Benchmarking Multiprocessing Parameters in a Virtualized Multi-Core Environment

IIT Bombay Technical Report TR-CSE-2015-76

Chinmay Satish Kulkarni, Purushottam Kulkarni
Department of Computer Science and Engineering, IIT Bombay

Abstract

A virtualized multi-core environment is a multi-core machine running virtual machines with multiple virtual cpus. Previous research has demonstrated that the performance of multi-threaded, disk intensive, and network intensive applications in such an environment can be greatly improved by correctly tuning certain simple multiprocessing related system parameters like cpu affinity and interrupt affinity. However, there is no clarity on the implications of tuning these parameters together, and the implications of tuning these parameters in an environment where application behavior is dynamic. We present preliminary results of experiments designed towards filling up these gaps, and developing a unified framework to tune this set of parameters.

1 Introduction

The past decade has witnessed the emergence of cloud computing, a new paradigm for offering services that has dramatically changed and simplified the way consumers and businesses operate. Formally, cloud computing is based on the concept of offering computation as a service. This service can be in the form of software (SaaS), a product (PaaS), or an infrastructure (IaaS). A recent study [3] listed cloud computing as one among the twelve most disruptive technologies of the current generation.

The arrival of cloud computing has spurred systems research towards improving the utilization, efficiency, manageability, and security of modern day systems. Out of all proposed solutions, system virtualization has emerged as a simple and effective method of doing so. Similarly, research in computer architecture has shifted towards improving the power efficiency and parallelism of microprocessors with multi-core processors having emerged as a straightforward and scalable way of doing so [14]. The simplicity and popularity of these two technologies has encouraged cloud service providers towards running virtual machines on multi-core processors and exposing this infrastructure to users by allowing a virtual machine to be configured with multiple virtual cpus. For example, amazon ec2 allows a user to configure a virtual machine (instance) with upto 40 virtual cpus [1]. Such environments with multiple virtual cpus running on multiple cores are called *virtualized multi-core environments*.

1.1 Challenges in Virtualized Multi-Core Environments

1.1.1 Synchronization Latency

The arrival of multi-core processors has encouraged multi-threading, a concept that allows an application to have multiple threads of execution. Applications of this form typically use primitives such as spinlocks, mutexes, conditions, and barriers to synchronize access to shared variables and data structures. Research into the implications of executing such applications in virtualized multi-core environments has revealed large overheads associated with these synchronization primitives. Non blocking synchronization within the operating system's kernel is a major contributor to this overhead. A hypervisor being unaware of synchronization within a virtual machine can decide to preempt a lock holder i.e a virtual cpu running a thread holding a non blocking kernel synchronization primitive, or a lock waiter i.e a virtual cpu running a thread waiting to acquire a similar primitive. The former, called lock holder preemption [17] can cause lock waiters to waste cpu time busy waiting on a preempted synchronization primitive if the primitive happens to be a kernel spinlock. The latter, called lock waiter preemption [15] can cause lock waiters to waste cpu time busy waiting on a free synchronization primitive if the primitive happens to be a ticket spinlock i.e a kernel spinlock which enforces an ordering among threads waiting to acquire it. Recent research has shown that shorter virtual

cpu time slices can mitigate the overhead due to both, lock holder preemption and lock waiter preemption [5].

1.1.2 Interrupt Latency

Device interrupt delivery in multi-core machines is managed by the *Advanced Peripheral Interrupt Controller* or APIC [9]. The APIC's most important component is an interrupt redirection table which determines an interrupt's affinity i.e the subset of cores an interrupt can be delivered to. An interrupt is said to be bound to this subset of cores. Research has identified this affinity as a key barrier to improving io performance in virtualized multi-core environments. In such an environment, an interrupt can be delivered to a virtual machine only when one of the virtual cpus it is bound to is executing on the physical host. If the environment happens to be under cpu contention, this strict requirement for interrupt delivery could lead to a large latency between an interrupt's arrival at the physical host and it's delivery to a virtual machine. IO intensive applications like database servers which wait on device (disk) interrupts have been shown to experience a significant hit in throughput due to this latency. Recent research has demonstrated that balancing interrupt affinity between a virtual machine's virtual cpus [10], or running virtual cpus at a shorter time slice [5, 18] can lead to significant reductions in interrupt handling latency, and improvements in io throughput.

1.2 The Problem

There are three simple and easy to tune parameters available in a virtualized multi-core environment - i) virtual cpu time slice, ii) virtual cpu affinity, iii) interrupt affinity. These *multiprocessing* parameters are a result of the multiple virtual cpus and processors available in the environment, and have been previously used to mitigate the synchronization latency and interrupt latency problems. However, there is no clarity on the implications of tuning these parameters together, and the implications of tuning these parameters in an environment where application behavior is dynamic. In other words, the following question still remains unanswered

> *How should multiprocessing parameters in a virtualized multi-core environment be tuned to maximize application performance?*

We intend to answer the above question by benchmarking the set of multiprocessing parameters using a set of relevant workloads, and present preliminary results in this report.

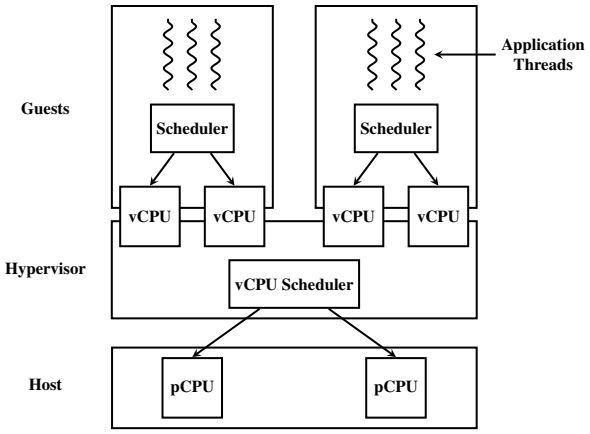

Figure 1: A typical virtualized multi-core environment

1.3 Organization

The rest of this report is organized as follows. In Section 2, we present background on a virtualized multi-core environment and the set of multiprocessing parameters. In Section 3, we present a summary of the previous work in the area. In Section 4, we present the design of our experiments, and preliminary results. In Section 5, we conclude with the future work.

2 Background

We first present an overview of a virtualized multi-core environment, and the set of multiprocessing parameters available in one. We provide a brief description of each parameter, and briefly discuss how each one of them can be tuned. We assume a linux based host and guest operating system.

2.1 Virtualized Multi-Core Environment

Figure 1 illustrates a typical virtualized multi-core environment. The physical host (host) is a multi-core machine configured with two cores (pCPUs). Two virtual machines (guests) configured with two virtual cpus (vCPUs) each have been consolidated on the host. Applications running within the guests are multi-threaded, and are scheduled by the guest's scheduler on vCPUs. vCPUs are scheduled on pCPUs by the hypervisor's vCPU scheduler.

2.2 Multiprocessing Parameters

Table 1 presents the set of multiprocessing parameters in a virtualized multi-core environment based on the level at which they can be tuned. Parameters can be tuned at either the host, the hypervisor, or the guest.

Level	Configuration(s)
Host	Interrupt affinity
Hypervisor	vCPU affinity vCPU time slice
Guest	Interrupt affinity

Table 1: Multiprocessing parameters in a virtualized multi-core environment

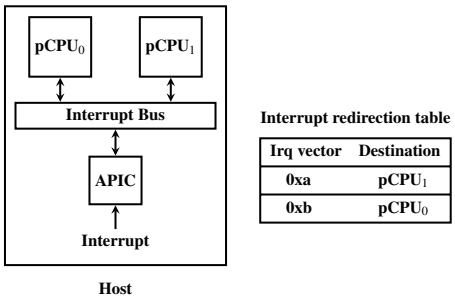

Figure 2: Interrupt affinity at the host

2.2.1 Parameters at the Host

Interrupt Affinity A device interrupt arriving at the host of a virtualized multi-core environment can potentially be forwarded to any of it's pCPUs. The Advanced Peripheral Interrupt Controller or APIC takes care of this forwarding. The APIC contains an interrupt redirection table which determines the pCPU an interrupt is forwarded to. This redirection table can be tuned by the host's operating system to set an interrupt's affinity i.e the subset of pCPUs an interrupt can be forwarded to. This affinity can range from a single pCPU to all the pCPUs on the host. Figure 2 provides a high level view of this parameter. An interrupt arriving at the host is first delivered to the APIC. Based on it's affinity, it is either forwarded to $pCPU_0$ (irq 0xb) or to $pCPU_1$ (irq 0xa). The linux operating system allows an interrupt's affinity to be set through the `/proc` filesystem.

2.2.2 Parameters at the Hypervisor

vCPU Affinity In a virtualized multi-core environment, a vCPU is by default allowed to execute on any pCPU by the hypervisor. This allows the vCPU scheduler to balance the environment's load across the pCPUs of the host. However, the scheduler can be tuned to restrict the execution of a vCPU to a subset of the host's pCPUs. This subset is called the vCPU's affinity. For example, in figure 3, the affinity of $vCPU_1$ and $vCPU_3$ allows them to execute on any of the host's pCPUs. On the other hand, the affinity of $vCPU_0$ and $vCPU_2$ restricts

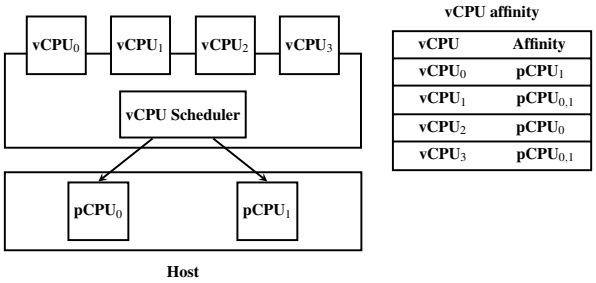

Figure 3: vCPU affinity at the hypervisor

their execution to $pCPU_1$ and $pCPU_0$ respectively. A vCPU's affinity can be set independent of the guest it belongs to.

vCPU Time Slice The vCPU scheduler typically allows a vCPU to run for a continuous interval of time before preempting it. The duration of this interval of time is called the vCPU's time slice. The xen hypervisor [6] provides vCPUs with a default time slice of 30ms while the kvm hypervisor [12] provides vCPUs with a time slice between 3ms and 24ms depending on the environment's load. Both hypervisor's allow these values to be modified. In kvm, this can be done through the `/proc` filesystem on the host operating system.

2.2.3 Parameters at the Guest

Interrupt Affinity A device interrupt arriving at the physical host of a virtualized multi-core environment is first forwarded to a pCPU where it is handled by the hypervisor. If destined to a guest, the hypervisor virtualizes the interrupt and forwards it one of the guest's vCPUs where it is handled by a kernel thread. From a high level perspective, a virtual APIC provided by the hypervisor to each guest performs this forwarding. The redirection table in this virtual APIC can be tuned by the guest's operating system to set the affinity of a virtual interrupt i.e the subset of vCPUs it can be forwarded to. Figure 4 illustrates this parameter. Once virtualized by the hypervisor, an interrupt is forwarded by the virtual APIC to one of the guest's four vCPUs based on it's affinity. In this example, interrupts 0xa and 0xc are forwarded to $vCPU_1$, and interrupts 0xb and 0xd are forwarded to $vCPU_3$. A virtual interrupt's affinity can be set through the `/proc` filesystem in a linux based guest.

3 Related Work

We present in this section a summary of the previous work in the area, and the gaps that we intend to fill through our research.

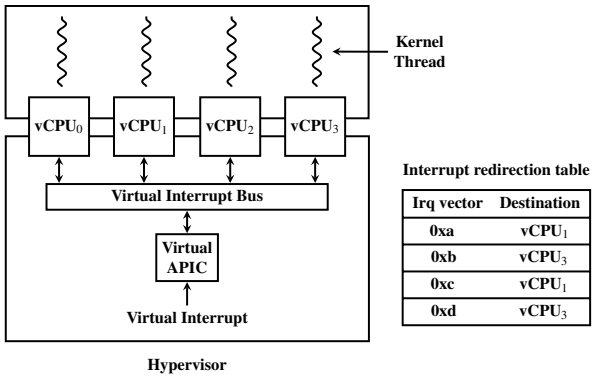

Figure 4: Interrupt affinity at the guest

3.1 vCPU Affinity

The authors of [13] benchmark the impact of vCPU affinity on the performance of the Parsec benchmark suite [7] in a virtualized multi-core environment, and report that it has no impact on application performance. However, our findings contradict theirs. This is because the authors fail to include an environment under cpu contention in their evaluation.

The authors of [16] analyze and evaluate the impact of different vCPU pinning strategies based on the cpu load in the environment using the Dacapo benchmark suite [8]. Based on their experiments, the authors come up with an adaptive pinning strategy that isolates guests on different processor chips under high cpu load, and spreads guest vCPUs across all cores under low or moderate cpu load.

3.2 vCPU Time Slice

vTurbo [18] employs shorter vCPU time slices to reduce interrupt handling latency in a virtualized multi-core environment. A subset of the host's pCPUs are run at a shorter time slice and are called turbo cores. In addition, every guest is configured with an extra vCPU called the turbo vCPU which performs all interrupt handling. vTurbo schedules all turbo vCPUs on turbo pCPUs. Doing so increases the frequency with which turbo vCPUs execute on the host, and decreases the latency between interrupt arrival at the host and interrupt handling in a guest.

The authors of [5] employ shorter vCPU time slices to overcome the virtual time discontinuity problem - a generalization of the synchronization and interrupt latency problem in virtualized multi-core environments. This generalization is based on the observation that a vCPU's execution in such an environment is discontinuous i.e a vCPU executes for an interval of time determined by the hypervisor, is scheduled out, and is subsequently rescheduled after some time interval depending on the hypervisor's scheduling policy. This discontinuity results in the synchronization and interrupt latency problems. A shorter vCPU time slice helps reduce this discontinuity by increasing vCPU execution frequency. In addition, the authors employ caching techniques like context preservation and context prefetching to overcome any context switch overheads.

3.3 Guest Interrupt Affinity

vBalance [10] balances interrupt affinity at the guest to ensure that virtual interrupt's are always bound to executing vCPUs. Doing so helps mitigate the interrupt latency problem in a virtualized multi-core environment by ensuring immediate delivery of an interrupt arriving at the host to the guest it is destined for as long as one of the guest's vCPUs is executing on the host. vBalance works by exposing a guest to the status of it's vCPUs (running/waiting). A load balancing component within the guest adjusts interrupt affinities accordingly.

3.4 Gaps in Prior Research

Table 2 summarizes the parameters analyzed by prior research. We conducted a comprehensive survey of this research and were able to identify the following gaps

- All parameters have been analyzed in isolation. There has been no research into the implications of tuning these parameters together.
- Parameters have been analyzed with a very limited set of workloads. For example, the impact of shorter time slices on a practical memory intensive workload is not clear.
- The implications of tuning these parameters in an environment where application behavior is dynamic is not clear.

4 Experimental Evaluation

We present in this section the design of our experiments, and preliminary results obtained with the Parsec benchmark.

4.1 The Question

The main focus of our work is to answer the following question

> *How should multiprocessing parameters in a virtualized multi-core environment be tuned to maximize application performance?*

Evaluation	vCPU Time Slice	vCPU Affinity	Guest Interrupt Affinity
[13]		✔	
[18]	✔		
[5]	✔		
[10]			✔
[16]		✔	

Table 2: A summary of parameters analyzed by prior research

Host CPU	Intel i7 3370 (3.4 GHz)
Host Memory	8GB
Host Storage	500GB 7200 RPM HDD
Host Kernel	64-bit Linux 3.16
Hypervisor	Kvm with Qemu 2.0.0
Guest Memory	4GB
Guest Storage	30GB Qemu HDD
Guest Kernel	64-bit Linux 3.16

Table 3: The test bed used for our experiments

In order to do so, we have designed experiments to understand

- How tuning a given multiprocessing parameter can impact application performance in a virtualized multi-core environment.
- How the entire set of multiprocessing parameters should be tuned to maximize a given application's performance.

4.2 Experimental Setup

4.2.1 Test Bed

Table 3 presents our test bed. All our experiments were run on a quad core host with hyper-threading and dynamic frequency scaling enabled. In addition, all guests were configured with 8 vCPUs.

4.2.2 Application Performance

We have used application level metrics like completion time as an indicator of application performance. Moreover, we have designed our experiments to understand the impact of the set of multiprocessing parameters on application performance under different degrees of pCPU contention in the environment. Our rationale for doing so is to account for the fact that most cloud providers perform some form of resource oversubscription to cut down on operational costs.

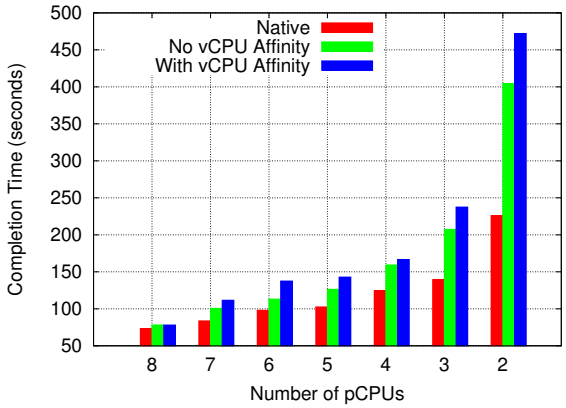

Figure 5: The impact of vCPU affinity on fluidanimate

4.3 Experiments

4.3.1 vCPU Affinity

We present the impact of vCPU affinity on workloads from the Parsec benchmark in this section. We compiled the workloads with the `gcc-pthreads` flag of Parsec, and configured them with 8 threads. We ran the workloads in an 8 vCPU guest, and set the affinity of every vCPU to a single pCPU. We took care to evenly distribute vCPUs on pCPUs while doing so.

Figure 5 and 6 present the outcome of our experiment on fluidanimate and streamcluster. All measurements are averages over five runs of the respective workload. We have also included the outcome of running the two workloads directly on the host of our test bed (Native). Results obtained on running our experiment with other workloads from Parsec were similar to those presented. The following observations can be made.

- vCPU affinity deteriorates the performance of fluidanimate under pCPU contention (< 8 pCPUs).
- vCPU affinity improves the performance of streamcluster under pCPU contention except in the case of 6 and 2 pCPUs.

To explain these results, we measured the number of rescheduling inter-processor interrupts (IPIs) and the number of cache misses during fluidanimate's and streamcluster's execution on our test bed. Figure 7 and 8[1] present our results. The following observations and inferences can be made.

[1] vCPU affinity has no impact on the number of cache misses during the execution of fluidanimate on our test bed.

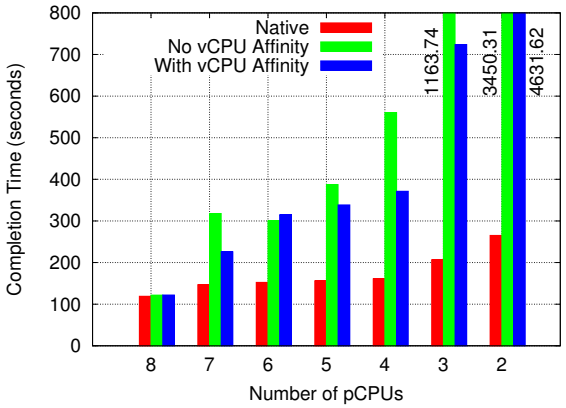

Figure 6: Impact of vCPU affinity on streamcluster

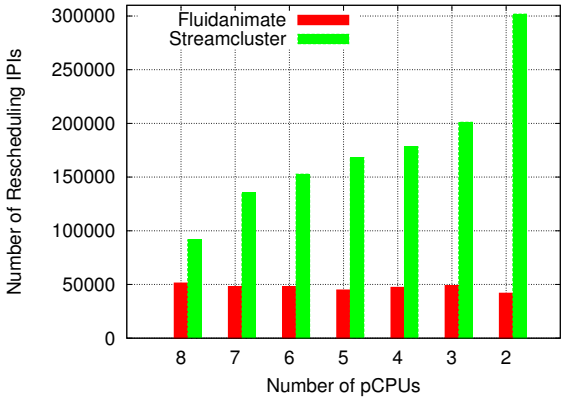

Figure 7: Number of rescheduling ipis during fluidanimate's and streamcluster's execution on our test bed

- **Fluidanimate:** A large number of rescheduling inter-processor interrupts (IPIs) occur during the execution of fluidanimate on our test bed ($\approx$ 50,000). These IPIs are caused due to a substantial amount of synchronization between the workload's threads, and result in vCPUs getting migrated to, and scheduled on idle pCPUs. Setting a vCPU's affinity to a single pCPU prevents such migration from taking place, and hence deteriorates the workload's performance.

- **Streamcluster:** An extremely large number of rescheduling IPIs occur during the execution of streamcluster on our test bed. However, on setting a vCPU's affinity, a reduction in the number of cache misses offsets any deterioration in performance.

Conclusion We conclude from our experiments that the impact of vCPU affinity on application performance in a virtualized multi-core environment depends on the number of rescheduling IPIs and cache locality during an application's execution. An application which results in a large number of rescheduling IPIs but poor cache locality will experience deteriorated performance, while an application which results in good cache locality will experience improved performance irrespective of the number of rescheduling IPIs.

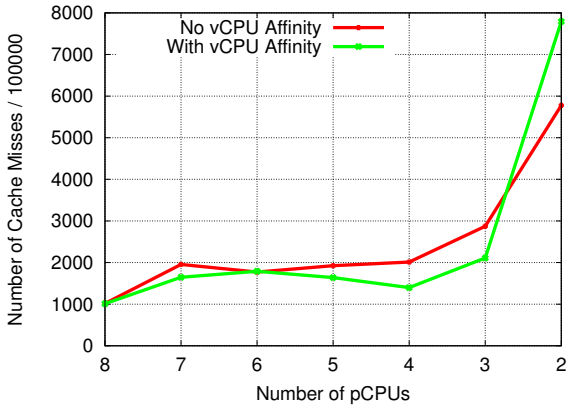

Figure 8: Number of cache misses during streamcluster's execution on our test bed

5 Future Work

5.1 vCPU Time Slice

We have so far restricted ourselves to vCPU affinity, a familiar parameter. We intend to follow it up with an analysis of the impact of vCPU time slice on application performance. Our preliminary experiments suggest that a shorter vCPU time slice can improve application performance, and complements observations made in [5]. However, we have also observed an accompanying increase in the number of cpu cycles on the host, and are currently running experiments towards identifying it's consequences.

5.2 Workloads

Our experiments have so far been restricted to Parsec - a cpu and synchronization intensive benchmark for multi-core processors. However, applications running in a typical cloud are very diverse, and usually include web servers and in-memory key value stores. To account for such diversity, we intend to expand our evaluation to the following applications/workloads.

- **Memcached [11]**: A popular multi-threaded in-memory key value store.

- **Dell DVD Store [2]**: A popular multi-threaded disk and network intensive benchmark.

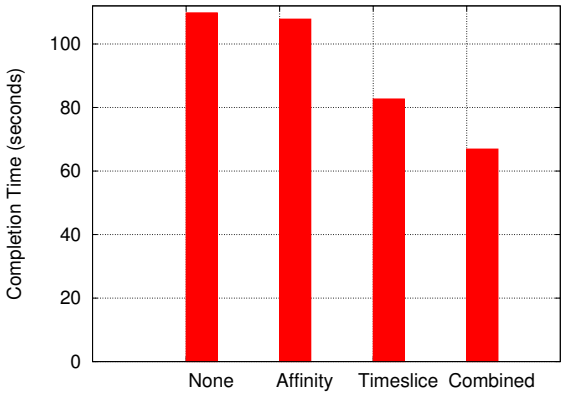

Figure 9: Impact of combining a 100μs vCPU time slice with vCPU affinity on vips

5.3 Combining Parameters

An experiment that we ran with the vips workload from the Parsec benchmark suite suggests that tuning two parameters together can lead to further improvements in application performance. We compiled the workload with the `gcc-pthreads` flag of Parsec, and configured it with 8 threads. We ran the workload in an 8 vCPU guest and enabled 4 pCPUs on our test bed. We repeated this four times. The first run had every parameter at it's default setting, the second had vCPU affinity enabled, the third run had the vCPU time slice tuned to a shorter value of 100μs, and the fourth had both, vCPU affinity enabled and the vCPU time slice tuned to 100μs.

Figure 9 presents the results of our experiment. It is evident that combining a shorter vCPU time slice with vCPU affinity improves the average completion time of vips. We are currently in the process of identifying the root cause of this behavior.

5.4 An Adaptive Framework

Our experiments suggest that setting a vCPU's affinity to a single pCPU improves the performance of one set of virtualized applications (streamcluster), and degrades the performance of another (fluidanimate). Therefore, an adaptive framework that sets vCPU affinity during the execution of an application that belongs to the former, and disables vCPU affinity during the execution of an application that belongs to the latter, will lead to improved application performance in a virtualized multi-core environment. We see three possible ways of implementing such a framework.

- **Shared Memory**: The guest can inform the hypervisor about the nature of the application executing within it through a shared memory object [4]. The hypervisor can then tune relevant parameters at it's level based on this information.

- **Hypercalls:** The guest can use hypercalls to directly tune parameters at the hypervisor depending on the nature of the application executing within it.

- **Host Indicators**: The hypervisor can keep track of metrics such as the cpu utilization and number of rescheduling inter-processor interrupts on the host, and use them as indicators of the type of application executing within a guest. Relevant parameters can then be tuned based on these indicators.

We expect the results of our experiments to determine the feasibility of these three approaches.

References

[1] Amazon ec2 instances. `http://aws.amazon.com/ec2/instance-types/`, November 2015.

[2] Dell dvd store. `http://linux.dell.com/dvdstore/`, November 2015.

[3] Disruptive technologies: Advances that will transform life, business, and the global economy. `http://www.mckinsey.com/insights/business_technology/disruptive_technologies`, November 2015.

[4] Inter-vm shared memory. `http://nairobi-embedded.org/linux_pci_device_driver.html`, November 2015.

[5] AHN, J., PARK, C. H., AND HUH, J. Micro-sliced virtual processors to hide the effect of discontinuous cpu availability for consolidated systems. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture* (Washington, DC, USA, 2014), MICRO-47, IEEE Computer Society, pp. 394–405.

[6] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2003), SOSP '03, ACM, pp. 164–177.

[7] BIENIA, C. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton, NJ, USA, 2011. AAI3445564.

[8] BLACKBURN, S. M., GARNER, R., HOFFMANN, C., KHANG, A. M., MCKINLEY, K. S., BENTZUR, R., DIWAN, A., FEINBERG, D., FRAMPTON, D., GUYER, S. Z., HIRZEL, M., HOSKING, A., JUMP, M., LEE, H., MOSS, J. E. B., PHANSALKAR, A., STEFANOVIĆ, D., VANDRUNEN, T., VON DINCKLAGE, D., AND WIEDERMANN, B. The dacapo benchmarks: Java benchmarking development and analysis. In *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications* (New York, NY, USA, 2006), OOPSLA '06, ACM, pp. 169–190.

[9] BOVET, D., AND CESATI, M. *Understanding The Linux Kernel*. Oreilly & Associates Inc, 2005.

[10] CHENG, L., AND WANG, C.-L. vbalance: Using interrupt load balance to improve i/o performance for smp virtual machines. In *Proceedings of the Third ACM Symposium on Cloud Computing* (New York, NY, USA, 2012), SoCC '12, ACM, pp. 2:1–2:14.

[11] FERDMAN, M., ADILEH, A., KOCBERBER, O., VOLOS, S., ALISAFAEE, M., JEVDJIC, D., KAYNAK, C., POPESCU, A. D., AILAMAKI, A., AND FALSAFI, B. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2012), ASPLOS XVII, ACM, pp. 37–48.

[12] KIVITY, A., KAMAY, Y., LAOR, D., LUBLIN, U., AND LIGUORI, A. kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium* (Ottawa, Ontario, Canada, 2007), vol. 1, pp. 225–230.

[13] LI, J., CHEN, X., SHEN, L., GAN, X., AND ZHENG, Z. Evaluating multithreaded workloads in cmp virtualization environment. In *Intelligent System Design and Engineering Application (ISDEA), 2012 Second International Conference on* (2012), pp. 254–258.

[14] OLUKOTUN, K., NAYFEH, B. A., HAMMOND, L., WILSON, K., AND CHANG, K. The case for a single-chip multiprocessor. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 1996), ASPLOS VII, ACM, pp. 2–11.

[15] OUYANG, J., AND LANGE, J. R. Preemptable ticket spinlocks: Improving consolidated performance in the cloud. In *Proceedings of the 9th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (New York, NY, USA, 2013), VEE '13, ACM, pp. 191–200.

[16] PODZIMEK, A., BULEJ, L., CHEN, L., BINDER, W., AND TUMA, P. Analyzing the impact of cpu pinning and partial cpu loads on performance and energy efficiency. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on* (May 2015), pp. 1–10.

[17] UHLIG, V., LEVASSEUR, J., SKOGLUND, E., AND DANNOWSKI, U. Towards scalable multiprocessor virtual machines. In *Proceedings of the 3rd Conference on Virtual Machine Research And Technology Symposium - Volume 3* (Berkeley, CA, USA, 2004), VM'04, USENIX Association, pp. 4–4.

[18] XU, C., GAMAGE, S., LU, H., KOMPELLA, R., AND XU, D. vturbo: Accelerating virtual machine i/o processing using designated turbo-sliced core. In *Proceedings of the 2013 USENIX Conference on Annual Technical Conference* (Berkeley, CA, USA, 2013), USENIX ATC'13, USENIX Association, pp. 243–254.