QUANTIFYING THE EFFECTS OF SHARED RESOURCE CONTENTION ON PERFORMANCE IN VIRTUALIAZED SYSTEMS

By

PRANAV PATHAK

Bachelor of Science in Electronics and

Telecommunication

Pune University

Pune, Maharashtra India

2007

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2011

QUANTIFYING THE EFFECTS OF SHARED

RESOURCE CONTENTION ON PERFORMANCE IN

VIRTUALIAZED SYSTEMS

Thesis Approved:

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Computing is undergoing a seismic shift from the client/server model to the cloud; this shift comes along with an evolving shift to multicore and further manycore CPUs on chip in hardware and the advent of virtualization technology in software as well as hardware.

Arguably, virtualization has been around in some form since the days of mainframe computing. The main motivation for virtualization in the early 70's was to increase the level of sharing and utilization of expensive computing resources. In the next few years, reduction in hardware costs meant that organizational computing needs could be fulfilled by a collection of minicomputers. Improved networking technology and ever increasing computing power gave rise to new computing models like client server and peer to peer systems. In a nonvirtualized datacenter, each application typically runs on its own physical server. In this model utilization of resources traditionally has been very low. Virtualization enables multiple applications to run on a single server [1]. The resurgence of virtualization is due to the move to multicore and the resulting underutilization of CPU cycles. In Large datacenters the new computing models presented new challenges in power consumption, reliability, security, complexity and cost. The virtualization technologies today seek to find a solution to these challenges in an elegant way [2, 3, 4, and 5].

In current virtualization solutions, guest Virtual Machines (VMs) and workloads are presented with an abstracted view of underlying physical resources [6]. The hypervisor or virtual machine monitor presents this virtual operating platform to the guest virtual machines and manages the execution of the guest operating systems. Typically, each VM runs its own operating system (OS) which in turn manages the resources presented to the VM to run its own workloads. The operating system of each VM is unaware of the other VMs sharing the physical resources. Varying workload configurations, across different VMs impact overall system performance and performance as seen by individual VMs; this is due to sharing of physical resources between VMs.

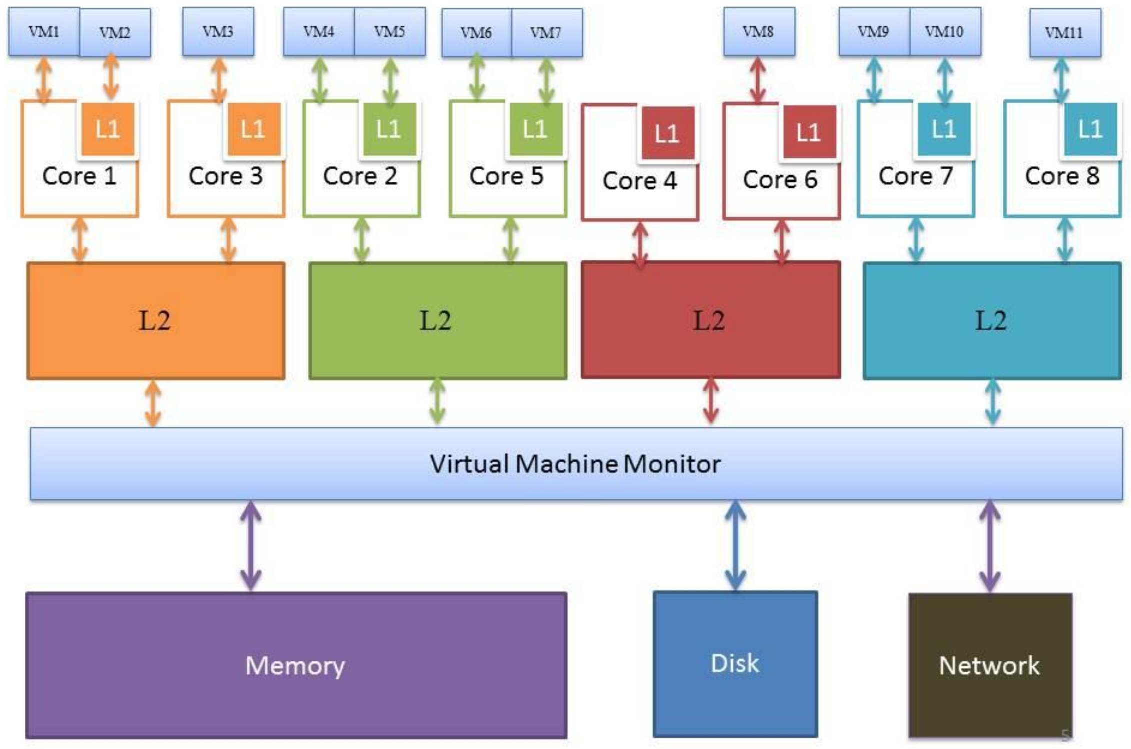

Figure 1: Example of virtualized system and resource sharing in the system. As seen here all virtual machines share Main memory, hard disk drive and network connection via the virtual machine monitor. As an example VM 1, 2 and 3 share L2 cache. VM1 and 2 share L1 cache and CPU core.

As hardware shifts towards many cores, software needs to evolve to provide flexible and efficient use of resources. Integrated hardware-software solutions for virtualization provide fault isolation as well as environment isolation to the applications. But current virtualization solutions do not provide explicit performance isolation between virtual machines [2]. A body of ongoing research exists, that profiles and models performance of varied workloads in virtualized environments [7, 8 and 9].

Koh et al. [7] employed an instrumented Xen hypervisor to collect performance characteristics and runtime characteristics of VMs that share a physical host. Wood et al. [8] measured the overhead of virtualization for Xen hypervisor running paravirtualized VMs. Menascé et al. [9] applied analytic queuing models to virtualized environments. Detailed profiling data and performance models go a long way towards handling resource sharing and scheduling in virtualized environments.

Management of physical resources especially of multicore processors and their memory hierarchies impacts the performance of virtualized systems [10, 11 and 12]. The decision of scheduling a virtual machine consists of two parts; first part is the decision, which physical resources a virtual machine should be allowed to use and second part is the decision of how long these resources can be allocated to a virtual machine. Scheduling of workloads is an attractive technique to reduce resource contention and achieve desired performance due to its low overhead. Scheduling does not need extra hardware and it is easy to integrate into the system. Researchers have explored the usefulness of scheduling techniques in virtualized systems. It has been demonstrated by Zhuravlev et al. [13] that a purely scheduling based approach based on cache miss rates reduces contention for shared caches effectively. They show that their algorithm provides aggregate performance improvement. Ghosh et al. [14] proposed architectural support to aid symbiotic scheduling and reduce cache contention.

1.1 Motivation

In a virtualized environment, guest VMs or applications cannot know about the tasks being performed by applications in the other VMs. Individual VMs have an abstracted view of the physical resources and hence are not in a position to capture the contention for physical resources. Thus many guest operating systems try to manage the shared resources without knowing about resource utilization, requirements, and decisions taken by other VMs to manage the resources. Koh et al. [7] have explained this phenomenon in their work in more detail. On the other hand the hypervisor can observe physical resources and their utilization, but it is unaware of resource utilization and requirements of each VM, and decisions taken by guest operating systems to manage the resources. Fedorova et al. [12] showed that contention for shared resources hampers the performance of multicore systems. *Thus if resource contention in virtualized systems can be captured, this information can be used to improve the overall system performance*.

The hardware counters are a set of special purpose registers available in most modern processors. Typically, these hardware counters can be programmed to count hardware related events. These counters can be used by programmers and operating systems alike to analyze and fine tune the performance of workloads. These counters can be programmed to capture the usage of shared resources at runtime in multicore architectures.

Large energy consumption in datacenters is an important concern these days. Bellosa et al. [16] have shown that hardware counters can be used to estimate the energy consumption in computing systems and this information can also be used to schedule the workloads for reduced energy consumption.

If the hypervisor can get an estimate of shared resource usage per virtual machine, then the hypervisor can make scheduling decisions at the physical host level. The hypervisor can target minimum shared resource contention for best system wide performance or can aim for minimum energy usage, while ensuring quality of service in either case.

1.2 Research overview

This research uses the hardware counters to capture resource contention information which will be conveyed to the hypervisor to aid scheduling of virtual machines**.** The goal of the scheduler can be minimizing interference for best system-wide performance and maximum throughput and minimizing the energy usage, while ensuring the quality of service in either case. There are two parts of the problem, first part is gathering the information that is useful for making the scheduling decisions and the second part is taking the actual scheduling decisions, which involves maintaining service quality, reducing resource contention, improving system performance and reducing energy consumption all at the same time. This is a complex multivariable optimization problem that should be solved within constrained time and computing resources.

In this research, we address the first part of the problem that is we find out what information can be gathered about executing of VMs that can aid the scheduling decisions. For this purpose the hardware counters were programmed to capture the hardware events for performance metrics that are good indicators of contention for shared resources in workloads. These performance metrics were then evaluated to model the resource contention and overall system performance in virtualized systems.

This work explores different configurations of virtual machine scheduling in virtualized systems and investigates which hardware events are good indicators of the contention for shared resources as well as the system performance.

CHAPTER II

RELATED WORK

In this chapter ongoing work in multicore research related to this work is discussed. This includes published work that deals with profiling and modeling performance in virtualized systems, publications related to multicore resource management, also papers related to runtime feedback and scheduling to optimize performance and power are discussed.

2.1 Profiling and modeling performance in virtualized systems

It is important to correctly model the application performance behavior, especially in virtualized systems, because here multiple physical resources are shared by virtual machines (VMs) and essentially by workloads. This resource contention is not obvious to VMs and it should be accounted for while taking scheduling decisions. Wood et al. [8] measured the overhead of virtualization on specific virtualization platforms. They have selected and used a specific set of microbenchmarks for this measurement. A regression based model was developed to map native system usage profile to virtualized system usage profile for the virtualization platform under consideration. This model is used to predict the resource requirements of any application to be executed on that virtualization platform. They demonstrated the use of this model at the time of launching new virtual machines on Xen platform. This approach is useful when launching new virtual machines on a platform but performance of applications for the native physical host should be known beforehand.

Thus it has been demonstrated that modeling the per VM resource requirements and using this information to take scheduling decisions can help the performance of virtualized system.

Koh et al. [7] employed an instrumented Xen hypervisor to collect performance characteristics and runtime characteristics of VMs that share a physical host. They also developed mathematical models to predict the performance of a new application from workload characteristics. Koh et al showed that per VM data gathered from VMM is useful in building performance models. In this work we gather per VM data that share physical host and explore the possibility of taking scheduling decisions to enhance the performance. Menascé et al. [9] applied analytic queuing models to virtualized environments. This analytic queuing model classifies resources into three types of resources, load independent, load dependent and delay resources. A case study on server utilization was used to illustrate the analytic model. Work of Menascé et al help us select the hardware resources that influence the performance for our study.

Bellosa et al. [16] have used hardware events for energy accounting and dynamic thermal management. A statistical approach is taken to, come up with a linear relationship between hardware events and energy consumed; furthermore this linear relationship is used for dynamic thermal management of resources. Although this work was not performed on virtualized systems, it is worth mentioning here because if this approach can be effectively extended to virtualized systems then it can address the important issue of large power consumption in the datacenters. Ge et al. [17] demonstrated the modeling of power consumption in multicore processors. The processor cores and the memory subsystems were modeled for their dynamic and idle power consumption. The processor frequency scaling also has been considered. First they developed performance models and power consumption models for different subsystems. These models were used to design energy consumption models.

2.2 Multicore resource management

One benefit of virtualization is its ability to multiplex several operating systems on hardware based on dynamically changing system characteristics. However, such multiplexing must be performed while observing per VM quality of service guarantees. It is important to manage the resources and meet the performance goals.

Matthews et al. [10] showcased a performance isolation benchmark that quantifies the ability of a virtualization system to limit the impact of a misbehaving virtual machine on other VMs that share the physical host. The work done by the kernel and the device driver for individual VMs is not accounted for, even when per VM resource constraints are in place. A comparison of different virtualization platforms is presented on the basis of different subsystems; they are memory, CPU, disk and network resources.

The contention of the shared resources significantly impedes the efficient operation of the multicore systems. Gupta et al [11] implemented a set of primitives in Xen to address this issue. They have developed three components, First Xenmon that measures per-VM resource consumption, including work done for a VM in Xen’s driver domain, second they modified SEDF-DC scheduler accounts for aggregate VM resource consumption, third shareguard that limits the total amount of resources consumed in Xen’s driver doamins.

Fedorova et al. [12] presented models to understand memory resource contention. The memory resources mentioned here include last level caches, memory controller and interconnects and, prefetching hardware. These new models were evaluated for their effectiveness in constructing contention-free thread schedules. They have developed a prototype of contention-aware scheduler for multicore systems called Distributed Intensity Online scheduler. Knauerhase et al [18] have used OS observations to improve performance it multicore systems. Similarly this work investigates observations made by VMM.

2.3 Run time feedback and scheduling

The scheduling of applications is an attractive tool for mitigation of contention for the shared resources because it does not require extra hardware and it is relatively easy to integrate into the system.

Zhuravlev et al. [13] investigated how and to what extent thread scheduling can mitigate contention for shared resources. While scheduling to reduce resource contention important problem is to find a classification scheme which would determine how competing threads affect each other when competing for shared resources. They studied different techniques and criteria for such classification and demonstrated their use with real systems and real schedulers.

Ghosh et al. [14] proposed bloom filter signatures, a low complexity architectural support to allow the hypervisor to infer cache footprint characteristics and interference of applications, and then perform scheduling based on symbiosis. They also proposed and studied three resource allocation algorithms to determine the optimal process-to-core mapping to minimize interference in L2 cache. In this work we seek to glean information that will lead to scheduling decisions such that overall system performance can be improved.

Verma et al. [15] studied server consolidation for reducing power consumption and identified application performance isolation and virtualization overhead as the key bottlenecks for server consolidation. They have showed that set size is a key parameter for scheduling applications on virtualized servers. They presented a framework and a methodology for power-aware application placement for HPC applications. Kim et al. [19] like Verma et al. [10] provided poweraware scheduling algorithm for bag-of-tasks applications on a DVS-enabled cluster system. Singh et al [20] use performance counters for thread scheduling. This work seeks to employ these concepts to virtualized systems.

CHAPTER III

METHODOLOGY

In this work, the hardware performance counters are used to record statistics related to cache hierarchy and memory, branch prediction statistics and other hardware events related to a physical core. The Hardware events related to a physical core include cycles for which reorder buffer or reservation stations are full, cycles during which instruction queue is full. The hardware events for our experiments were selected based on studies [8, 12] that have shown that these metrics are useful indicators of contention for resources and performance of the systems in computers, especially multicore systems. Also we selected events which we thought can intuitively indicate resource contention for shared resources, like cache misses when last level cache is shared.

The performance for a program or group of programs in this study means total time of execution for that program or group of programs. In other words, the time of execution is used as a measure of performance. In this study to check if a metric is a good indicator of contention for a shared resource and overall performance of the system we count hardware events related to a shared resource using performance counters, then we correlate between the event count and the execution time. The hardware event related to the shared resource that exhibits highest correlation with performance can be used to indicate contention for shared resource and its effect on overall system performance.

3.1 Experimentation Platform

The experiments were run on a 2 x Quad-core Intel Xeon L5410 system [21]. Each pair of cores in the processor shares a 6MB L2 cache. We use eight cores, of which each pair share an L2 cache for our experiments. The topology of the system is shown in figure 2.

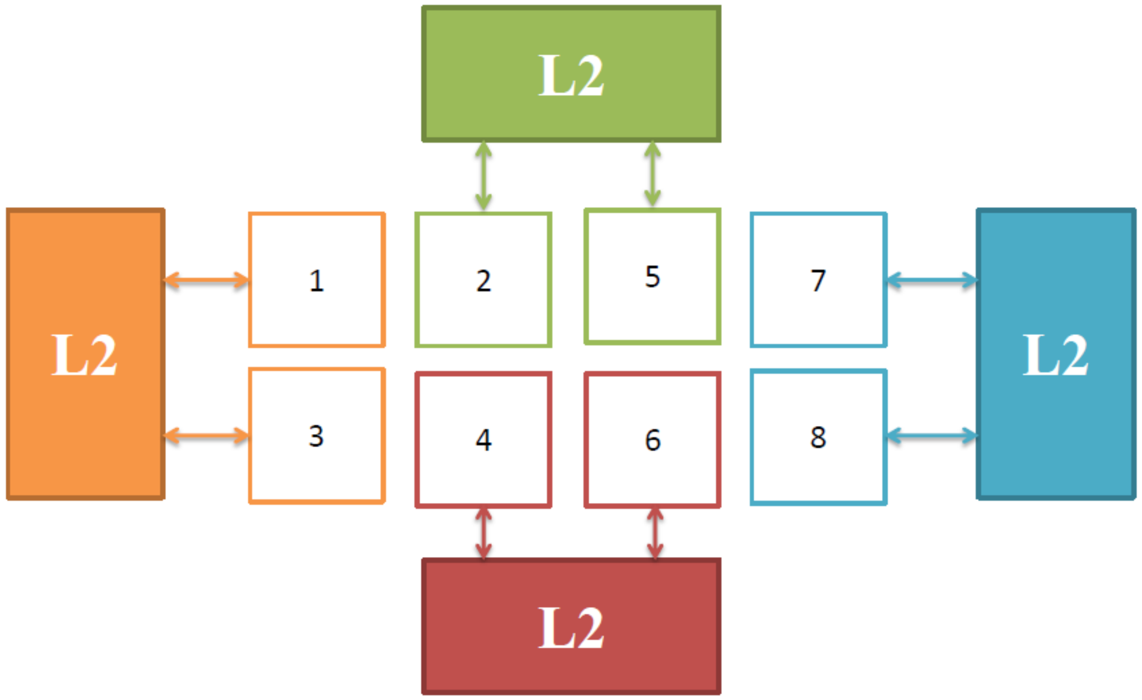

Figure 2 Topology of the experimentation platform. There are eight cores available in the system. Each pair of cores has a shared level 2 cache. The level 2 cache is also the last level cache in this system.

This work uses an established open-source virtualization solution comprised of the kernel based virtual machine (kvm) infrastructure [22][23] and the QEMU [24] frontend. KVM is a full virtualization solution for Linux on x86 hardware containing virtualization extension. It consists of a loadable kernel module that provides kernel infrastructure architecture and a processor specific module. KVM also requires QEMU which is a generic open source machine emulator. QEMU uses dynamic binary translation for machine emulation. We use the perf tools [25] available on the current Linux kernels to collect our data and Linux task set program to pin a VM to a specific CPU core. This pinning of VMs to a specific CPU core makes the task of using hardware counters for a VM easier.

3.2 Workloads

We used the workloads from the PARSEC benchmark suite [26]. The PARSEC benchmark suite is chosen for its research-oriented nature: the workloads are well characterized [27], relatively easy to setup, and the runtime for each application is reasonable. We executed each workload to completion to record its total runtime. We recorded the statistics for each workload for all the metrics that we evaluated. Further, in order to reduce the experiment space and execution time involved, we selected following eight benchmarks from the PARSEC benchmark suite for our experiments. Canneal, Dedup, Facesim, Ferret, Freqmine, Streamcluster, Bodytrack, Swaptions are the names of eight benchmarks. Appendix A has a brief description of these programs. It was ensured that all available types of working set sizes, execution times and types of programs themselves were selected from the PARSEC benchmark suite. Bienia et al. [16] characterized all the programs from the PARSEC benchmark suite; this information was useful for selecting programs from the PARSEC suite for our experiments. The PARSEC benchmark suite provides precompiled binaries for commonly used research platforms. These precompiled binaries, along with the built in utilities of PARSEC benchmark suite, were used to schedule and execute the experiments.

3.3 Experiments

This subsection describes the experiments performed for this study. Each experiment consists of two PARSEC benchmarks scheduled to execute on two CPU cores that share the last level cache (L2 cache). We recorded hardware events using Linux perf tools for each experiment. In each experiment, Linux perf command was executed for the hardware event under consideration. Immediately after that run-parserc utility was called, on each VM, with name of the benchmark to be executed. The perf command recorded data until end of execution of both benchmarks. On completion of the experiment the perf command created a file with information stored about the experiment. This file is queried later to extract per process information. An example script that executes an experiment is available in appendix B of this document. All programs were run to completion and the wall clock running time for each is recorded as a measure of performance.

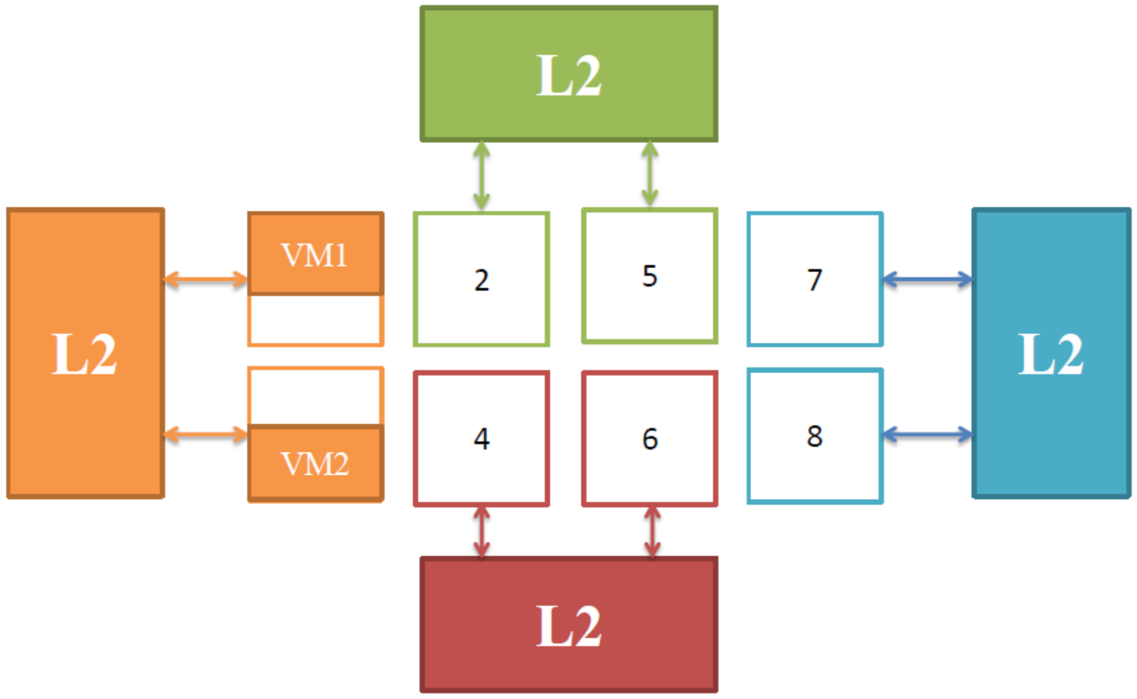

Figure 3 The first set of Experiments performed. Two VMs run on separate cores but they share the L2 cache. The effect of sharing the last level cache on system performance time is investigated in these experiments.

The first set of experiments was aimed at finding out the hardware events that can relate the contention for last level cache with the system performance. Nevertheless, for consistency purposes we not only recorded hardware events related to the last level cache but also recorded other events.

The information collected from these experiments was used to create co-schedules for an eight core machine. A co-schedule consists of eight different PARSEC benchmarks, each running on a different CPU core.

To create a schedule for the eight core machine, four experiments were selected such that eight different programs are present in the schedule. The time of execution for each program is added up to get the total execution time of the schedule. The hardware event count under consideration is also added up to come up with a total count for the schedule.

This procedure is repeated to get all of the possible different schedules and corresponding total execution time and hardware event count. We sorted these schedules according to their total time of execution. The optimal co-schedule is the one that finishes within minimum time on the target platform and the worst case co-schedule takes the longest to finish.

Further, we try to establish a correlation between the total time of execution and the total count of hardware event under consideration. This resulting correlation coefficient indicates if and how closely this hardware counter captures the contention for shared resource under consideration and relates it with overall system performance.

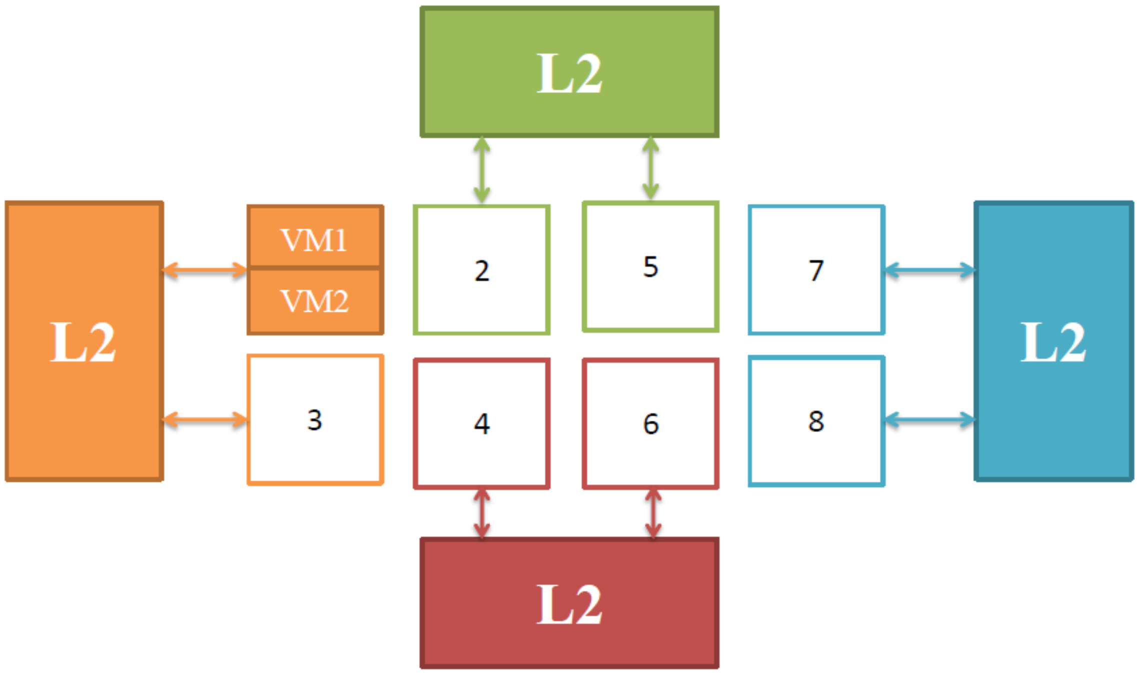

Figure 4: The second set of experiments performed. In these experiments two VMs share a CPU core. Here the effects of sharing a core and resources such as L1 cache that are associated with the core, on system performance is studied.

In the second set of experiments, two virtual machines share the same physical core. Here it was intended to capture the effects virtual machines sharing a core on the system performance. The emphasis of these experiments was on capturing the events related to a single core, such as L1 cache statistics, cycles when reorder buffer or reservation stations are full, cycles when instruction queue is full and so on. These events were selected because they are good candidates to indicate contention for shared resources in a physical core [28]. All the steps performed for first set of experiments were repeated. The results for these experiments are presented in the next section.

Figure 5 below represents third set of experiments performed for this study. In this case four virtual machines were fired up such that two VMs share a core and other two VMs share another core in such a way that two cores (all four VMs) share the last level cache. This set of experiments is interesting because it allows us to evaluate resource contention at what level dictates the system performance and which hardware events, last level cache related or core related or both, exhibit high correlation with performance in these complex scenarios.

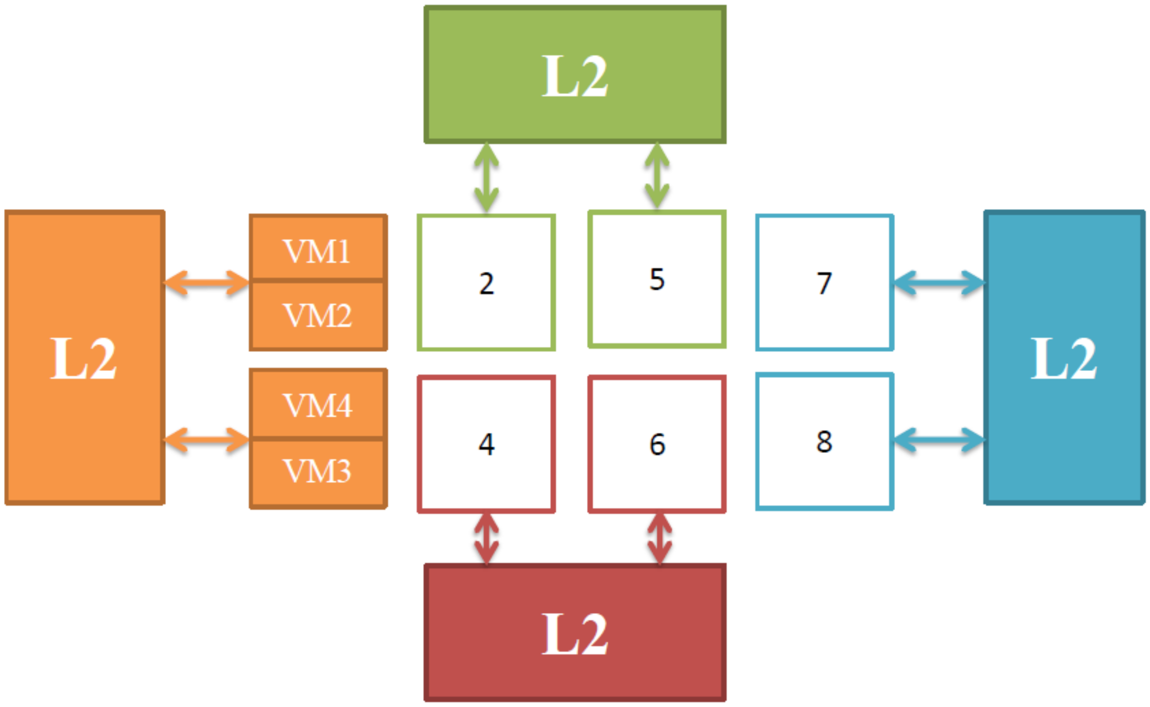

Figure 5: The third set of experiments. In these experiments each core is assigned two VMs. The cores are selected in such that they share the last level cache (the L2 cache).

Figure 6 depicts fourth set of experiments performed for this study. Here we again fired up four virtual machines sharing cores in pairs, but unlike third set of experiments, these cores have separate last level caches (level two caches).

The results for fourth set and third set are to investigate whether the last level cache and core's architectural resources are performance bottlenecks.

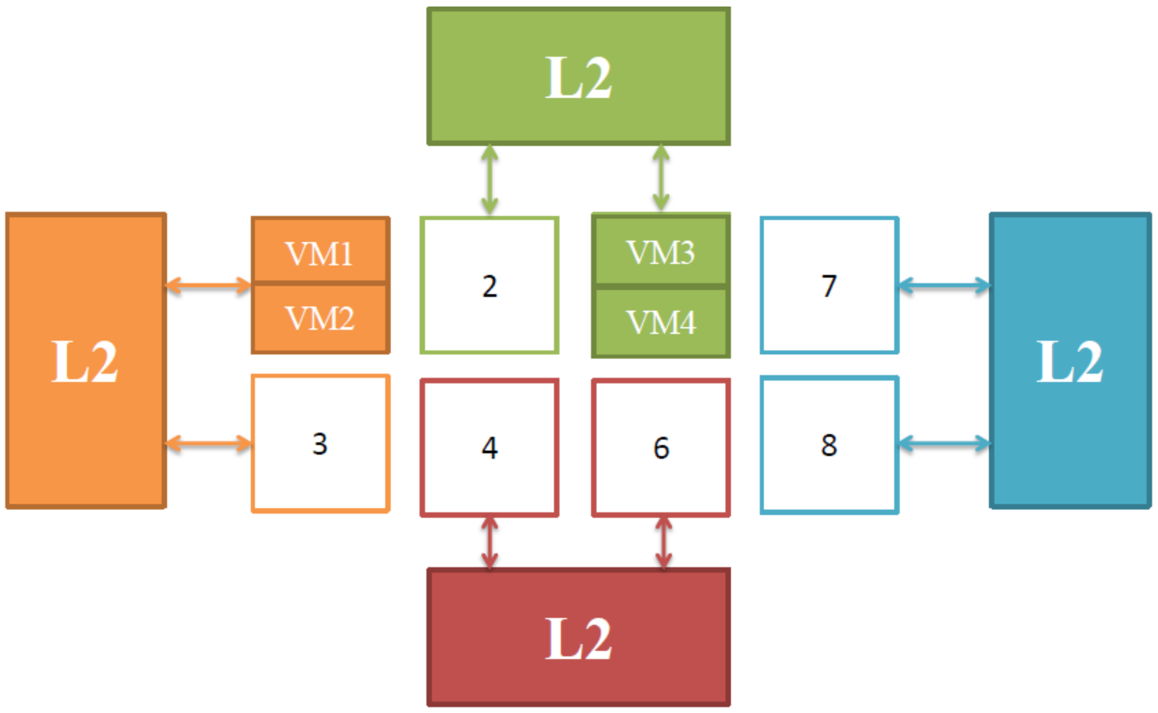

Figure 6: The fourth set of experiments. In these experiments each core is assigned two VMs. The cores are selected in such that they have separate last level cache (the L2 cache).

One of the important goals of this work is to capture the information that can be conveyed to the virtual machine monitor or across the virtual machines and, aid scheduling of the workloads and virtual machines. Thus we selected hardware events for shared resources and investigated if these events exhibit a high correlation with overall system performance. The results for all of the experiments are presented in the next section.

CHAPTER IV

RESULTS

In this section, results from our experiments are presented. As explained in section 3.3 each experiment creates a data file. This data file can be queried to extract the hardware event counts for the experiment. This process of querying the data file and capturing the information were automated using unix and perl scripts. These scripts are included in appendix C. The numbers extracted were inputs to a C program that created schedules, as mentioned in section 3. Section 4.1 illustrates this further with an example.

4.1 Example

This subsection illustrates with an example, how we extracted data from data files created by the perf tools.

The first step in this process is to query the data file using perf record command.

perf kvm --host --guest report -s pid -n --stdio -i " . $file . "

perf kvm –host –guest: This option means that this query is looking at the collected guest OS data.

-s pid: This option sorts the results of the query according to the process ids. The process ids for programs executing for experiment are known and hence statistics for the experiment can be separated from those for background processes of the operating system.

-n: This option ensures that a column with number of samples appears in the output of the query. In other words it simply means this option with –s pid gives per process count of the hardware event.

--stdio –i: This option allows to specify an input file to this query.

A sample output obtained by querying a data file is shown.

```
# Events: 1M LLC-loads
# Overhead Samples         Command:  Pid
# ........ ..........  .....................
   50.06%    831262          qemu-kvm:18099
   22.44%    459262          qemu-kvm:18097
   20.26%    277403          qemu-kvm:18095
    6.94%    124506          qemu-kvm:18093
    0.12%      5074            swapper:    0
    0.04%       221         khugepaged:   60
    0.02%       238      udisks-daemon: 1275
    0.01%       696         irqbalance:  748
    0.01%       763        kworker/2:1:22201
    0.01%       490               ssh:22245
    0.01%       411              perf:22238
    0.01%        97        kworker/0:0:21089
    0.01%       196              perf:22022
    0.00%       235        kworker/u:0:14637
    0.00%       272        kworker/2:2:21910
```

The output is trimmed to include only top 15 rows. As seen in the result above the process ids for programs of the experiment occur in top rows. This due to the fact no other program apart from Linux background processes was executing alongside the experiments.

In the next step, output in above form acts as an input to another script, written in perl, using this program data is arranged in following format.

Benchmark 1	Percentage	**Hardware event count**	**Benchmark 2**	Percentage	**Hardware event count**
Bodytrack	50.14%	14478	Canneal	45.46%	13248
Bodytrack	72.93%	293377	dedup	26.64%	137280
Bodytrack	31.84%	286460	facesim	67.93%	653218
Bodytrack	46.98%	302866	ferret	52.57%	540098
Bodytrack	62.54%	326703	freqmine	36.96%	570936
Bodytrack	28.92%	265136	streamcluster	70.91%	706903
Bodytrack	69.02%	323999	swaptions	30.58%	331129
canneal	63.13%	376236	dedup	36.45%	166254
canneal	21.38%	357703	facesim	78.33%	701113
canneal	34.1%	368753	ferret	65.46%	641583
canneal	49.38%	448961	freqmine	50.05%	635654
canneal	18.91%	269278	streamcluster	80.82%	810538
canneal	56.38%	440902	Swaptions	42.82%	408154
Dedup	14.69%	159312	Facesim	85.06%	686472
Dedup	23.32%	158846	Ferret	76.34%	603922
Dedup	36.84%	180384	Freqmine	62.55%	653184
Dedup	12.51%	129699	Streamcluster	87.32%	730781
Dedup	43.64%	182687	Swaptions	55.87%	410496
Facesim	65.36%	634891	Ferret	34.28%	576131
Facesim	78.53%	805382	Freqmine	21.04%	513650
Facesim	46.28%	509313	Streamcluster	53.48%	779949
Facesim	82.71%	744109	Swaptions	16.97%	366302
Ferret	65.79%	738963	Freqmine	33.51%	514360
Ferret	30.92%	461358	Streamcluster	68.74%	833372
Ferret	71.34%	692381	Swaptions	28.05%	351548
Freqmine	18.6%	352880	Streamcluster	81.03%	916254
Freqmine	56.44%	665622	Swaptions	42.83%	480416
Streamcluster	84.61%	833287	Swaptions	15%	219057

Table 1: An example of hardware event counts gathered in our experiment. Here the hardware event concerned is loads from the last level cache.

The data collected from all of the experiments was arranged as shown. This data acts as an input to another program (see appendix C). This program creates all possible schedules for the eight core machine using the experimental data. This involves selecting four experiments such that eight different benchmarks are selected. Also it is ensured that schedules do not repeat. The hardware event count for each schedule is calculated as total of hardware counts for each benchmarks involved. The schedules are sorted according to this total hardware event count. Similar to the hardware event count, the total time of execution for a schedule is found by adding the total execution time for each benchmark in the schedule.

After finding the total of hardware event count and the total time of execution for all of the schedules, Microsoft excel was used to find the coefficient of correlation between the total hardware event count and the total time of execution. The value of correlation coefficient varies between -1 and 1, a value of correlation coefficient close to 1 means variables involved change closely in tandem. A correlation coefficient of negative one means when one variable increases the other decreases. A value of zero for correlation coefficient means variables involved are uncorrelated. The coefficient of correlation between the total of time of execution and the total of last level cache loads, for above example was calculated to be 0.959.

4.2 Results for first set of experiments following table shows results of our first set of experiments, represented in Figure 3.

Hardware events (performance metric)	Correlation with total time
Cycles for which instruction queue is full	0.9746
Cycles during which ROB is full	0.9653
L1 Accesses	0.9599
Loads from LLC	0.9590
Mispredicted Branches	0.9584
Cycles when Reservation Stations are full	0.9564
Branches	0.9520
L1 misses	0.9477
Stores to LLC	0.8236
Number of lines fetched from memory	0.4732

Table 2: shows hardware events and coefficient of correlation between the total event count and the total of time of execution. These numbers are calculated from results for first set of experiments performed in this work.

Table 2 shows the correlation coefficients obtained after processing the data gathered from the first set of experiments. As seen from the table hardware events that closely correlate with the performance that is the total time of execution are cycles for which instruction queue is full, cycles during which reorder buffer is full, accesses to the level one cache, loads from the last level cache, mispredicted branches and cycles when the reservation stations are full. First these results confirm the importance of the memory related events towards the performance. Second, close correlation of the hardware events related to architectural resources in each core with the performance means utilization of architectural resources reservation station, reorder buffer, instruction queue are good indicators of performance. Third, close correlation of branches and mispredicted branches with performance reaffirms the importance of branch prediction with performance, also shown by Bellosa et al. [13].

The platform under consideration belongs to Intel x86 family of computers, which implements deep pipelining this means a branch is associated with severe performance penalty. As pipeline has to be stalled until the branch is resolved. To avoid this branch prediction is implemented. This way pipeline can use predicted result for the branch and predicted target when a branch occurs. A problem in this scheme is that pipeline has to be flushed if result of the branch instruction is mispredicted. This means that whenever such a misprediction occurs the performance takes a hit. This phenomenon is reflected in the statistics gathered.

In the first set of experiments, varied hardware events were considered in order to find hardware events that can be good statistical indicators of the performance. These results from the first set of experiments also dictate the selection of hardware events for further experiments.

4.3 Results for second set of experiments: The table below shows results for second set of experiments represented in Figure 4.

Hardware events (performance metric)	Correlation with total time
Cycles in which instruction queue is full	0.954
Cycles in which Reservation Stations are full	0.937
L1 misses	0.878
Cycles during which ROB	0.867
LLC loads	0.835
Mispredicted Branches	0.817

Table 3: shows hardware events and coefficient of correlation between the total event count and the total of time of execution. These numbers are calculated from the results for the second set of experiments in this work.

The second set of experiments was intended at study the case where virtual machines share a physical core. The statistics obtained from the results of the second set of experiments show that when two virtual machine share same physical core hardware events related to architectural resources within the core, namely instruction queue, reservation stations, level one cache and, reorder buffer, correlate closely with performance of the system and the coefficient of correlation for hardware event for last level cache takes a lower value than that observed in the first set of experiments where each virtual machine had a core of its own.

4.4 Results for third set of experiments: The table below shows results for third set of experiments represented in Figure 5.

Hardware events (performance metric)	Correlation with total time
Cycles in which Instruction queue is full	0.951
Cycles in which Reservation Stations are full	0.937
LLC loads	0.922
# lines loaded from memory	0.732

Table 4: shows hardware events and coefficient of correlation between the total event count and the total of time of execution. These numbers are calculated from results for third set of experiments performed in this work.

In the third set of experiments virtual machines shared a CPU core and four virtual machines shared a last level cache. The sharing of last level cache means the last level cache becomes important for the overall performance, this is reflected by the increased value of coefficient of correlation for loads from the last level cache. Notice that this value was much less in second set of experiments. Also it can be observed that the number of lines loaded from the memory have a higher correlation with the performance than that observed in the first set of experiments. Because two virtual machines share a core hardware events related to the physical core also exhibit very high correlation with overall system performance.

4.5 Results for fourth set of experiments: The table below shows results for Fourth set of experiments represented in Figure 6.

Hardware events (performance metric)	Correlation with total time
Cycles in which Instruction queue is full	0.957
Cycles in which Reservation Stations are full	0.93
LLC loads	0.926
# lines loaded from memory	0.778

Table 5: shows hardware events and coefficient of correlation between the total event count and the total of time of execution. These numbers are calculated from results for fourth set of experiments performed in this work.

The fourth set of experiments scheduled four virtual machines such that they share CPU cores in pairs but each pair has its own last level cache. The hardware events of the last level cache are closely correlated with the performance because virtual machines share the last level cache in pairs. Another interesting observation is that correlation of the number lines loaded from the memory has increased as compared to the third set of experiments. In this set of experiments each pair of virtual machines has its own last level cache and thus the main memory becomes for important for performance. Higher values of correlation between events related physical core and system performance are present in both third and fourth set of experiments. This means the contention for shared resources within a core influences the the system performance.

CHAPTER V

CONCLUSIONS

As seen from section 4.2 table 1, hardware events related to architectural resources within the core exhibit very high correlation with performance. This reaffirms that the architectural resources within a core are very important for performance as seen by the workloads.

In all of the experiments performed, the count of loads from the last level cache, the cycles for which instruction was full, cycles for which reorder buffers are full, are the hardware events that always exhibited good correlation with overall system performance. These events can be good indicators of resource contention and overall system performance in virtualized system.

When virtual machines are scheduled to share the last level cache the hardware event, which counts the loads coming from last level cache, shows highest correlation with overall system performance, amongst other last level cache related hardware events. Thus the count of the loads coming from the last level cache is a good indicator of the contention between virtual machines for the last level cache and the effect of this contention on overall system performance.

When virtual machines are scheduled to share the same physical core the hardware events, that count cycles for which instruction queue was full, cycles for which reorder buffers were full and, misses in level one cache, exhibit highest values of correlation with overall system performance. Thus cycles for which instruction queue is a full, cycle for which reorder buffers are full and misses from level one cache are good indicators of contention for the core's architectural resources and effect of this contention on overall system performance.

When scheduling of virtual machines is such that virtual machines share a physical core and a last level cache hardware events, that count cycles for which instruction queue was full, cycles for which reorder buffers were full, loads from last level cache, exhibit highest values of correlation with overall system performance. In this case contention occurs at physical core level as well for the last level cache and hence hardware events related to both resources show high correlation with overall system performance. These three events showed highest values of correlation among other events and hence are good indicators of resource contention as well as overall system performance.

For a fixed number of virtual machines, if scheduling is so performed as to reduce the contention for last level cache, hardware event that counts number of loads coming from the memory shows higher correlation with overall system performance. This hardware event is an indicator of contention for main memory and an increase in correlation of loads coming from main memory and overall system performance means contention for memory is affecting the performance of the system. Thus in such scenarios the count of loads coming from memory is a good indicator of contention for memory and overall system performance.

5.1 Future work:

In this work hardware events related to share resources in virtualized systems were studied to see how closely they can correlate the usage of the shared resource with overall system performance. These hardware events captured by the performance counters and details about sharing of resources will act as inputs to the scheduler. The scheduler of a virtualization solution in turn will use this information to take scheduling decisions that will reduce contention for shared resources, reduce energy consumption and still maintain service quality. Studying the scheduler of a virtualized system and modifying it to solve this multivariable optimization problem will make this work usable on a real virtualization solution.

REFERENCES

[1] R. Harms and M.Yamartino, The economics of the cloud, white paper, Microsoft Corporation.

[2] M. Armbrust, et al. *Above the clouds: A Berkeley view of cloud computing.* Tech. Rep. UCB/EECS-2009-28,EECS Department, U.C. Berkeley, Feb 2009.

[3] INTEL CORPORATION. *Intel R_ Virtualization Technology Specification for the IA-32 Intel R_ Architecture*, April 2005.

[4] T. Anderson, L. Peterson, S. Shenker, and J. Turner. *Overcoming the Internet impasse through virtualization.* IEEE Computer, 38(4):34–41, Apr. 2005.

[5] Ristenpart, Thomas,et al *"Hey, you, Get Off of MY Cloud: Exploring Information Leakage in Third-Party Compute Clouds.*

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, *Xen and the art of virtualization,* in: Proc. 19th ACM Symposium on Operating Systems Principles, SOSP 2003, Bolton Landing, USA, Oct. 2003.

[7]Y Koh, R. C. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, *"An analysis of performance interference effects in virtual environments,"* in ISPASS. IEEE Computer Society, 2007, pp. 200-209.

[8] T. Wood, L. Cherkasova, K. M. Ozonat and P. J. Shenoy. *Profiling and modeling resource usage of virtualized applications.* In Val´erie Issarny and Richard E. Schantz, editors, Middleware, volume 5346 of Lecture Notes in Computer Science, pages 366–387. Springer, 2008.

[9] D. A. Menascé, *"Virtualization: Concepts, applications, and performance modeling,"* in Proc. 31th Int. Computer Measurement Group Conf., Orlando, FL, 2005, pp. 407–414.

[10] J.N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, *Quantifying the Performance Isolation Properties of Virtualization Systems, ACM Workshop on Experimental Computer Science (ExpCS).*

[11] D. Gupta, L. Cherkasova, Rob Gardner, and Amin Vahdat *Enforcing Performance Isolation across Virtual Machines in Xen.* HP Labs, Technical Report HPL-2006-77.

[12] A. Fedorova, S. Blagodurov, and S. Zhuravlev. *Managing contention for shared resources on multicore processors.* Communications of the ACM, 53(2), 2010.

[13] S. Zhuravlev, S. Blagodurov, and A. Fedorova. *Addressing shared resource contention in multicore processors via scheduling.* In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2010.

[14] M. Ghosh, R. Nathuji, M. Lee, K. Schwan, H. Hsin S. Lee *Symbiotic Scheduling for shared caches in Multi-core Systems using memory footprint signature* arch.ece.gatech.edu.

[15] A.Verma, P. Ahuja, A. Neogi *Power-aware Dynamic Placement of HPC Application*, ICS'08, June 7–12, 2008.

[16] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. *Event-driven energy accounting for dynamic thermal management.* In Proceedings of the Workshop on Compilers and Operating Systems for Low Power (COLP'03), New Orleans, LA, September 27 2003.

[17] R. Ge, X. Feng, and K. W. Cameron. *Modeling and evaluating energy-performance efficiency of parallel processing on multicore based poweraware systems.* Parallel and Distributed Processing Symposium, International, 0:1{8, 2009.

[18] R. Knauerhase, P. Brett, B. Hohlt, T. Li, and S. Hahn. *Using OS Observations to Improve Performance in Multicore Systems*. IEEE Micro, 28(3):54–66, 2008.

[19] K.H. Kim, R. Buyya, J. Kim, *Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters*, in: Proc. 7th IEEE Int. Symposium on Cluster Computing and the Grid, CCGrid 2007, Rio de Janeiro, Brazil, May 2007.

[20] Singh, K., Bhadauria, M., McKee, S., *Real Time Power Estimation and Thread Scheduling via Performance Counters*. In ACM SIGARCH Computer Architecture News, vol 37 issue 2, may 2009.

[21] Intel xeon L5410 processor *http://ark.intel.com/products/33090/Intel-Xeon-Processor-L5410-(12M-Cache-2_33-GHz-1333-MHz-FSB)*

[22] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. *kvm: the Linux virtual machine monitor. In OLS '07: The 2007 Ottawa Linux Symposium, pages 225–230, July 2007.*

[23] T. Deshane, Z. Shepherd, J.N. Matthews, M. Ben-Yehuda, A. Shah, and B. Rao. *Quantitative comparison of Xen and KVM.* Xen Summit, Boston, MA, USA, pages 1–2, 2008.

[24] QEMU generic and open source machine emulator and virtualizer http://wiki.qemu.org/Main_Page.

[25] perf: linux profiling with performance counters. https://perf.wiki.kernel.org/

[26] C. Bienia. Ph.D. Thesis. *Benchmarking Modern Multiprocessors,* Princeton University, January 2011.

[27] C. Bienia, S. Kumar, et al. *The parsec benchmark suite: Characterization and architectural implications.* Tech. Rep. TR-811-08, Princeton University, 2008.

[28] Intel IA- 32 architecture Software Developer's Manual

APPPENDICES

Appendix A

Description of benchmark programs from PARSEC suite used in our experiments.

1) **Canneal**: This kernel was developed by Princeton University. It uses cache-aware simulated annealing (SA) to minimize the routing cost of a chip design . Canneal uses fine-grained parallelism with a lock-free algorithm and a very aggressive synchronization strategy that is based on data race recovery instead of avoidance.
2) **Dedup**: This kernel was developed by Princeton University. It compresses a data stream with a combination of global and local compression that is called 'deduplication'. The kernel uses a pipelined programming model to mimic real-world implementations. The reason for the inclusion of this kernel is that deduplication has become a mainstream method for new generation backup storage systems.
3) **Facesim:** This Intel RMS application was originally developed by Stanford University. It computes a visually realistic animation of the modeled face by simulating the underlying physics. The workload was included in the benchmark suite because an increasing number of animations employ physical simulation to create more realistic effects.
4) **Ferret:** this application is based on the Ferret toolkit which is used for content-based similarity search. It was developed by Princeton University. The reason for the inclusion in the benchmark suite is that it represents emerging next-generation search engines for non-text document data types. In the benchmark, we have configured the Ferret toolkit for image similarity search. Ferret is parallelized using the pipeline model.

5) **Freqmine:** This application employs an array-based version of the FP-growth (Frequent Pattern-growth) method for Frequent Itemset Mining (FIMI). It is an Intel RMS benchmark which was originally developed by Concordia University. Freqmine was included in the PARSEC benchmark suite because of the increasing use of data mining techniques.
6) **Streamcluster**: This RMS kernel was developed by Princeton University and solves the online clustering problem. Streamcluster was included in the PARSEC benchmark suite because of the importance of data mining algorithms and the prevalence of problems with streaming characteristics.
7) **Bodytrack**: This computer vision application is an Intel RMS workload which tracks a human body with multiple cameras through an image sequence. This benchmark was included due to the increasing significance of computer vision algorithms in areas such as video surveillance, character animation and computer interfaces.
8) **Swaptions:** The application is an Intel RMS workload which uses the Heath-Jarrow-Morton (HJM) framework to price a portfolio of swaptions. Swaptions employs Monte Carlo (MC) simulation to compute the prices.

Appendix B

This appendix has an example script that was used to execute an experiment.

```
perf kvm --guest --host --guestmount /mnt/guest-5555-root record -e
$5 -a &
time ssh -p5554 localhost "./run-parsec.sh $1 native 1" &
pid0=$!
time ssh -p5555 localhost "./run-parsec.sh $2 native 1" &
pid1=$!
wait $pid0
wait $pid1
killall -s SIGINT perf
mv perf.data.kvm ./output-data/$5-5554-5555-$1-$2
```

The Steps performed in this script are

1) Perf record command is executed with $5 as the parameter which is the mask for the hardware event taken from Intel's manual for xeon machine.
2) Two parsec benchmarks are started within two different virtual machines.
3) The script runs until both benchmarks finish and records the hardware event count.
4) In the end it saves the data file and kills the perf command

Appendix C

Appendix C describes and presents the scripts used for processing the data gathered in experiments.

1) The Perl script below processes the perf file that has captured the count of loads coming to the last level cache. It queries the data file and store the query output in another text file.

```
#!/usr/bin/perl -w
use strict;
use warnings;
my @files = </root/output-data/LLC-loads-5554-5555*>;
my $file;
foreach $file (@files){
    system "echo " . $file . ">>~/myout \n";
    system "perf kvm --host --guest report -s pid -n --stdio -i " . $file . " | grep \"31548\\|31550\\" 
>>~/myout\n";
}
```

2) #!/usr/bin/perl -w
```
use strict;
use warnings;

my @benchmark1;
my @benchmark2;
my @array;
my $file = "myout";
my $fileout = ">>results.csv";
my $perfcounter = "empty";
my $pctr;
my @p;
open FILE, $file;
my $flag1 = 0;
my $flag2 = 0;
my $flag3 = 0;
#open MYFILE, $fileout;
# read the file line by line
my $line;
foreach $line(<FILE>){
    chomp($line);
    if($flag1 == 1 && $flag2 == 1 && $flag3 == 1){
            open MYFILE,">>",$fileout;
            print MYFILE @benchmark1;
            print MYFILE @benchmark2;
            print MYFILE "\n";
            close MYFILE;
            @benchmark1 = ();
            @benchmark2 = ();
            $flag1 = 0;
            $flag2 = 0;
            $flag3 = 0;
    }
    if(index($line, "5554") != -1){
                    # print the data to file
                    # clear all the arrays
                    @array = ();
                    @array = split(/-/,$line);
                    #Benchmark 1
                    push(@benchmark2, pop(@array));
                    push(@benchmark2,',');
                    #Benchmark 2
                    push(@benchmark1, pop(@array));
                    push(@benchmark1,',');
                    #What are we counting ?
                    pop(@array);
                    pop(@array);
                    $pctr = pop(@array);
                    @p = ();
                    @p = split(/\//,$pctr);
                    $pctr = pop(@p);
                    if($pctr ne  $perfcounter){
                            $fileout = $pctr.".csv";
                            #print $fileout;
                            $perfcounter = $pctr;
                            #close MYFILE;
                            open MYFILE,">",$fileout;
                            print MYFILE "Benchamark1".",".""Percentage".",".""Count".",";
                            print MYFILE "Benchmark2".",".""Percentage".",".""Count";
                            print MYFILE "\n";
                            #print $fileout;
                            close MYFILE;
                    }
                    $flag1 = 1;
            }
    #Case 2 We are on second line get the stats for one benchmark
    elsif(index($line,"324") != -1){
            @array = split(' ',$line);
            if(index($line, "32469") != -1 ){
                    #Percent 1
                    push(@benchmark1, shift(@array));
                    push(@benchmark1,',');
                    #count 1
                    push(@benchmark1, shift(@array));
                    push(@benchmark1, ',');
                    $flag2 = 1;
            }
            elsif(index($line, "32471") != -1){
                    #percent 2
                    push(@benchmark2,shift(@array));
                    push(@benchmark2,',');
                    #count 2
                    push(@benchmark2,shift(@array));
                    $flag3 = 1;
            }
    }
}
close FILE;
close MYFILE;
```

This perl script processes the text output generated by script one described in this appendix. This script produces results for each experiment. It gives out details about each experiment, the names of the benchmarks executed and hardware event count associated with each benchmark

1	2	3	4	5	6	7	8	time	LLC loads
freqmine	ferret	streamclus	dedup	swaptions	canneal	facesim	bodytrack	4294	4070235
swaptions	ferret	streamclus	dedup	freqmine	canneal	facesim	bodytrack	4298	4073247
swaptions	freqmine	streamclus	dedup	ferret	canneal	facesim	bodytrack	4322	4081275
freqmine	ferret	swaptions	facesim	streamclus	dedup	canneal	bodytrack	4326	4113140
freqmine	ferret	facesim	dedup	swaptions	canneal	streamclus	bodytrack	4327	4073625
swaptions	facesim	streamclus	dedup	freqmine	canneal	ferret	bodytrack	4328	4114465
swaptions	ferret	facesim	dedup	freqmine	canneal	streamclus	bodytrack	4331	4076637
swaptions	facesim	freqmine	dedup	ferret	canneal	streamclus	bodytrack	4331	4104561
swaptions	facesim	streamclus	dedup	ferret	canneal	freqmine	bodytrack	4334	4106009
swaptions	ferret	freqmine	dedup	facesim	canneal	streamclus	bodytrack	4335	4105437
streamclus	freqmine	ferret	dedup	swaptions	canneal	facesim	bodytrack	4338	4112264
swaptions	ferret	streamclus	dedup	facesim	canneal	freqmine	bodytrack	4338	4106885
ferret	facesim	freqmine	dedup	swaptions	canneal	streamclus	bodytrack	4338	4098164
swaptions	facesim	ferret	dedup	freqmine	canneal	streamclus	bodytrack	4339	4100031
ferret	facesim	streamclus	dedup	swaptions	canneal	freqmine	bodytrack	4341	4099612
freqmine	facesim	streamclus	dedup	swaptions	canneal	ferret	bodytrack	4343	4098999
swaptions	ferret	freqmine	facesim	streamclus	dedup	canneal	bodytrack	4345	4100686
freqmine	facesim	ferret	dedup	swaptions	canneal	streamclus	bodytrack	4354	4084565
swaptions	freqmine	facesim	dedup	ferret	canneal	streamclus	bodytrack	4355	4084665
swaptions	freqmine	streamclus	dedup	facesim	canneal	ferret	bodytrack	4356	4123369
freqmine	ferret	streamclus	dedup	facesim	canneal	swaptions	bodytrack	4356	4131962
freqmine	ferret	swaptions	facesim	dedup	canneal	streamclus	bodytrack	4358	4114943
streamclus	ferret	freqmine	dedup	swaptions	canneal	facesim	bodytrack	4359	4123240
streamclus	freqmine	facesim	dedup	swaptions	canneal	ferret	bodytrack	4360	4130088
swaptions	freqmine	ferret	facesim	streamclus	dedup	canneal	bodytrack	4361	4117783
streamclus	freqmine	swaptions	ferret	dedup	canneal	facesim	bodytrack	4361	4130188
streamclus	freqmine	swaptions	ferret	facesim	dedup	canneal	bodytrack	4362	4131775
ferret	facesim	streamclus	dedup	freqmine	canneal	swaptions	bodytrack	4363	4127701
swaptions	freqmine	ferret	dedup	facesim	canneal	streamclus	bodytrack	4367	4108935
streamclus	freqmine	swaptions	facesim	ferret	dedup	canneal	bodytrack	4370	4155169
freqmine	facesim	streamclus	dedup	ferret	canneal	swaptions	bodytrack	4371	4118632
streamclus	freqmine	swaptions	dedup	ferret	canneal	facesim	bodytrack	4372	4144039
freqmine	ferret	swaptions	dedup	facesim	canneal	streamclus	bodytrack	4373	4129670
swaptions	ferret	freqmine	facesim	dedup	canneal	streamclus	bodytrack	4377	4102489
ferret	facesim	swaptions	dedup	freqmine	canneal	streamclus	bodytrack	4380	4125409
streamclus	freqmine	facesim	dedup	ferret	canneal	swaptions	bodytrack	4388	4149721
freqmine	facesim	swaptions	dedup	ferret	canneal	streamclus	bodytrack	4388	4116340
streamclus	freqmine	swaptions	facesim	dedup	canneal	ferret	bodytrack	4391	4171406
streamclus	ferret	swaptions	facesim	freqmine	dedup	canneal	bodytrack	4391	4166145
swaptions	freqmine	ferret	facesim	dedup	canneal	streamclus	bodytrack	4393	4119586
streamclus	ferret	facesim	dedup	swaptions	canneal	freqmine	bodytrack	4395	4128078
streamclus	freqmine	ferret	dedup	facesim	canneal	swaptions	bodytrack	4400	4173991
streamclus	ferret	swaptions	dedup	freqmine	canneal	facesim	bodytrack	4401	4150485
streamclus	freqmine	swaptions	facesim	ferret	canneal	dedup	bodytrack	4404	4178287
streamclus	freqmine	swaptions	dedup	facesim	canneal	ferret	bodytrack	4406	4186133

streamclus	freqmine	swaptions	facesim	ferret	canneal	dedup	bodytrack	4404	4178287
streamclus	freqmine	swaptions	dedup	facesim	canneal	ferret	bodytrack	4406	4186133
streamclus	freqmine	swaptions	ferret	facesim	canneal	dedup	bodytrack	4408	4179163
streamclus	freqmine	ferret	facesim	swaptions	dedup	canneal	bodytrack	4411	4180547
streamclus	freqmine	ferret	facesim	swaptions	canneal	dedup	bodytrack	4411	4171890
swaptions	ferret	freqmine	dedup	streamclus	canneal	facesim	bodytrack	4411	4179476
swaptions	freqmine	streamclus	ferret	dedup	canneal	facesim	bodytrack	4414	4144662
swaptions	freqmine	streamclus	ferret	facesim	dedup	canneal	bodytrack	4415	4146249
streamclus	ferret	facesim	dedup	freqmine	canneal	swaptions	bodytrack	4417	4156167
streamclus	ferret	freqmine	dedup	facesim	canneal	swaptions	bodytrack	4421	4184967
streamclus	freqmine	ferret	facesim	dedup	canneal	swaptions	bodytrack	4426	4184642
streamclus	ferret	swaptions	facesim	dedup	canneal	freqmine	bodytrack	4426	4169396
streamclus	ferret	swaptions	facesim	freqmine	canneal	dedup	bodytrack	4433	4184733
streamclus	ferret	swaptions	dedup	facesim	canneal	freqmine	bodytrack	4441	4184123
swaptions	facesim	freqmine	dedup	streamclus	canneal	ferret	bodytrack	4441	4220694
swaptions	freqmine	ferret	dedup	streamclus	canneal	facesim	bodytrack	4443	4182974
swaptions	ferret	facesim	dedup	streamclus	canneal	freqmine	bodytrack	4447	4184314
streamclus	ferret	freqmine	facesim	swaptions	dedup	canneal	bodytrack	4448	4177924
streamclus	ferret	freqmine	facesim	swaptions	canneal	dedup	bodytrack	4448	4169267
freqmine	ferret	swaptions	dedup	streamclus	canneal	facesim	bodytrack	4449	4203709
swaptions	facesim	ferret	dedup	streamclus	canneal	freqmine	bodytrack	4455	4207708
swaptions	freqmine	streamclus	ferret	facesim	canneal	dedup	bodytrack	4461	4193637
streamclus	ferret	freqmine	facesim	dedup	canneal	swaptions	bodytrack	4463	4182019
swaptions	freqmine	facesim	dedup	streamclus	canneal	ferret	bodytrack	4465	4200798
freqmine	ferret	facesim	dedup	streamclus	canneal	swaptions	bodytrack	4465	4209391
ferret	facesim	freqmine	dedup	streamclus	canneal	swaptions	bodytrack	4476	
freqmine	ferret	swaptions	facesim	streamclus	canneal	dedup	bodytrack	4481	4237957
streamclus	facesim	freqmine	dedup	swaptions	canneal	ferret	bodytrack	4483	4257181
swaptions	ferret	streamclus	facesim	freqmine	dedup	canneal	bodytrack	4485	4258868
freqmine	facesim	ferret	dedup	streamclus	canneal	swaptions	bodytrack	4492	4220331
ferret	facesim	swaptions	dedup	streamclus	canneal	freqmine	bodytrack	4496	4233086
streamclus	facesim	ferret	dedup	swaptions	canneal	freqmine	bodytrack	4497	4244195
freqmine	facesim	swaptions	dedup	streamclus	canneal	ferret	bodytrack	4498	4232473
swaptions	ferret	freqmine	facesim	streamclus	canneal	dedup	bodytrack	4500	4225503
streamclus	facesim	freqmine	dedup	ferret	canneal	swaptions	bodytrack	4511	4276814
swaptions	freqmine	ferret	facesim	streamclus	canneal	dedup	bodytrack	4516	4242600
swaptions	freqmine	streamclus	facesim	ferret	dedup	canneal	bodytrack	4517	4262366
streamclus	facesim	ferret	dedup	freqmine	canneal	swaptions	bodytrack	4519	4272284
swaptions	ferret	streamclus	facesim	dedup	canneal	freqmine	bodytrack	4520	4262119
freqmine	ferret	streamclus	facesim	swaptions	dedup	canneal	bodytrack	4523	4283101
freqmine	ferret	streamclus	facesim	swaptions	canneal	dedup	bodytrack	4523	4274444
streamclus	facesim	swaptions	dedup	freqmine	canneal	ferret	bodytrack	4525	4284426
swaptions	ferret	streamclus	facesim	freqmine	canneal	dedup	bodytrack	4527	4277456
streamclus	facesim	swaptions	dedup	ferret	canneal	freqmine	bodytrack	4531	4275970
swaptions	freqmine	streamclus	facesim	dedup	canneal	ferret	bodytrack	4538	4278603
freqmine	ferret	streamclus	facesim	dedup	canneal	swaptions	bodytrack	4538	4287196

Table 6: **S**chedules generated according to hardware event loads form the last level cache and mapped with total time of execution for each schedule.

VITA

Pranav Suresh Pathak

Candidate for the Degree of

Master of Science

Thesis: QUANTIFYING THE EFFECTS OF SHARED RESOURCE CONTENTION ON PERFORMANCE IN VIRTUALIAZED SYSTEMS

Major Field: Electrical Engineering

Biographical:

Education:

Completed the requirements for the Master of Science in your major at Oklahoma State University, Stillwater, Oklahoma in December, 2011.

Completed the requirements for the Bachelor of Science in Electronics and Telecommunication at Pune University, Pune India in December 2007.

Experience: Employed by Oklahoma State University, as research assistant and teaching assistant 2009 –present.
Employed by Infosys technologies limited as a software engineer 2007-2009.

Name: Pranav Pathak Date of Degree: December, 2011

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: QUANTIFYING THE EFFECTS OF SHARED RESOURCE CONTENTION ON PERFORMANCE IN VIRTUALIAZED SYSTEMS

Pages in Study: 39 Candidate for the Degree of Master of Science

Major Field: Electrical Engineering

Scope and Method of Study: This study explores the use of hardware performance counters to capture correlation between shared resource usage and performance of the system, in a virtualized computer system.

Findings and Conclusions: This thesis found hardware events that can be used in different scheduling scenarios to capture the correlation between contention for shared resource and overall system performance. Schedulers in virtualized systems need to be studied and modified to leverage these hardware events for better scheduling decisions.

ADVISER’S APPROVAL: Dr. Sohum A. Sohoni