Performance Comparison of Two Virtual Machine Scenarios Using an HPC Application

A Case study Using Molecular Dynamics Simulations

Anand Tikotekar, Hong Ong, Sadaf Alam, Geoffroy Vallée,
Thomas Naughton, Christian Engelmann & Stephen L. Scott *

Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN, USA.

Abstract

Obtaining high flexibility to performance-loss ratio is a key challenge of today's HPC virtual environment landscape. And while extensive research has been targeted at extracting more performance from virtual machines, the idea that whether novel virtual machine usage scenarios could lead to high flexibility Vs performance trade-off has received less attention.

We, in this paper, take a step forward by studying and comparing the performance implications of running the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) application on two virtual machine configurations. First configuration consists of two virtual machines per node with 1 application process per virtual machine. The second configuration consists of 1 virtual machine per node with 2 processes per virtual machine. Xen has been used as an hypervisor and standard linux as a guest virtual machine. Our results show that the difference in overall performance impact on LAMMPS between the two virtual machine configurations described above is around 3%. We also study the difference in performance impact in terms of each configuration's individual metrics such as CPU, I/O, Memory, and interrupt/context switches.

* ORNL's work was supported by the U.S. Department of Energy, under Contract DE-AC05-00OR22725.

1. Introduction

High-performance computing is increasingly leveraging virtualization due to its flexibility, such as the ability to perform fault-tolerance using live migration, the ability to efficiently allocate resources, the customizability according to application requirements, etc. Yet, such flexibility cannot often offset the performance overhead of using virtualization solutions. To alleviate this trade-off between the flexibility offered and the generated performance hit, many in the HPC community have looked at solutions that optimize hypervisors, kernels, I/O, etc. And while this vertical optimization is certainly helpful, no such extensive research is being extended to the efficacious use of virtual machines themselves. After all, virtual machines are nothing if not malleable. Therefore, it seems reasonable to think that a novel configuration of virtual machines could end up providing a favorable trade-off between flexibility and the performance impact. For example, based on an application's communication pattern, communicating virtual machines could be kept on the same nodes to free more nodes as spare nodes for fault-tolerance. Another configuration design option is to decide whether to create a separate domain for a monitoring thread or add the thread to an existing domain. Many other virtual machine configurations such as efficient resource allocation of memory may be costly from a performance point of view due to additional memory pressure, but can provide certain flexibility by allowing to host another light-weight virtual machine that uses idle time of the running application for check pointing. The idea thus is to employ a virtual machine configuration for which the "ratio of flexibility" offered to the performance impact is high. Two logical questions arise: One, how to quantify the difference between the flexibility offered by two virtual machine configurations? Second, what, if any, is the performance impact between two virtual machine configurations? Please note that all the discussion pertains to a fixed run of an application instance.

In this paper, we try to tackle the simpler of the two questions, which is about the performance impact. We have selected the following two virtual machine configurations. Configuration 1 consists of two virtual machines per node with 1 application process per virtual machine. Configuration 2 consists of 1 virtual machine per node with 2 processes per virtual machine. Please note that we are not claiming that these configurations are suitable for this instance of LAMMPS application. We only want to highlight the difference, if any, in the performance impact between these configurations as they do the same amount of work.

Our contributions in this paper are the following: We advocate the idea that virtual machines could be used in novel ways so as to benefit the flexibility versus performance impact equation. Second, we study the performance impact of two virtual machine configurations on LAMMPS (12), and report the overall as well detailed performance impact of each configuration.

2. Related Work

Researchers in HPC domain have mainly focused on bringing down the performance overhead of virtualization. Towards this end, the research is focused on efficient hypervisors such as Xen (2). Efficient I/O solutions are advocated in (6), (8), (17) that seek to minimize the I/O overhead of virtualization. Many other research efforts focus on using lightweight domains (16), microkernels (3), and lightweight kernels (15) to reduce the cost of the hypervisor itself. Many studies have evaluated the performance impact of virtualization on HPC applications. Such studies include (18), (9), (5). The common conclusion is that Xen is efficient in executing HPC applications.The flexibility aspect of virtualization is mainly exploited for fault tolerance (10), (11) in HPC.

Grid computing has leveraged various usage and execution models of virtual machines. Vmplants (7) provides automatic configuration of virtual machines to meet various application requirements. Other studies for resource provisioning in grids include (14).

The idea of determining whether novel virtual machine usage scenarios, configurations could lead to better flexibility vs performance-loss ratio in HPC has not been explored in detail. Authors in (13) study whether intelligent managent of virtual machines could improve the utilization of physical resources or not. Authors in (4) compare their distributed system MemX on three modes within Xen vm. These modes include driver domain, and individual guest domain. In this paper we compare two virtual machine configurations that are similar to each other. In the paper (1), authors study the network processing overheads in Xen on two configurations. Their configurations include running I/O service vm and the guest vm on the same CPU, and on a different CPU. The authors are interested in comparing the network performance in these two modes with the native mode. Our study deals with determining if there is any impact on an HPC application when subjected to two similar virtual machine configurations.

3. Experimentation Description

3.1 Hardware and Software

We have used Xen 3.0.4 and linux kernel version 2.6.16.33 with FC5 distribution and NFS filesystem. Our cluster consists of 16 nodes each with 2GHz processor with 768MB RAM. Both of our virtual machine configurations use Xen as their hypervisor and para-virtualized linux as their guest operating system. Xen is responsible for domain scheduling, aggregating domain I/O requests and responses, memory handling between domains, etc. Thus, it is easy to see that Xen operates at domain level. Linux guest machines, on the other hand, are responsible for their processes, their internal memory, the aggregation of internal I/O requests and responses. Therefore, guest machines are responsible for their own affairs, and yet are dependent on Xen for their domain as a whole.

3.2 Methodology

Our methodology for the experiment consists of one VMstat process that runs per second per virtual machine. VMstat measures all the above metrics that we want to study. Further, since vmstat is based inside a virtual machine, its view is limited. Specifically, it does not have information on other virtual machines or domains including the privileged domain. Because, vmstat inside a guest vm does not have information on the privileged domain, metrics reported by vmstat such as system utilization, I/O blocks only reflect how the guest vm receives and sends information. But since we are comparing two virtual machine scenarios as described above, the comparison is valid. Furthermore, a virtual machine containing the vmstat process does not have any information about the events such as real interrupts, including I/O interrupts, timer interrupts etc. The data collected by the vmstat process is only based on the virtual events delivered to it by the underlying hypervisor, in this case, the xen hypervisor. However, a guest virtual machine does have knowledge about the real as well as virtual time in the case of xen. This knowledge about the real time allows each virtual machine to respect the periodic time, in our case 1 second, of the vmstat process. Apart from the real time, both configurations treat xen as a black box when gathering data through vmstat.

3.3 The Two Scenarios

The first scenario which consists of running two virtual or guest linux machines -with each machine executing one application process- on one physical node which runs Xen as the hypervisor, gives more control to Xen in handling the execution of given application processes. Xen, in this case, is responsible for domain scheduling, domain I/O request response, memory management of the domains etc. This

setup is replicated across 8 physical nodes so as to have 16 virtual machines and 16 LAMMPS application processes.

On the other hand, the second scenario consists of running one virtual machine per node with two LAMMPS application processes. This scenario gives more control to the linux guest machine in which the processes are executing. Specifically, the virtual machine is in charge of handling the I/O request aggregation, internal memory management of the two processes, scheduling between the processes, etc. This setup is replicated across 8 virtual machines on 8 physical nodes so as to have 16 processes.

The comparison of these scenarios is valid only when both scenarios are given the same amount of application work to perform. Thus, in our experiment design, we have tasked both scenarios with performing the same amount of LAMMPS computing, which means the same number of atoms and the interactions between the atoms. Moreover, we have allocated the same total resources to each of our scenarios. For instance, in both situations, each application process will require the same amount of memory (231MB). Please also note that from a particular physical node's point of view, it is executing two processes with the approximately same total resources per node to manage no matter what virtual configuration is on the node. And therefore, it is not obvious to state which of the two configurations is beneficial in terms of its performance impact.

Further, our experiment is not just intended to evaluate the performance overhead of running applications on virtual machines. Many papers including few of ours have established that linux guest machines hosted by Xen are quite efficient in executing HPC applications (Please see section on related work). *The purpose of this paper is to highlight differences, if any, in the performance impact between the two virtual machine configurations on LAMMPS.*

4. Results and Analysis

In section 3, we discussed our virtual machine configurations. This section describes the overall as well as detailed performance impact of these two configurations on LAMMPS. To this end, we want to investigate key metrics such as CPU utilization, memory and swap allocation, I/O movement, and system metrics such as context switches, interrupts, etc.

4.1 Overall Performance

The question of overall performance is answered by Table 1. It shows that configuration 2 (scenario 2) is slightly more efficient in terms of wall clock time taken to complete a given run of LAMMPS. The wall clock time values are averaged over 5 runs of each configuration. The standard deviation in configuration 1 was 3%, while the standard deviation for configuration 2 was 1.5%.

Virtual confs	Wall clock time in seconds
2 VMs/node 1 MPI task/VM	1686
1 VM/node 2 MPI tasks/VM	1646

Table 1. Overall performance impact of the two scenarios

4.2 CPU Metrics

In this subsection, we present statistics on 5 CPU metrics namely user, system, wait time, idle, and stolen time. All numbers for CPU metrics are in percentages. Figures 1(a) and 1(b) represent configuration 1, which runs two virtual machines per node with 1 application process per VM. Figure 2 represents configuration 2, which runs two application processes per VM with one VM per node.

The average CPU time distribution percentage numbers for the virtual configuration 2 across user, system, wa, idle, and stolen time are 63.9%, 9%, 0.7%, 20.4%, 6% respectively. Please note that the virtual machine configuration 2 runs 2 application processes per VM/per node. The numbers in the case of configuration 1 are the following: 27.4%, 1%, 4.1% , 39.2%, 28.3% for the first VM and 26.4%, 1.2%, 4.5% ,39.5%, 28.4% for the second VM. The stolen time for each vm in each configuration represents the time when the vm wanted to run but was not scheduled. The idle time, on the other hand, represents the voluntary wait by the vm. To compare the cpu metrics of configuration 1 with configuration 2, we must combine the cpu metrics of the two virtual machines in configuration 1. To combine the two set of metrics for configuration 1, we can use the following argument. First, the user and system time of each vm in configuration 1 can simply be added as the numbers represent the real time (Since the stolen time is removed from the numbers). Thus, the combined user and system time become 53.8%, and 2.2% respectively. We can not add the idle time and I/O wait times as they contain some busy time of other domains. Second, since the busy time of configuration 1 is 56% (user + system), the idle time is 44% as a whole for configuration 1. Third, to get the I/O wait time out of the idle time, we can use the proportion of the I/O wait time with respect to the idle time of an individual vm of configuration 1. This way the I/O wait time is estimated to be 2.6% for configuration 1 and the remaining time, that is 41.4% as idle time. Therefore, the numbers for configuration 1 are 53.8%, 2.2%, 2.6%, 41.4% and for configuration 2 are 63.9%, 9%, 0.7%, 26.4% (idle + stolen time). Next, we study the cpu metrics individually for both configurations.

The first number is the CPU user utilization. The cpu user utilization for configuration 1, which runs two virtual machines on a physical node with each virtual machine running one process, is 53.8%. Similarly, configuration 2 reports the number at 63.9%. A quick comparison shows that from a physical node's perspective, which holds the CPU, there is significant difference. Yet there is not much

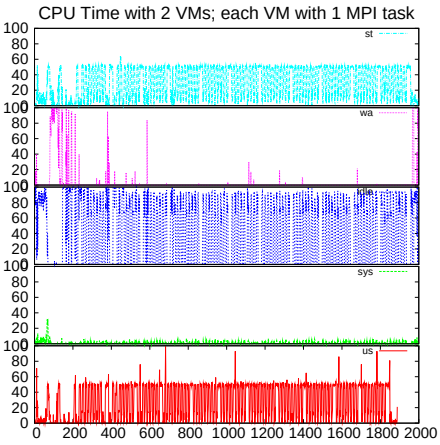
difference in the overall performance. There could be two reasons. First, in configuration 1, it is possible that xen is not able to exploit the idle time very effectively because of the lack of guest process level knowledge. (It relies on the guest to block and hand over the control) Second, in configuration 2, L2 cache and TLB misses could contribute to the higher user utilization.

The second number from each set represents the system utilization. As can be seen, configuration 2 has much higher CPU system utilization than configuration 1. This may be because of the number of context switches (please refer to system metrics sections, where configuration 2 shows some spikes in the number of context switches). Moreover, virtual machine in configuration 2 has to schedule two linux processes, whereas, in configuration 1, from a virtual machine's point of view only one process needs to be scheduled, while xen has to schedule two domains. Please note that the system utilization number in configuration 1 does not include the time between domain switches, and domain scheduling by xen. Furthermore, in configuration 2, due to high number of context switches, a high TLB cost is also possible. The third number reports an interesting difference between configuration 1 and 2 with respect to the disk wait time. Since configuration 1 has two virtual machines running, it has more disk I/O and NFS mount pressure, which can be seen in its wait time numbers. We will further discuss the I/O issue in the I/O metrics section, but it can be seen that the wait times are in part a reflection of how the aggregation of disk I/O requests occurs. In case of configuration 2, for instance, the aggregation of I/O requests is possible. Further, higher disk wait time also implies higher idle time which is supported in configuration 1.

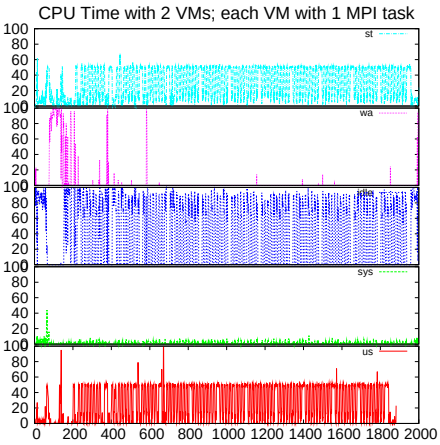
4.3 Memory Metrics

In this subsection, we describe the relevant memory metrics. Specifically, we discuss four memory metrics: active memory, inactive memory, free memory, and the amount of virtual memory used. Figures 3(a) and 3(b) represent configuration 1 with default memory, which is 256MB per virtual machine. Similarly, Figure 4(a) represents configuration 2, which runs 1 virtual machine with 2 LAMMPS processes with 512MB of memory. These figures describe how different memory metrics have changed with time, but it does not show average values for the memory metrics. Therefore, we calculated the average values for these metrics. For configuration 1, we combined the averages of the two virtual machines to get a single set of numbers, so that we could compare them with our second configuration (it may be noted that the memory profiles of the two virtual machines in configuration 1 are similar, as we would expect). The total average virtual memory allocated for any virtual machine is given by:

$$AvailableRAM + Swapcommited - Freememory \quad (1)$$



(a) First Virtual Machine



(b) Second Virtual Machine

Figure 1. CPU behavior for configuration 1

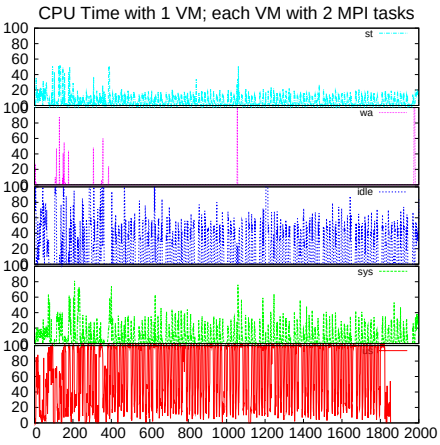


Figure 2. CPU behavior for configuration 2

Thus, the total average memory per virtual machine for first configuration is 285.57MB, which is much greater than 256MB of RAM allocated. Similarly, in the second configuration, the average total memory commited is 509.68MB, which is slightly less than 512MB allocated. Please note that while the application memory required in each configuration is the same, the actual higher memory allocated in configuration 1 as compared to configuration 2 could be due to the management of the resources. To refresh, application memory in configuration 1 is 231MB, and in configuration 2 it is 462MB (231*2).

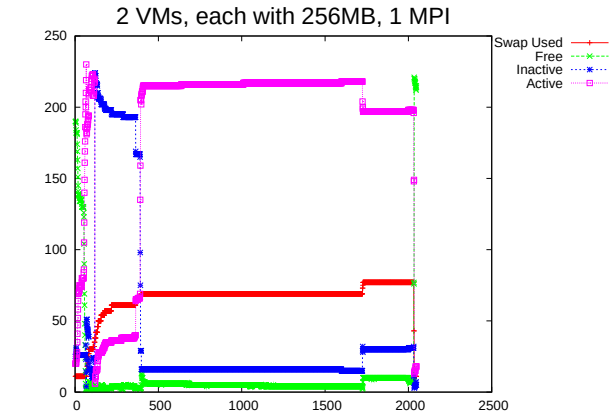
Out of 285.57MB of memory commited/allocated in our first configuration, the four metrics namely swap, free, inactive, and active consume 20.15%, 3.05%, 18.69%, 57.60% respectively. Similarly, for configuration 2, the numbers are 8.09%, 3.93%, 26.13%, and 61.83%. It is easily seen that configuration 2 commits much less proportion of swap or virtual memory as compared to the first configuration, which is clearly confirmed by the fact that configuration 1 has more memory pressure per virtual machine (Actual 256 versus 285 required for configuration 1, as opposed to 512 actual versus 509 required for configuration 2). The reason behind greater memory pressure in configuration 1 compared to configuration 2 is due to the fact that kernel is shared between the two processes in configuration 2. Please note that we use the word "allocated/commited" for swap memory because of the fact that it was never actually used. This information is extracted from the page in/page out columns of the vmstat data. The page in/page out numbers are 0, and therefore not shown here. Furthermore, the numbers reported here are averages, and thus the simultaneous existence of free memory and swap memory usage should not be construed as an error. Other obvious observation is that both configurations have the same average free memory proportion.

Next, we added 20% more memory to each virtual machine in both configurations to observe the effect of swap commit usage, and subsequently the impact on other three metrics. Figures 3(c) and 3(d) represent configuration 1 when 20% more memory is added per virtual machine. Similarly, figure 4(b) describes configuration 2 with 20% more memory. Both configurations now show the expected pattern, that is the usage of swap is zero.

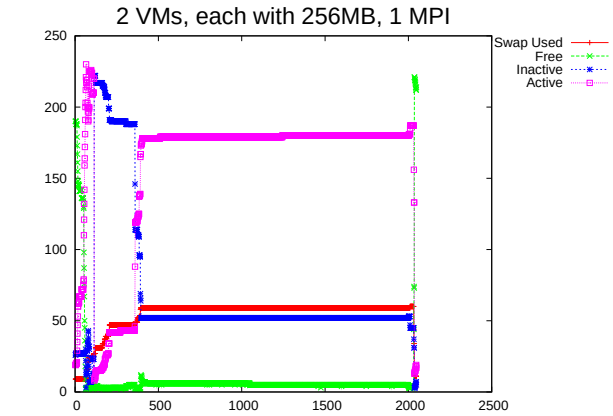
4.4 I/O Metrics

Figures 5(a) and 5(b) represent configuration 1, which runs two virtual machines with one process per VM. Similarly, Figure 6 represents configuration 2, which runs one VM with 2 processes per VM.

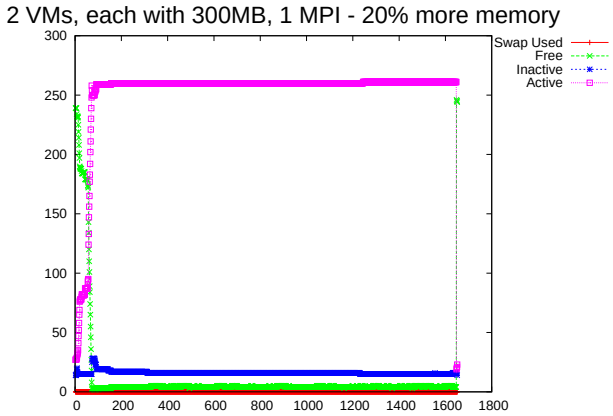
As can be seen from these figures, the two virtual machines in configuration 1 report much more I/O activity than configuration 2. Specifically, On average, each virtual machine in configuration 1 reports that 25.76 blocks were received and 32.76 blocks were sent per second from the disk device. Since these are absolute numbers, we should double them so that we could compare configuration 1 with config-



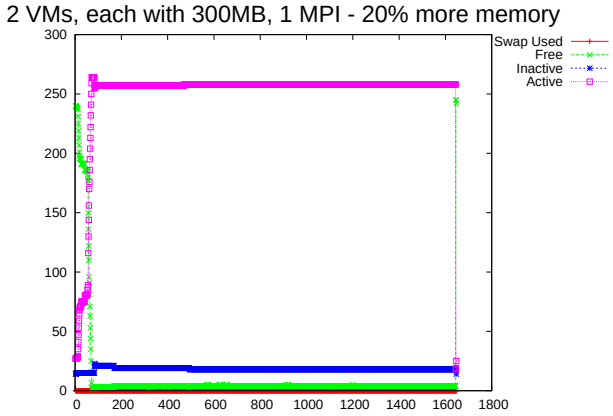
(a) First VM - Default memory



(b) Second VM - Default memory

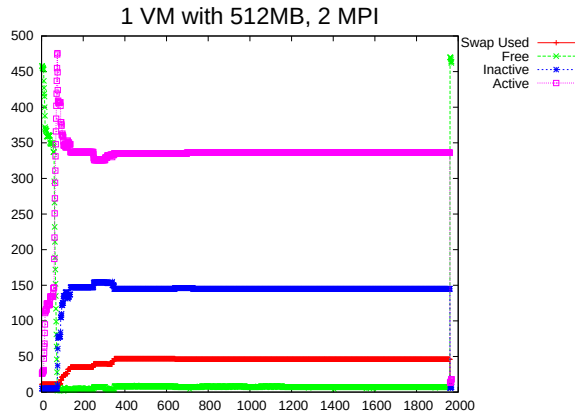


(c) First VM - 20% more memory

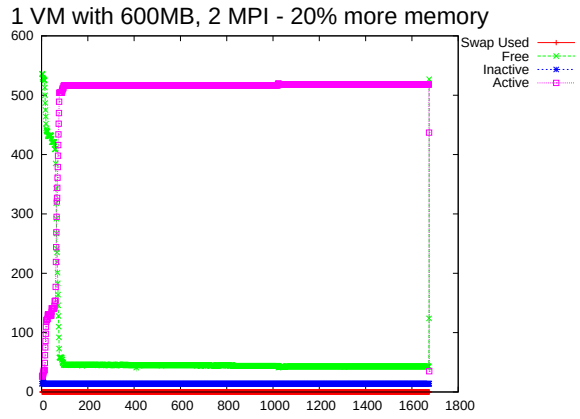


(d) Second VM - 20% more memory

Figure 3. Memory behavior for configuration 1

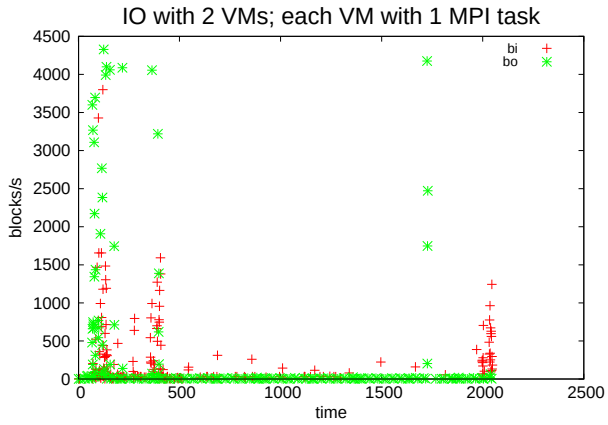


(a) Default memory

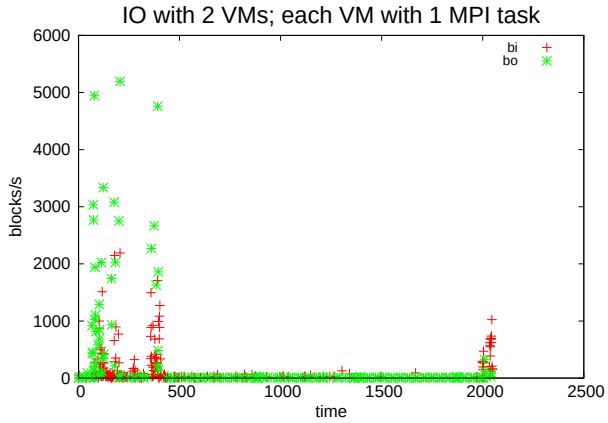


(b) 20% more memory

Figure 4. Memory behavior for configuration 2



(a) I/O behavior of the first VM



(b) I/O behavior of the second VM

Figure 5. I/O behavior of the First Configuration

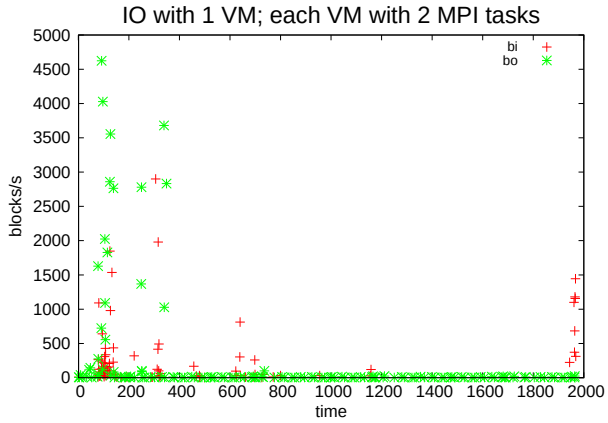
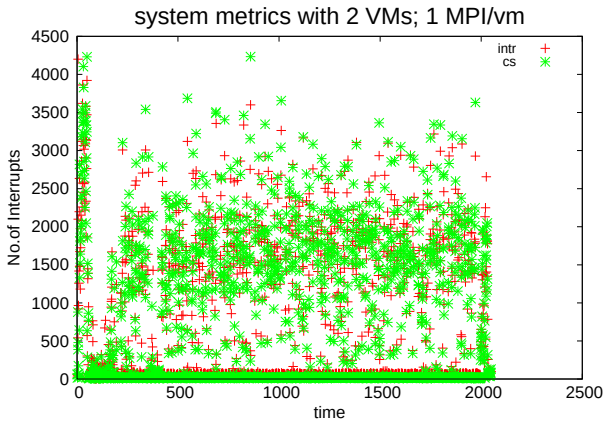


Figure 6. I/O behavior of the Second Configuration

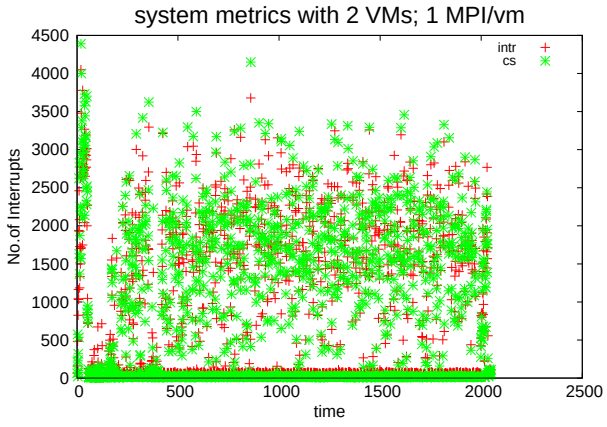
uration 2. Therefore, 51.52 blocks were received and 65.52 blocks were sent per second by configuration 1 as a whole. By contrast, configuration 2 reports that 13.22 blocks, on average, were received per second and 19.99 blocks were sent per second to the disk device. One reason behind why configuration 1 reports much stronger I/O activity could be due to less efficient aggregation of I/O requests and responses by Xen as opposed to the linux virtual machine. However, on average, the difference between the number of blocks received and the number of blocks sent per second remains very similar in both configurations, which is around 7 blocks/second.

4.5 System Metrics

Figures 7(a) and 7(b) represent configuration 1. Similarly, Figure 8 represents configuration 2. All these figures show the number of system interrupts and the number of context switched made. Configuration 1 shows that two virtual machines have almost identical behavior of system interrupts and context switches, which we would expect. The average numbers of system interrupts for two virtual machines are 680.7 and 723.06 respectively. Similarly, the numbers



(a) First VM



(b) Second VM

Figure 7. System Behavior for configuration 1

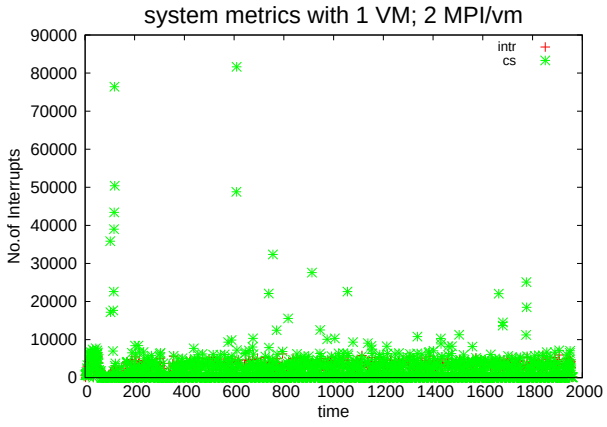


Figure 8. System Behavior of the Second Configuration

for context switches are 651.14 for first virtual machine and 681.7 for the second virtual machine. Therefore, for configuration 1 as a whole, the average numbers for interrupts and context switches are 701.88 and 666.42 per second respectively. The average numbers for configuration 2 are 1357.53 and 1698.1 per second for interrupts and context switches respectively. Interestingly, configuration 2 shows some spikes for the number of context switches, but the average number of context switches in configuration 2 remains low. It can be easily seen that by doubling the average numbers in configuration 1, we get close to configuration 2. However, the doubling exercise clearly shows that while the number of interrupts in configuration 1 is more than configuration 2, the opposite is true for context switches.

Further, while the figures show that the average numbers of interrupts and context switches are similar in both configurations, they do not follow a particular order. This may be due to following scenarios: Not every interrupt causes a context switch (such as if the Interrupt Descriptor Table (IDT) entry does not point to a task-gate) and not every context switch is associated with an interrupt (such as if a task yields). Further, it is possible that certain interrupts may have been masked and serviced later, or an interrupt may cause multiple context switches. Please also note that, the VMstat is configured to report every 1 second, and therefore, it is possible that the boundary interrupts and context switches may not be aligned.

5. Discussion

In this section, we try to fit the four metrics (CPU, memory, IO, and system) together into a big picture. First, we can see that the percentage of CPU user time in configuration 2 is much higher than in configuration 1, which suggests that configuration 2 is efficient in terms of executing application code. Yet, the wall clock times of configuration 1 and 2 do not reflect the higher difference between the user cpu utilization. This could be due to the fact that the user code weight may not be very high as compared to the system code weight. Second, although configuration 2 has a higher percentage of CPU utilization in general, which is seen by the idle numbers, it also has a higher CPU system utilization. The higher system utilization in configuration 2 may be related to the higher cost of context switching, TLB and cache misses. Third, application I/O behavior in configuration 1 clearly reflects a far greater activity than configuration 2. This is confirmed by the higher disk wait time (wa) percentage in configuration 1, which is almost 4 times higher than configuration 2. Higher I/O activity may also explain why configuration 1 requires higher memory allocation.

From a flexibility standpoint, configuration 1 offers more flexibility such as the ability to move virtual machines to spare nodes in the case of faults without disrupting the application. This added flexibility of configuration 1 could be preferred to configuration 2, even though configuration 1

has more performance penalty than configuration 2. This is the crux of our study

6. Conclusion

In this paper, we have shown that in terms of wall clock time, the difference in overall performance is 3%, with configuration 2 being more efficient. Yet, the four metrics that we have studied reflect different results. CPU usage results are interesting in terms of the disk wait, CPU idle numbers, and user code utilization. Memory allocation/usage is also dissimilar in two configurations. Further, I/O behavior shows a markedly different behavior in one configuration with respect to another. Finally, system interrupts and context switches are similar on average with few spikes for context switches in configuration 2. To sum up, our study provides details about the impact of virtualization on HPC applications such as LAMMPS. The study specifically sheds light on how different virtual configurations impact the overall performance, and also how the configurations impact the individual performance metrics. Further the study finds that there is some evidence to suggest that a linux virtual machine handles the application and its resources better than Xen does. Moreover, a thematic implication of this study is that application containers, such as linux virtual machines, could be given more control than hypervisors such as Xen, in handling applications and their resources for efficiency purposes.

References

[1] Padma Apparao, Srihari Makineni, and Don Newell. Characterization of network processing overheads in xen. In *VTDC '06: Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing*, page 2, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating System s Principles (SOSP19)*, pages 164–177. ACM Press, 2003.

[3] D. R. Engler, M. F. Kaashoek, and J. O'Toole, Jr. Exokernel: an operating system architecture for application-level resource management. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 251–266, New York, NY, USA, 1995. ACM.

[4] Michael R. Hines and Kartik Gopalan. Memx: supporting large memory workloads in xen virtual machines. In *VTDC '07: Proceedings of the 3rd international workshop on Virtualization technology in distributed computing*, pages 1–8, New York, NY, USA, 2007. ACM.

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

[6] Wei Huang, Matthew J. Koop, Qi Gao, and Dhabaleswar K. Panda. Virtual machine aware communication libraries for high performance computing. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.

[7] Ivan Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, and Renato J. Figueiredo. Vmplants: Providing and managing virtual machine execution environments for grid computing. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 7, Washington, DC, USA, 2004. IEEE Computer Society.

[8] Jiuxing Liu, Wei Huang, Bulent Abali, and Dhabaleswar K. Panda. High performance vmm-bypass i/o in virtual machines. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 3–3, Berkeley, CA, USA, 2006. USENIX Association.

[9] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoe. Diagnosing performance overhead in the xen virtual machine environment. In *Proceedings of the 1st ACM Conference on Virtual Execution Environments*, June 2005.

[10] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L. Scott. Proactive fault tolerance for HPC with Xen virtualization. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32, New York, NY, USA, 2007. ACM Press.

[11] Dan Pelleg, Muli Ben-Yehuda, Rick Harper, Lisa Spainhower, and Tokunbo Adeshiyan. Vigilant: out-of-band detection of failures in virtual machines. *SIGOPS Oper. Syst. Rev.*, 42(1):26–31, 2008.

[12] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1):1–19, 1995.

[13] Fernando Rodríguez, Felix Freitag, and Leandro Navarro. On the use of intelligent local resource management for improved virtualized resource provision: challenges, required features, and an approach. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pages 24–31, New York, NY, USA, 2008. ACM.

[14] P.; Dongyan Xu Ruth, P.; McGachey. Viocluster: Virtualization for dynamic computational domains. In *Cluster Computing, 2005. IEEE International*, pages 1–10. IEEE Computer Society, 2005.

[15] Ron Brightwell Suzanne Kelly. Software architecture of the light weight kernel, catamount. *In 47th Cray User Group (CUG 2005)*, 2005.

[16] Samuel Thibault and Tim Deegan. Improving performance by embedding hpc applications in lightweight xen domains. In *HPCVirt '08: Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, pages 9–15, New York, NY, USA, 2008. ACM.

[17] Jian Wang, Kwame-Lante Wright, and Kartik Gopalan. Xenloop: a transparent high performance inter-vm network loopback. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 109–118, New York, NY, USA, 2008. ACM.

[18] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Paravirtualization for HPC Systems. In *ISPA Workshop on XEN in HPC Cluster and Grid Computing Environments (XHPC'06)*, pages 474–486, December 2006.