

The Effect of Virtualization on OS Interference

Eric Van Hensbergen
(bergevan@us.ibm.com)

IBM Research

Abstract

This document evaluates the impact of virtualization technology on the degree of operating system interference experienced by high-performance computing applications. We overview the IBM Research Hypervisor and describe an infrastructure enabling the use of logical partitions (LPARs) for the execution of stand-alone applications along side traditional operating systems. We use a simple micro-benchmark to compare the operating system interference in this environment to that of a traditional Linux system. We then discuss extensions of virtualization technology which can be used to manage this OS interference and improve the efficiency of parallel applications.

1. Introduction

Virtualization technology has existed since the early days of computer science [Singh04], providing mechanisms to safely partition larger mainframes into discrete virtual machines. There has been much recent interest in applying virtualization technology to provide an easy means of partitioning commodity clusters within data centers to efficiently multiplex hardware resources while providing security and quality of service guarantees to customers. This has resulted in mainstream interest in having virtualization support incorporated into microprocessor design. While architectures such as the PowerPC have always provided some level of support for virtualization, recent chips from IBM, AMD, and Intel provide unprecedented support for system partitioning [Tsao04][VTwp][Pacifica].

These new hardware virtualization features support more efficient partitioning of system resources and create an opportunity to rethink the systems software stack. *Hypervisors*, the software agents which manage partitioning of the system, subsume a substantial portion of roles typically performed by an operating system. By embracing this fact, one can engineer the hypervisor to collaborate with operating systems in order to provide a more efficient virtualized environment. This approach is often called partial virtualization, or para-virtualization [Whitaker02]. Recent research projects such as *Xen* have shown minimal performance degradation from running in such para-virtualized environments for typical workloads even without extensive hardware support [Barham03]. It has been suggested that such technology may be applied in high perfor-

mance computing clusters to better manage resources, improve overall reliability, and provide fine-grained systems software customization [PROSE05].

An increasingly important factor in parallel cluster applications is interference from the systems software stack [Tsafir05]. The impact of this so-called “OS noise” creates problems synchronizing barriers across large clusters and creates efficiency problems along with low-utilization of system resources. Virtualization technology has been put forth as part of the solution to better managing these resources, but also can be a source of increased system noise.

The rest of this paper gives an overview of hypervisor technology and analyzes its impact on the amount of system software interference applications are likely to experience. Section 2 describes modern hypervisor technology in more detail, while section 3 describes our use of this technology to provide stand-alone HPC applications kernels. Section 4 describes our measurement methodology and section 5 reports our initial experimental results. Section 6 describes potential extensions to virtualization technology to reduce OS noise and increase cluster efficiency and we conclude in section 7.

2. Hypervisors

The hypervisor forms the foundation of the virtualized system software stack. It provides high-level memory, page table management as well as partition scheduling policies. Some hypervisors handle virtualizing I/O devices and resources while others delegate this responsi-

bility to special purpose I/O partitions. Interrupts are handled in a variety of ways. They can preempt the running partition and force a context-switch to the interrupt destination partition. Other hypervisors mitigate interrupts by preempting the running partition, acknowledging the interrupt, scheduling its delivery to the appropriate logical partition and then returning to the preempted partition. Finally, some hardware has interrupt routing allowing hardware assisted interrupt mitigation [CellArch1].

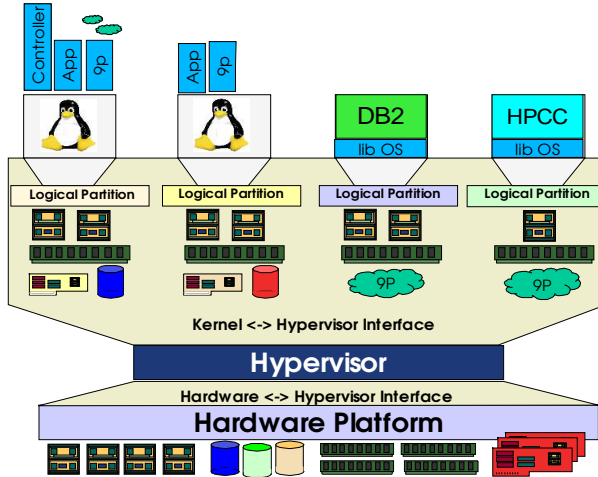


Illustration 1: Para-Virtualized Software Stack

We have based our evaluation on the architecture employed by the IBM research hypervisor, *rHype*. *rHype* is a small (~30k lines of code), low-latency, modular, multi-platform (supports x86 and PowerPC) para-virtualization engine. Logical partitions (LPARs) access *rHype* services through a "system call" like mechanism known as a *hcall*.

rHype employs an extremely simple round-robin slot scheduler and mitigated interrupts. Each partition is assigned to fixed length scheduling slots. In *rHype*'s case on the PowerPC, the default scheduling interval is 20ms (specified as 50,000 cycles of 250 MHz timer) as determined by a special processor-supplied hypervisor decremter counter (HDEC). In our experiments we assigned the PROSE partition 9 slots, with the I/O controller taking a single slot (approximately a 10% time slice of the CPU).

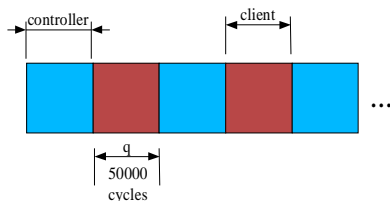


Illustration 2: *rHype* scheduler

Since the HDEC functions independent of the system decremter it can provide deterministic intervals immune to interference from underlying operating systems. Also, hypervisors are able to arbitrarily preempt underlying operating systems at any point in their operation, removing interference caused due to lengthy interrupt latencies.

The interference isolation hypervisors provide is limited by the fact that they are typically used only to multiplex hardware resources between multiple instances of traditional operating systems. These operating system instances are just as susceptible to noise when running in partitions as when they are running stand-alone.

3. HPC Library OS Overview

Our approach, the Partitioned Reliable Operating System Environment (or PROSE), extends the idea of hosting multiple operating systems on a virtualized machine to using partitions to host stand-alone applications. These applications can be services such as databases or they can be end-user applications such as the high performance computing challenge workloads [HPCC]. Since they essentially "own the virtual machine" there is an increased level of control and determinism during execution. They can also benefit from custom system components such as specialized schedulers and memory allocators. This approach has the same motivations and intent as *Exokernels* [Engler95], but leverages the more secure protection mechanisms of virtualization and enables interaction with legacy operating systems and drivers executing in other partitions.

Application partitions are executable from the command line of a traditional "controller" operating system and users interact with the application via familiar mechanisms such as standard I/O. Likewise, the application partition has full access to the controller's resources such as the file system via familiar library interfaces. To facilitate development and debug, partition memory and run-time control interfaces are accessible to the end-user on the "controller". A library of system service building blocks such as thread support, memory allocators, page table management, and other components are provided to allow developers to build more complicated custom application kernels.

I/O to and from these application partitions is resolved by communication through shared memory channels to the controlling partition. This same mechanism can be used to directly share devices such as disks or share system services such as networking stacks. In order to minimize complexity and code size, we use Plan 9's 9P [9man] as a single unified resource sharing protocol.

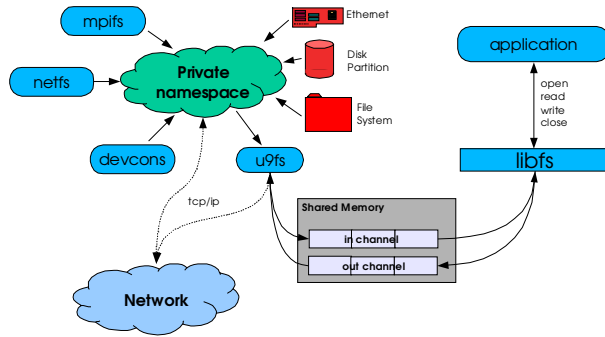


Illustration 3: Partition Resource Sharing

We envision several potential applications for such an infrastructure. We are most interested in exploring the potential improvements by applying the hybrid model to databases, high performance computing, real-time and multimedia applications.

Applications with immense memory footprints, such as databases or Java virtual machines, typically attempt to manage their own memory and paging behavior. In many cases, operating system facilities and policies interfere with their ability to efficiently manage these resources. During certain operations, such as sparse access and update of data, page table miss penalties can dominate overall performance. Large page support provides some relief, but is not readily available in stock Linux environments and does not scale to the memory hierarchies of the near future. Virtualization allows these applications more explicit control of their memory hierarchy. Custom application partitions can even operate with virtual memory disabled, removing the overhead of page table operations and miss exceptions.

High performance computing (HPC) applications could also benefit from more explicit control over hardware resources. Like databases, they are characterized as having large data sets with sparse access patterns. Additionally, HPC workloads often employ custom programming and threading models. Such customizations benefit from being able to "push the operating system out of the way" and interact directly with the hardware.

When HPC applications run on a cluster of machines, tight timing constraints and short communication latencies are vital to overall system productivity and performance. Hypervisors can provide a very thin, low-latency control layer which can provide more deterministic scheduling policies. It is our hope that applications running under such an infrastructure will be isolated from typical sources of OS interference. The next section evaluates how effective rHype is in achieving this goal.

4. Measurement Methodology

We have performed a preliminary evaluation using identically configured single-processor 1.6GHz IBM BladeCenter JS20 server blades with 4 GB of memory and gigabit Ethernet. One blade runs the application under a traditional Linux kernel, while the other blade runs the micro-benchmark in a stand-alone application kernel running alongside a Linux controller kernel under rHype. The Linux kernels we used were 2.6.12 kernels configured with a scheduler HZ rate of 1000.

Since we are only evaluating the degree of noise instead of looking to analyze any patterns in the measured noise, we chose the Fixed Work Quantum (FWQ) micro-benchmark. This benchmark has clear deficiencies [Sottile04] preventing deep analysis of system interference, and we intend to use more complex micro-benchmarks such as the Fixed Time Quantum (FTQ) benchmark in future experiments.

```

for(i=0; i<num_samples; i++) {
    start = mftb();
    for(w=0; w<work_len; w++);
    stop = mftb();
    delta[i] = stop-start;
}

```

Example 1: Fixed Work Quantum Benchmark

As can be seen in Example 1, our version of FWQ consists of measuring the time to execute an empty loop. We use the PowerPC's internal time base cycle counter versus OS ticks to eliminate potential interference. In order to determine the adequate number of samples and workload length we performed a series of sensitivity benchmark runs.

For sample sizes we took measurements on an idle Linux blade at rates of 10, 100, 1000, and 2000 samples. Samples around 1000 seemed sufficient to capture periodic behavior while keeping reasonable run-times.

OS interference tends to be fairly independent of the length of a particular workload. Workloads which are too small may not be as likely to be affected by scheduler interference and may exaggerate the overall effect of the interference. We settled on a workload size of 100,000,000 which had an average run time of 100 milliseconds on an otherwise idle Linux test-node. This is likely much longer than the typical time between synchronization barriers for HPC applications, but was necessary to capture scheduling interference from both the Linux operating system and the hypervisor partition scheduler.

A major source of interference outside of the scheduler and servicing of timer interrupts are interrupts from various peripheral devices. Since most cluster HPC workloads exist in rather congested and complicated

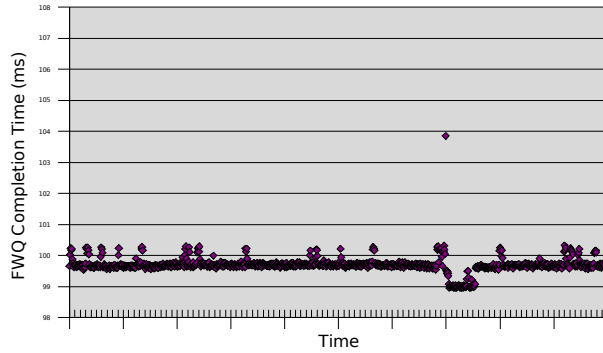


Illustration 4: Idle Linux

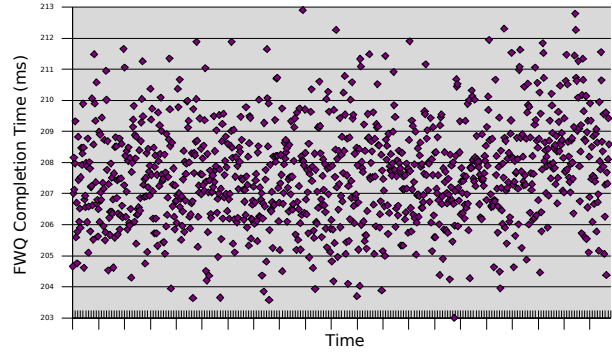


Illustration 5: Loaded Linux

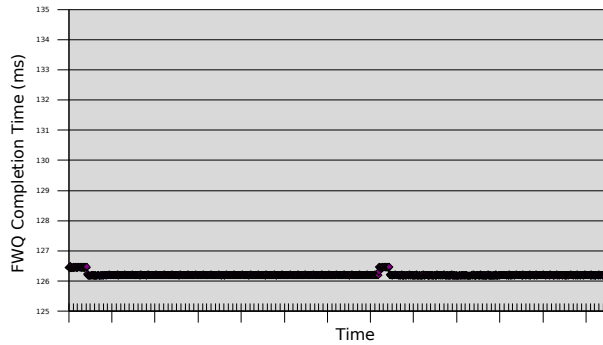


Illustration 6: Idle PROSE

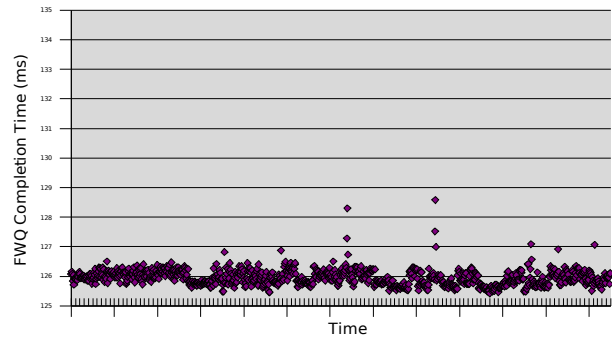


Illustration 7: Loaded PROSE

network environments, we took noise measurements while both of the aforementioned systems were experiencing heavy network traffic.

We initially tried to model this traffic using netperf [NetPerf] and were able to calibrate noise induced by the network to discrete values based on the load netperf was exerting. However, during analysis we realized that the majority of the variation in execution time was not introduced by the system software stack, but rather by the netperf server application.

To combat this, we switched to a method of injecting network interrupts which could be serviced entirely by the kernel without requiring user-space intervention. We used the ping(8) utility to provide a stream of traffic to the target system. We attempted to vary the intensity of the ping packet stream by varying the preload window which determines the number of outstanding ping requests. We then took noise measurements with preload windows of 100, 1000, and 10,000 packets in order to have multiple points for comparison. In the end, any network load seemed to negatively impact the standard deviation of the results so we will just be reporting the measurements taken while the network was idle and when the network was loaded by ping with a preload buffer of 10,000 which resulted in a system load of 45% on an idle target.

5. Results

Our initial experimental results are summarized in illustrations 4-7. All graphs are on the same scale, with the y axis representing the number of milliseconds it took to complete the fixed workload.

Illustration 4 shows measurements taken on an idle Linux system with only background network traffic and system daemons causing intermittent variation. By contrast, illustration 5 shows the same measurements taken on a system experiencing heavy network load. Not only do workloads take a factor of 2 longer to complete, but the resulting run times are heavily dispersed – demonstrating the effectiveness of our mechanism for increasing system noise by using ping to increase the network (and consequently interrupt) load.

Its worth noting that our “noise generation” methodology tends to generate a fairly constant amount of additional load on the target system. A more aggressive noise generator would constantly vary the load between the idle extreme and heavy network load over the course of a work unit. As work takes about twice as long under heavy load this would incur noise penalties of around 100%, radically increasing the standard deviation of result samples.

Illustration 6 exhibits the results of a PROSE application kernel running along side a Linux controller kernel on an idle system. As can be seen from the y-axis, workloads take approximately 25% longer to complete. This is in part due to the fact that the Linux Controller and the PROSE application kernel are sharing a processor without yielding to the other partition when idle. However, this only explains 10% of the increase, the rest must be chalked up to virtualization overhead. Despite taking longer in the idle case, the PROSE kernel run-times demonstrate less variance than a standard Linux kernel.

The small periodic interference which can be seen at the beginning and towards the middle of the samples are the result of scheduling-phase interference caused by workload not being synchronized with the fixed-slot scheduling algorithm. The difference in when a workload starts determines how many partition quanta it takes to execute – since the partitions are fixed, this may increase the noise by the length of the quanta of the hypervisor. Increasing the number of slots the application is scheduled should reduce this overhead at a cost of I/O latency and throughput. This trade-off must be carefully balanced for an application.

We did some runs varying the HDEC quanta, measuring idle noise at 2ms and 200ms to compare with our 20ms results. What we found was that the magnitude of this interference along with duration of the interference is directly related to the HDEC quanta. The smaller the quanta the smaller degree of interference. At 2ms, the phase-scheduling variance was imperceptible, the variance at 200ms can be seen in Illustration 8.

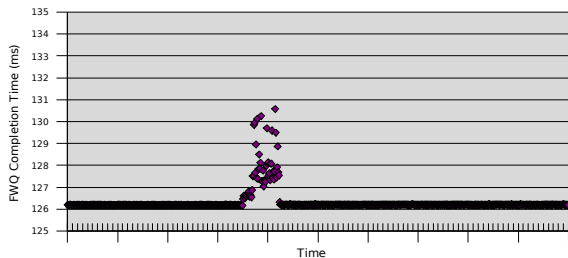


Illustration 8: Idle PROSE w/200ms HDEC

When the same heavy network load applied in illustration 5 is applied to the PROSE application kernel, the overall run-time remains the same with a slight increase in variance. The additional noise is due to the hypervisor having to mitigate delivery of the external interrupts. As can be seen in Illustration 7, the interference caused by acknowledging the interrupts is relatively minimal resulting in 1-2% variance under high load (as compared to 10-100% variance in a normal Linux kernel).

6. Future Work

This study was done on single-processor systems. It would be interesting to extend the study to SMP systems. On such systems, application kernels can be given dedicated CPUs which should hypothetically eliminate any noise from the hypervisor and controlling partition.

We have only evaluated a single hypervisor platform, a comparison of the interference introduced by various other virtualization solutions (such as Xen, VMWare, IBM's commercial hypervisor, etc.) on different platforms (including Intel and AMD's new architectures) would be a valuable tool in selecting virtualization technologies for HPC projects.

A major source of remaining interference was due to the overhead of mitigating interrupts. Measurements should be taken on systems with hardware assisted interrupt routing to see if this additional interference can be eliminated. Masking external interrupts during PROSE partition execution may be another way to eliminate this additional noise.

The fact that workloads take longer on idle PROSE kernels is an unfortunate consequence of the fixed slot scheduling. Partitions could yield when idle, but this would likely increase the amount of noise on idle systems. In such cases, the degree of maximum noise could be limited by manipulating the maximum number of slots assigned to I/O handling. Another alternative is dynamically adjusting the number of slots assigned to I/O versus computation. This would allow system architects to control the amount of processing power to devote to servicing I/O requests versus doing computation while limiting the effects of OS interference.

We focused the evaluation in this paper on the PROSE application kernel environment and did not do an evaluation of noise levels on the I/O host or Linux running as a client partition. Future, more complete studies will include measurements on client Linux partitions and not just PROSE application kernels.

Our micro-benchmark illustrated the use of virtualization to control operating system interference. However, the workload used was far too simplistic and uncharacteristic of a high-performance computing workload. Further studies need to be done using distributed HPC benchmarks with I/O and/or synchronization barriers in order to prove the real-world effectiveness of these techniques. In order to maintain determinism in these more complicated workloads, we will likely have to retrofit our naive scheduler with something more adaptive – applying lessons from real-time operating systems.

7. Conclusions

Virtualization is rapidly becoming a common piece of system infrastructure, particularly in high-end servers. Understanding the impact of various virtualization technologies and techniques on the run times of workloads is important, particularly for large parallel cluster applications which may require deterministic synchronization in order to run efficiently. It is apparent from our measurements that existing virtualization engines can increase the amount of system interference present, but the same technology can be used in order to limit the degree of interference in high-load environments. Exploring how to adapt hypervisor partition scheduling is one key to efficient use of cluster resources for distributed high performance applications.

8. Acknowledgment

This work would not be possible without the contributions of Jimi Xenidis, Michal Ostrowski, Orran Krieger, and the rest of the rHype team. This work was supported in part by the Defense Advanced Research Projects Agency under contract no. NBCH30390004.

9. References

[Barham03] Xen 2002, Paul R. Barham, Boris Dragovic, Keir A. Fraser, and et al. , ucam-cl-tr-553, January 2003, University of Cambridge, Computer Laboratory.

[CellArch1] Cell Broadband Engine Architecture, Version 1.0, Sony Computer Entertainment, IBM, Toshiba, 2005. (<http://cell.scei.co.jp>).

[CellularDisco] Cellular disco: resource management using virtual clusters on shared-memory multiprocessors, Kingshuk Govil, Dan Teodosiu, Huang Yongqiang, and Mendel Rosenblum, 2000, ACM Transactions on Computer Systems, vol 18:3 , 229-262.

[Disco] Disco: Running Commodity Operating Systems on Scalable Multiprocessors, Edouard Bugnion, Scott Devine, Kinshuk Govil, and Mendel Rosenblum, 1997, ACM Transactions on Computer Systems, vol 15:4 , 412-447.

[Engler95] Exokernel: An operating system architecture for application-level resource management, Dawson R. Engler, M. Frans Kaashoek, and James O Toole, Jr., 1995, Proceedings 15th Symposium on Operating Systems Principles, 251-267.

[HPCC] High Performance Computing Challenge, (<http://icl.cs.utk.edu/hpcc/>).

[NetPerf] NetPerf Home Page, (<http://www.netperf.org/>).

[9man] Plan 9 Programmer s Manual, Volume 1, AT & T Bell Laboratories, Murray Hill, NJ, 1995.

[Pacifica] AMD Virtualization Codenamed "Pacifica" Technology, Secure Virtual Machine Architecture Reference Manual, AMD, May 2005.

[PROSE05] Partitioned Reliable Operating System Environment, Eric Van Hensbergen, 2005. IBM Technical Report #RC23694.

[rHype] IBM Research Hypervisor, Jimi Xenidis, 2005. (<http://www.research.ibm.com/hypervisor>).

[Singh04] An Introduction To Virtualization, Amit Singh, <http://www.kernelthread.com/publications/virtualization>, 2004.

[Sottile04] Analysis of microbenchmarks for performance tuning clusters, Matthew Sottile, Ron Minnich, Proceedings of Cluster 2004, September 2004.

[Tsafrir2005] System noise, OS Clock Ticks, and Fine-Grained Parallel Applications, D. Tsafrir, Yoav Etsion, et. al, Proceedings of the 19th ACM International Conference on Supercomputing, June 2005.

[Tsao04] Server Consolidation Using POWER5 Virtualization White Paper, H. Tsao and B. Olszewski, http://www-1.ibm.com/servers/eserver/pseries/hardware/whitepapers/570_serverconsol.html , 2004.

[VTwp] Enhanced Virtualization on Intel Architecture-based Server, Intel Solutions White Paper, March 2005.

[Whitaker02] Denali: Lightweight virtual machines for distributed and networked application, A. Whitaker, M. Shaw, and S. Gribble, 2002, Proceedings of the USENIX Annual Technical Conference.