. RIDGE, D. BECKER, P. MERKEY AND T. STERLING, *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs*, Proceedings of IEEE Aerospace, 1997.
[4] B. RICHARD AND P. AUGERAT, *I-Cluster: Intense computing with untapped resources*, MPCS'02, Ischia, Italy, April 2002.
[5] C. A. F. DE ROSE, F. BLANCO, N. MAILLARD, K. SAIKOSKI, R. NOVAES AND B. RICHARD, *The virtual cluster: a dynamic environment for exploitation of idle network resources*, Proceedings of 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), pp. 141–148, Vitï£¡ia, ES, Brazil, 2002.
[6] R. J. FIGUEIREDO, P. A. DINDA AND J. A. B. FORTES, *A Case for Grid Computing on Virtual Machines*, Proceedings of International Conference on Distributed Computing Systems (ICDCS), April 2003.
[7] K. LAWTON, *The new Plex86 x86 Virtual Machine Project*, WWW, August 2003. http://plex86.sourceforge.net/.
[8] VMWARE, *VMware Workstation - Powerful Virtual Machine Software for the Technical Professional*, WWW, April 2003. http://www.vmware.com/pdf/ws_specs.pdf.
[9] C. KUDIGE, *Umlwin32*, WWW, March 2003. http://umlwin32.sourceforge.net/.
[10] A. A. CHIEN, B. CALDER AND S. ELBERT, *Entropia: Architecture and Performance of an Enterprise Desktop Grid System*, Journal of Parallel Distributed Computing, vol. 63, no. 5, pp. 597–610, May 2003.
[11] B. J. WALKER, *OpenSSI Linux Cluster Project*, WWW, April 2003. http://openssi.org/ssi-intro.pdf.
[12] NATIONAL SCIENCE FOUNDATION, *BLAS (Basic Linear Algebra Subprograms)*, WWW, March 2003. http://www.netlib.org/blas/.
[13] P. J. MUCCI, K. LONDON AND J. THURMAN, *The MPBench Report*, November 1998. WWW, March 2003. http://icl.cs.utk.edu/projects/llcbench/mpbench.pdf.

[14] P. J. Mucci, K. London and J. Thurman, *The CacheBench Report*, November 1998. WWW, March 2003. http://icl.cs.utk.edu/projects/llcbench/cachebench.pdf.
[15] Message Passing Interface Forum, *MPI: A Message Passing Interface Standard*, May 1994.
[16] J. Sugerman, G. Venkitachalam and B. Lim, *Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor*, Proceedings of the USENIX Annual Technical Conference, June 2001.
[17] T. E. Anderson, D. E. Culler, D. A. Patterson and the NOW Team, *A Case for Networks of Workstations: NOW*, IEEE Micro, February 1995.
[18] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvï£¡ F. A. Barbosa da Silva, C. Osthoff Barros and C. Silveira, *Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach*, Proceedings of the ICCP'2003 - International Conference on Parallel Processing, WWW, October 2003. http://walfredo.dsc.ufpb.br/resume.html#publications.

Edited by: Wilson Rivera, Jaime Seguel.
Received: July 9, 2003.
Accepted: September 1, 2003.