

Evaluating the Performance Characteristics of a Virtual Machine Used on Simultaneous
Multi-Threaded (SMT) Processors

by

HiuShanYim

A Thesis submitted to the

Graduate School-New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

written under the direction of

Prof. Pompili

and approved by

New Brunswick, New Jersey

January 2000

Abstract of the Thesis

Evaluating the Performance Characteristics of a Virtual Machine Used on Simultaneous Multi-Threaded (SMT) Processors

by HiuShan Yim

Dissertation Director: Prof Dario Pompili

Virtualization of computing hardware is one technique which can make possible the use of fewer physical computers, thus lowering resource consumption. Today, as in the past, hardware performance remains a major bottleneck to virtual machine performance. Simultaneous multithreaded, or SMT, processors provide thread-level parallelism and are being used to overcome the performance limitations of virtual machines. These same processors are also being used to decrease the cost of computing systems since less hardware and power is required when compared with multiple CPU systems. Virtual machines should benefit from the properties of SMT processors since they have a common cache and parallel execution threads. As a result, using virtual machines in combination with SMT processors should be an efficient way to maintain or increase performance, save money and reduce physical hardware requirements. This study attempts to determine if an improvement on virtual machine performance exists through the use of an SMT processor. If the performance of an SMT processor-based system is on-par with several independent computers or multiple CPU systems, then the use SMT would be an efficient way for organizations to achieve their performance requirements at a reduced cost. This thesis evaluates the performance of a virtual machine used with and

without SMT. This study shows that a definite, measurable performance improvement exists when a virtual machine is run with an SMT processor and that better virtual machine performance is achieved as load increases. Then a performance modeling method is suggested for various combinations of SMT and virtual machines in order to predict and maximize system performance and achieve proper load balancing.

Acknowledgement

I would like to express my thanks to Professor Manish Parashar and Professor Pompili for their guidance and advice.

Table of Content

Acknowledgement.....	iv
Table of Content.....	v
List of Figures	vii
List of Tables.....	viii
1 Introduction	1
1.1 Problems with available computational resources and possible performance improvement	1
1.2 Simultaneous multithreading (SMT)	2
1.2.1 Comparison of SMT and fine-grained multi-threading (MT).....	3
1.2.2 Comparison of SMT and Chip Multiprocessing (CMP).....	4
1.2.3 Comparison of SMT and Symmetric Multiprocessing (SMP).....	5
1.3 CPU examples	6
1.3.1 Intel Hyper-Threading	6
1.3.2 IBM Power 5 SMT	6
1.4 Virtualization.....	7
1.4.1 Introduction to VMware	8
1.4.2 VMware with Simultaneous Multi-processors	9
1.4.3 Introduction to Hypervisor	10
1.4.4 Cost of Virtualization versus Performance Benefits	10
1.5 Main Contribution of this Thesis	10
2 Experimental Setup	12
2.1 Introduction	12
2.2 Problem Formulation and system description.....	12
2.2.1 Introduction to the Processor being used.....	12
2.2.2 Introduction to the Operation System being used.....	13
2.2.3 Introduction to the virtual machine being used.....	16
2.2.4 Introduction to the Load and Benchmark	17
2.2.5 Rationality behind the choice of benchmark and load.....	17
2.2.6 Baseline: base operating system without virtual machine without SMT	21
2.2.7 Base operating system without virtual machine with SMT	22
2.2.8 One virtual machine on base operating system without SMT	23
2.2.9 One virtual machine on base operating system with SMT	24
2.3 System Setup Procedure	25
2.3.1 Setup Oracle and Swingbench	25
2.3.2 Tuning SMT	26
2.4 Conclusions	27
3 Performance Metric and Selection.....	29
3.1 Results collected.....	29
3.2 Problem Formulation	34
3.2.1 Introduction to problem formulation	34
3.2.2 Simple Result Analysis.....	35
3.3 Modeling methods used.....	41
3.3.1 Previously suggestion methods.....	41
3.3.2 Modeling for SMT and VM Combined	42
3.3.3 Load Balancing and Load Distribution.....	45
4 Thesis Conclusion	47
4.1 Virtualization Challenges and SMT Advantages.....	47
4.2 Performance Determination and Analysis	48
4.3 Suggestions.....	48
5 Future Work	49
5.1 Multiple virtual machines with SMT mode.....	49
5.2 VMware with Oracle	50

5.3	Discussion on IBM P595 Micro partitioning.....	50
	Reference.....	51
	Appendix	54
A.	Terminology	54

List of Figures

Figure 1.	Comparison of multi-threading and SMT	4
Figure 2.	Comparison of chip multiprocessing and SMT	4
Figure 3.	SMT architecture	5
Figure 4.	ESX server structure diagram	9
Figure 5.	The O(1) scheduler	15
Figure 6.	Scheduler also supports hyper-threading	16
Figure 7.	Swing Bench	18
Figure 8.	Oracle Architecture Overview	19
Figure 9.	Illustration of test system setting	21
Figure 10.	Oracle System Monitor-top	29
Figure 11.	Oracle Monitoring—top modules	30
Figure 12.	Oracle Console Data Collection	30
Figure 13.	CPU time per each Transaction	36
Figure 14.	Average Response Time	37
Figure 15.	Throughput (Oracle Transactions Per Second)	38
Figure 16.	Memory Free in %	39
Figure 17.	IO Statistics	41

List of Tables

Table I.	Top monitoring sessions and host machine monitoring session	32
Table II.	Sample result for database “insert” benchmark	33
Table III.	Details of all results collected	34
Table IV.	Legend for Table II,III and Color Scheme Used in the Graphs	34

1 Introduction

1.1 Problems with available computational resources and possible performance improvement

This thesis studies the possible performance improvement of combining the use of simultaneous multithread processors with the virtualization of operating systems. The cost of computation in large organizations and corporations is enormous. For instance, there are many costs associated with maintaining a laboratory of hundreds of computers. In addition to machine costs, there are operational costs such as energy costs, cooling costs and space costs, plus administrative costs which also rise as the quantity and variety of systems increase. Therefore, if it is possible to reduce the number physical computers and still meet well-defined performance requirements, environmentally friendly cost savings can be realized.

Virtualization is a technique that abstracts away the hardware from the operating system so that multiple operating systems can share a single physical system at the same time. This is a way to save resources. For instance, an organization can install multiple servers in a single physical host. When each of these virtual machines has different peak usage times, they can share the available resources without interfering each other. [12.][13.] However, this approach has not been considered a viable option due to virtual machine overhead and degradation of application speed within the virtual machine. Thus, system designers have been ignoring the advantages of virtual machines simply because early virtual machine implementations were often the source of performance bottlenecks.

Processors with high computation power, such as multi-core processors and simultaneous multithreaded processors are becoming more desirable due to the fact that they have properties similar to symmetric multiprocessors. The main advantage of symmetric multiprocessors is that they can achieve parallel processing more efficiently than regular high frequency processors where thread switching is avoided. Through the use of simultaneous multithreading and multi-core processors, the improvements in performance of virtualization have been shown to be statistically significant.

1.2 Simultaneous multithreading (SMT)

Simultaneous multithreading is the ability of the microprocessor to fetch instructions from multiple threads per cycle. Simultaneous multithreading combines hardware features seen in two other types of processors: wide-issue superscalars and multithreaded processors. From superscalars it inherits the ability to issue multiple instructions each cycle; and similar to multithreaded processors it can execute several programs or threads at once. The result is a processor that can issue multiple instructions from multiple threads each cycle. [1.][2.][3.][4.] The SMT approach attacks the two major impediments to processor utilization – long latencies and limited per-thread parallelism. SMT processors have a larger register file, and thus it has the capacity to hold data for multiple threads. There can be 4, or even 8, concurrent threads. These multiple threads can execute different instructions in the same clock cycle. Out-of-order executions are supported in SMT processors which allows for additional performance improvement.

There are several architectural requirements to achieve SMT. First, multiple program counters are needed and a mechanism by which the fetch unit selects one counter each cycle; second, a separate return stack for each thread for predicting

subroutine return destinations; third, per thread instruction retirement, instruction queue flush, and trap mechanisms; forth; a thread id, each with branch target buffer entry to avoid predicting phantom branches; and fifth, a large register file, to support logical registers for all the threads in addition to register renaming.

The extra threads supported by SMT processors can be used to proactively seed a shared resource like cache, to improve the performance of another single thread. One of the other uses of SMT is to provide redundant computation, for error detection and recovery.

Companies give different names to SMT processors. Intel calls the SMT feature Hyper-Threading and is included in the Pentium 4 processor family. IBM uses the term Simultaneous Multithreading, which is essentially the same as Intel's "Hyper-Threading". Further details of the processors will be discussed in section 1.3.

Figures 1 and 2 are illustrations of the corresponding issue slots for different processor types. Each color (red, yellow, blue and green) represents a different thread. The follow sections 1.2.1-1.2.3 compare the issue slots in various processors. [3.]

1.2.1 Comparison of SMT and fine-grained multi-threading (MT)

Fine-grained multithreading occurs by time slicing, wherein a single processor switches between different threads, in which case the processing is not literally simultaneous. It is actually a single processor executing instruction serially. However, simultaneous multithreading is literally a processor executing all the threads at the same time.

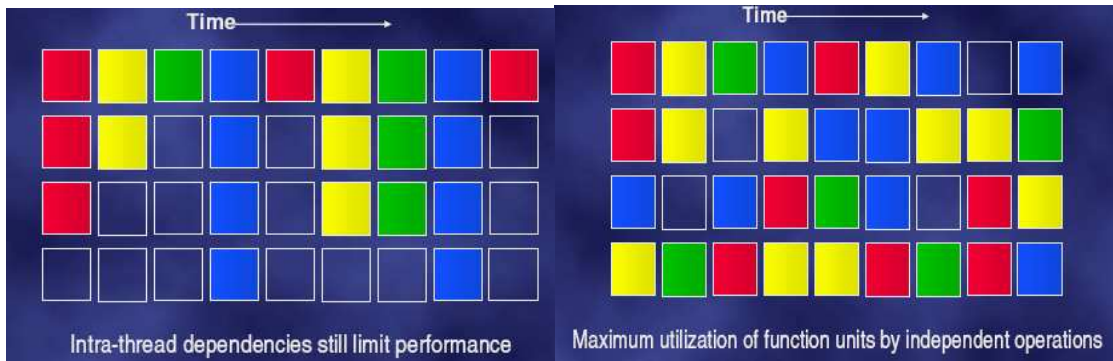


Figure 1. Comparison of multi-threading and SMT

In the Figure 1, each color represents a thread. The illustration on the left shows fine-grained multi-threading (MT), and on the illustration on the right depicts SMT. In MT, each issue slot is occupied by one thread and there are redundant resources since the pipeline is not full. In SMT, each issue slot can have multiple threads.

1.2.2 Comparison of SMT and Chip Multiprocessing (CMP)

Chip multiprocessing occurs on a multi-core processor. Much of the instruction execution logic is shared between the two cores, but each core has its own register set, including any related addressing. This helps keep the processing pipelines full.

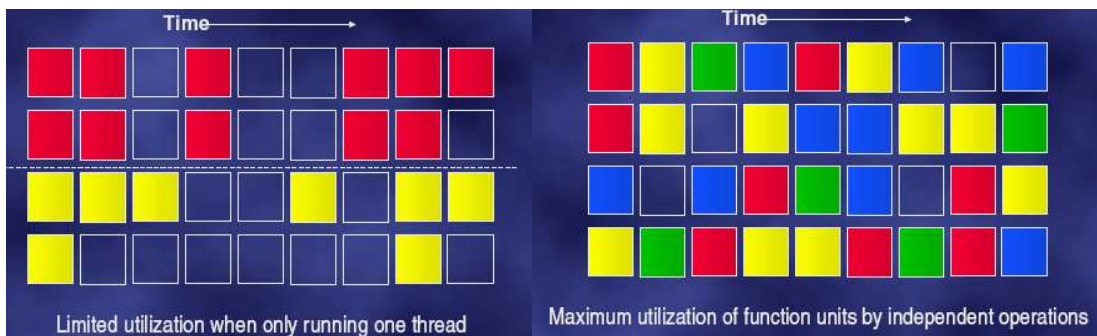


Figure 2. Comparison of chip multiprocessing and SMT

Figure 2 illustrates the comparison between chip multiprocessing and SMT. Even though chip level multi-processors have a similar property of running multiple threads

simultaneously, the diagram on left shows that the issue slots are sliced in two and each sub-pipeline is not fully utilized. The diagram on the right shows that SMT is fully utilizing the pipeline resources.

1.2.3 Comparison of SMT and Symmetric Multiprocessing (SMP)

Symmetric multiprocessing is independent physical processors connected via the processor design interface for inter-processor communication. Thus, the operating system will schedule the processes to run on multiple physical processors. Since most programs are designed for single processors, symmetric multiprocessors do not provide performance gains to such programs. However, SMP is good for programs designed with parallel execution in mind, especially embarrassingly parallel programs.

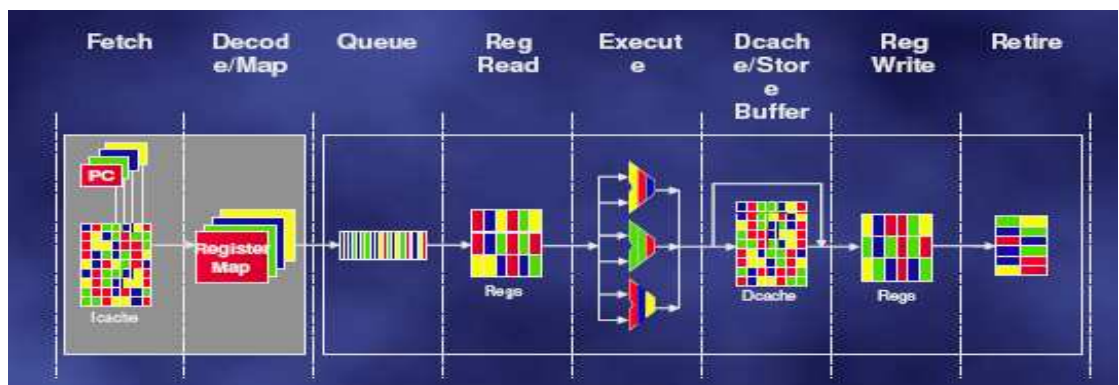


Figure 3. SMT architecture

Figure 3 shows the flow of threads in SMT architecture, where a thread is received from an individual process counter and is mixed into the rest of the stages, which include register read, execute, store buffer, register write and retire stage. [4.] shows an 8-wide superscalar RISC microprocessor with simultaneous multi-threading.

1.3 CPU examples

1.3.1 Intel Hyper-Threading

Intel's Hyper-Threading Technology enables two logical processors on a single physical processor by replicating, partitioning, and sharing the resources within the Intel NetBurst micro architecture pipeline.

Internally, the processor replicates the resources to create copies of each resource for the two threads. These resources include, all per-CPU architectural states, instruction pointers, renaming logic and also smaller resources, such as return stack predictor, ITLB, etc. Partitions divide the resources between the executing threads, such as the Re-Order Buffer, Load/Store Buffers and queues, etc. However, execution threads continue to share some CPU resources within the CPU such as the Out-of-Order execution engine and Caches.

Typically, each physical processor has a single architectural state on a single processor core to service threads. With Hyper-Threading/SMT, each physical processor has two architectural states on a single core, making the physical processor appear as two logical processors to service threads. The system BIOS enumerates each architectural state on the physical processor. Since Hyper-Threading-aware operating systems take advantage of logical processors, those operating systems have twice as many resources to service threads.

1.3.2 IBM Power 5 SMT

IBM's latest Power 5 ensures smooth operation of SMT. The register-renaming resources and associativities of instruction cache and data cache are increased. The branch information queue is split. The load recorder queue is split and store recorder

queue is split. There is out-of-order execution; however, it is more difficult to switch threads on a cache miss because at the time of the miss some earlier instructions may not have been performed while some later instructions may already have been completed. The problem becomes where and how to stop one thread, leaving the thread and its resources in a state that will allow it to be restarted after the switch is made to the other thread. Because there is no thread-switch overhead, SMT can hide even short-duration stalls in the execution pipeline. If, due to pipeline latency, an instruction from one thread is delayed waiting for a result, or if, because of the misprediction of a branch, a portion of a thread's instructions have been flushed from the execution pipeline, instruction from the other thread can continue to be executed. [5.][7.][10.]

1.4 Virtualization

Virtualization is the process of presenting a logical grouping or subset of computing resources so that they can be accessed in ways that give benefits over the original configuration. For example, the abstracting of computing hardware resources to provide enhanced utility of the physical system is a goal of virtualization.

There are several advantages to using virtualization. First, lower hardware and management costs can be recognized, since the physical quantity of hardware required is reduced, the costs associated with maintaining the hardware diminished. Second, virtualization is a good way to sandbox an application's use of physical resources. Third, legacy systems can be preserved on virtual machine operating systems and in a fully operational state. Forth, the virtualization engine is capable of presenting new hardware to the virtual operating system even when the hardware is not physically present. Fifth, there can be known-good, hot-standby virtual machines of different operating systems.

Sixth, virtualization can provide a powerful platform for debugging, monitoring and testing due to the ease with which one can switch operating systems to meet their needs. Seventh, virtualization enables easier system migration, backup and recovery. Eighth, there can be co-located hosting on the same physical hardware.

The virtualization tools introduced in this study are VMware and Hypervisor. VMware is a product of EMC Corporation and Hypervisor is a product of IBM Corporation.

1.4.1 Introduction to VMware

VMware software runs on Windows and on Linux, and will soon debut on Mac OSX. VMware offers several virtualization products. VMware workstation consists of a virtual machine suite for the Intel x86 architecture, and can be used for setting up multiple x86 computers on top of a single host operating system. VMware server can create, edit, and run virtual machines. It uses a client-server model, allowing remote access to virtual machines. ESX Server 3.x (RHEL3) has a service console and acts as a boot-loader for the vmkernel and provides its own management interfaces, such as a command line interface, webpage multi-user interface and a remote console. It has low overhead and better control and granularity for allocating resources, such as CPU time, disk bandwidth, network bandwidth, and memory utilization to the virtual machines. [26.]



Figure 4. ESX server structure diagram

VMware uses the CPU to run code directly whenever possible. For example running user mode and virtual 8086 mode code on an x86 processor will be executed directly on the physical hardware. When direct execution cannot operate, VMware software re-writes code dynamically. This occurs at the VMware kernel level and with real mode code. VMware puts the translated code into a spare area of memory, typically at the end of the address space, which it can then protect and make invisible using segmentation mechanisms.

1.4.2 VMware with Simultaneous Multi-processors

Over committing physical CPUs is a common and accepted practice when running multiple virtual servers. The advantage of over-committing is to slice the application's performance into smaller pieces, therefore providing good performance-on-demand. This is also a common practice for application service providers and other hosted environments.

1.4.3 Introduction to Hypervisor

Para-virtualization is a virtualization technique that presents a software interface to virtual machines that is similar but not identical to that of the underlying hardware. This requires operating systems to be explicitly ported to run on top of the virtual machine monitor (VMM). [8.][9.]

There are several benefits to para-virtualization. First, start time is reduced. The “virtual reboot” avoids the latencies of hardware re-initialization by the BIOS. Also, a pre-booted and frozen virtual machine image can be shipped to all nodes in a cluster. In other words, changing virtual machines is fast.

1.4.4 Cost of Virtualization versus Performance Benefits

New processors are providing features to improve performance of virtualization. Each operating system in each virtual machine can be tuned solely for the hosted application. [30.]

1.5 *Main Contribution of this Thesis*

The research conducted leads to the following contributions:

First, to understand virtualization and simultaneous multithreading, I have generalized and introduced both technologies and investigated several existing products and solutions, such as VMware, IBM’s Hypervisor, Intel’s Hyper-threading and the technology that makes SMT different from SMP and multi-core processors.

Second, this research demonstrates a viable method to design, setup and benchmark virtualization and SMT, in spite of the increased the complexity and difficulty in formulating a proper comparison. In addition, this study demonstrates how different

levels of load interact with system settings and how important such settings are to achieve maximum efficiency.

Third, this paper examined several different methods of modeling virtualization and modeling SMT. Further, the models this study established and verified can be used for comparing various configurations and also for load balancing between the virtual machines.

2 Experimental Setup

2.1 Introduction

This section describes the current experiment details and methodology. In order to test the SMT processor with a virtualization layer, there are several experiments needed. For instance, we need to compare and define the number of threads used in the benchmarking applications, the maximum performance of an application's throughput and establish a baseline case of system settings. The reasoning behind the selection of these methods is also discussed.

2.2 Problem Formulation and system description

There are several cases we needed to consider. The first case is the basic fast-switching multiple-thread single CPU core with the application whose performance is being analyzed running. The second case is layered on top of the first case, where one virtual machine is running on top of one CPU with the same application running. In contrast to the single CPU cases are the SMT CPU cases where same application is running with and without the virtual machine.

2.2.1 Introduction to the Processor being used

The processor used in the experiment is the Intel LV (low voltage) Xeon processor from the Intel Gallatin processor family. It is based on the Intel® NetBurst™ micro-architecture. [24.][25.]

The Intel Xeon processor includes 512KB (L2) cache and includes the following advanced micro-architecture features: Hyper Threading, Hyper Pipelined Technology (Jackson Technology) Rapid Execution Engine, Advanced Dynamic Execution, Trace Cache, Streaming SIMD (Single Instruction, Multiple Data) Extensions 2, Advanced

Transfer Cache, Enhanced Floating Point and Multimedia Engine. The Xeon processor uses a source-synchronous transfer of address and data to improve performance and enables addressing at 2x the system bus frequency and data transfer at 4x the system bus frequency. The 400 or 533 MHz system bus is a quad-pumped bus running off a 100 or 133 MHz bus clock, making 3.2 GB/sec or 4.3GB/sec data transfer rates possible. The LV Xeon processor is based on 0.13-micron process technology. The processor contains 12 kmOps instruction cache, 8 kbyte data cache as L1 cache (Harvard architecture) and a 512KB L2 cache. 4 Mbyte L3 cache is provided. The LV Xeon processor is similar to the full power Xeon processor but runs at a reduced voltage and power level. The clock speed of the processor is 3.4 GHz.

When a thread is scheduled and dispatched to a logical processor, LP0, the Hyper-Threading technology utilizes the necessary processor resources to execute the thread. When a second thread is scheduled and dispatched on the second logical processor, LP1, resources are replicated, divided, or shared as necessary in order to execute the second thread. Each processor makes selections at points in the pipeline to control and process the threads. As each thread finishes, the operating system idles the unused processor, freeing resources for the running logical processor.

2.2.2 Introduction to the Operation System being used

Not all operating systems support hyper-threading. Older, previous-generation operating systems, such as SCO UnixWare SVR5 do not recognize processors as hyper-threaded and will only utilize the processor as one single unit. For the 2.4 Linux kernel, the operating system schedules and dispatches threads to each logical processor, just as it would in a dual-processor or multi-processor system. As the system schedules and

introduces threads into the pipeline, resources are utilized as necessary to process two threads.

To test the problem statement, I used the Red Hat Enterprise Linux 4, kernel 2.6.9-42.ELsmp. This operating system has the following advantages [18.][17.][33.]:

- Native Posix Threading Library (NPTL): This Linux 2.6 kernel feature, originally designed and implemented by Red Hat, provides excellent performance for multi-threaded applications (for example, Java applications). It enables multi-threaded applications that previously required the performance offered by proprietary Unix systems to be successfully deployed on Red Hat Enterprise Linux. The implementation provides full POSIX compliance, support for Thread Local Storage and Futex-based synchronization.
- Asynchronous I/O support allows processes to continue running after issuing a disk read/write I/O. Previously, processes were required to wait for their disk I/O requests to complete before they could continue processing. The feature is particularly useful for processes that issue multiple writes in rapid succession, such as database processes. However, asynchronous I/O can be very useful for any multi-user application.
- The $O(1)$ scheduler in the 2.6 Linux kernel provides greatly increased scheduling scalability. This increase has been achieved by a full redesign of the scheduler algorithm in the 2.4 kernel so that the time taken to choose a process for placing into execution is constant, regardless of the number of processes. The new scheduler scales very well, regardless of process count or

processor count, and imposes a low overhead on the system. The algorithm uses two process priority arrays; active and expired. Processes are being scheduled based on their priority and prior blocking rate. When a processes' time-slice expires, the time-slice is placed on the expired array. When all processes in the active array have expired their time-slice, the two arrays are switched, restarting the algorithm. For general interactive processes (as opposed to real-time processes) this results in high-priority processes, which typically have long time-slices, getting more compute time than low-priority processes. However, it does not get to the point where high-priority processes can starve the low-priority processes.

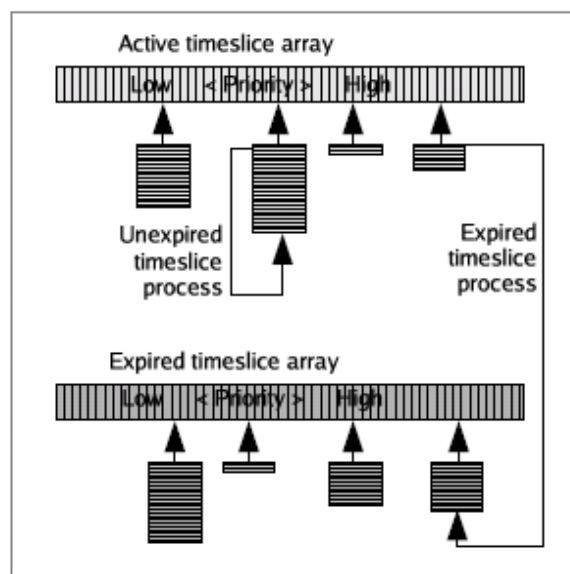


Figure 5. The O(1) scheduler

- This scheduler support for hyper-threaded CPUs was developed by Red Hat. Hyper-threading support ensures that the scheduler can distinguish between physical CPUs and logical (hyper-threaded) CPUs. As shown in Figure.7, the

scheduler compute queues are implemented for each physical CPU, rather than each logical CPU, as was the case previously. This results in processes being evenly spread across physical CPUs, thereby maximizing utilization of resources, such as CPU caches and instruction buffers.

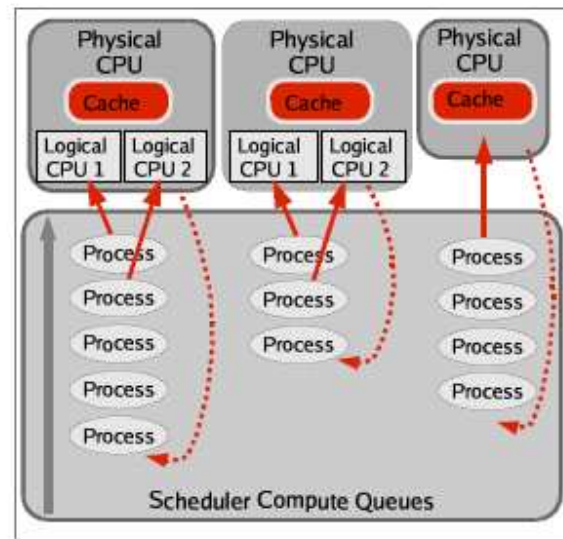


Figure 6. Scheduler also supports hyper-threading

2.2.3 Introduction to the virtual machine being used

The virtual machine used in the experiment is VMware workstation 5.5. [21.][22.][23.] VMware workstation works by creating fully isolated, secure virtual machines that encapsulate an operating system and its applications. The VMware virtualization layer maps the physical hardware resources to the virtual machine's resources, so each virtual machine has its own CPU, memory, disks, and I/O devices, and is the full equivalent of a standard x86 computer.

VMware 5.5 also supports 32-bit and 64-bit host and guest operating systems running on multiprocessor host machines. Further, it provides experimental support for two-way Virtual SMP. This includes any SMP hardware, including dual-core systems

and hyper-threaded uniprocessor systems. The number of processors can be configured in the VMware application settings. Two virtual processors can be assigned in both 32-bit and 64-bit guests.

2.2.4 Introduction to the Load and Benchmark

Load 1:

Oracle database server is installed on the host computer and used for benchmarking. The Oracle application also runs on the host computer and calls the server with multiple queries.

The application being used is Swingbench. The software enables a load to be generated and the transactions, or response times to be recorded and charted. While it is primarily used to demonstrate Real Application Clusters it can also be used to demonstrate functionality such as online table rebuilds, standby databases, online backup and recovery etc. The benchmark run on Swingbench is OrderEntry. OrderEntry is based on the "oe" schema that ships with Oracle9i/Oracle10g and can be run continuously. It introduces heavy contention on a small number of tables and is designed to stress interconnects and memory.

2.2.5 Rationality behind the choice of benchmark and load

Why use 'New customer registration'?

Swingbench, the load generator, is used to stress the database. The application has a java-based GUI. While running, this application will spawn threads as individual clients. There are five benchmarks to be chosen, new customer registration, browse products,

order products, process products and browse orders. New customer registration will be used as the standard of the benchmark. This process will include running of the java application with multiple threads and multiple Oracle connections, and each of them will connect to each client and data will be inserted into the database with commits. As the Oracle instance writes to the database, we can monitor the disk I/O and the impact with hyper-threading (HT) and virtualization. Figure 7. displays the Swingbench GUI used to generate load.

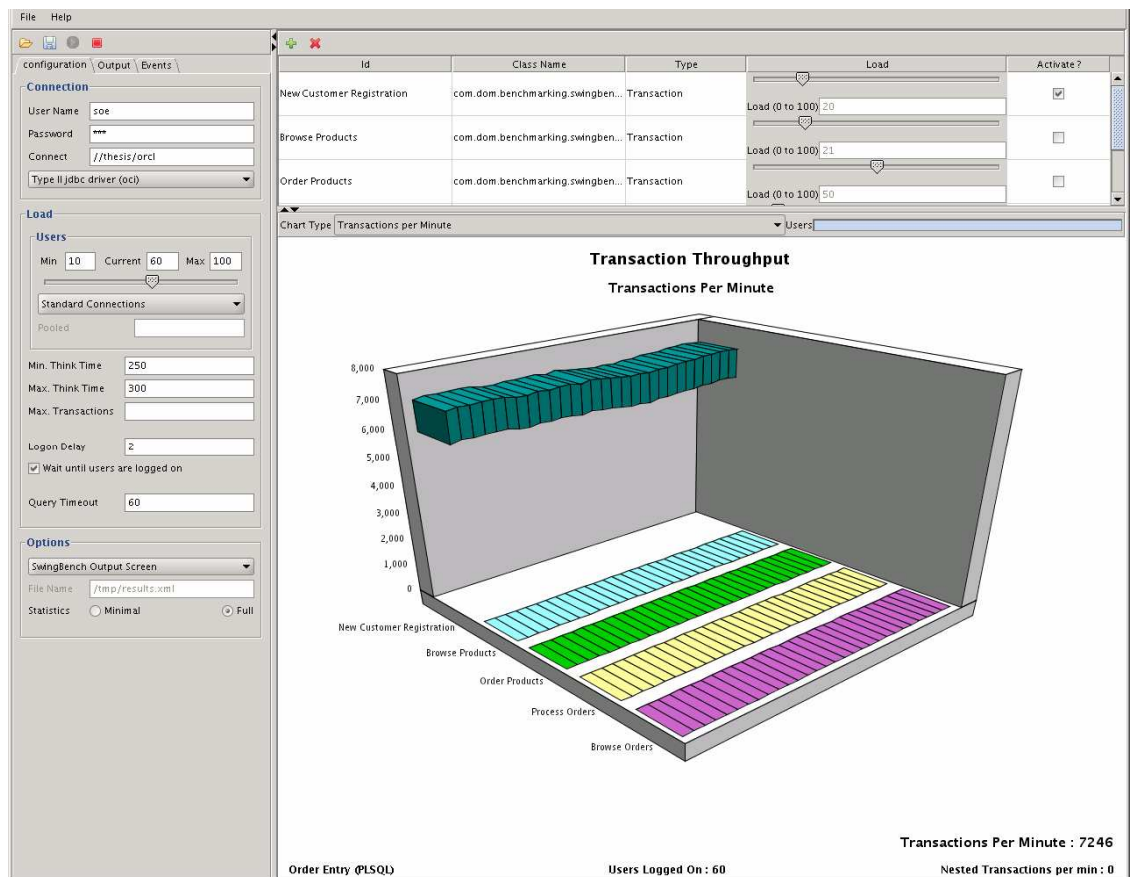


Figure 7. Swing Bench

Why use Oracle?

Oracle database architecture is well established and performance orientated. The Oracle instance can be considered as two major parts, first, the physical processes (threads), second, the memory structure. [36.][11.]

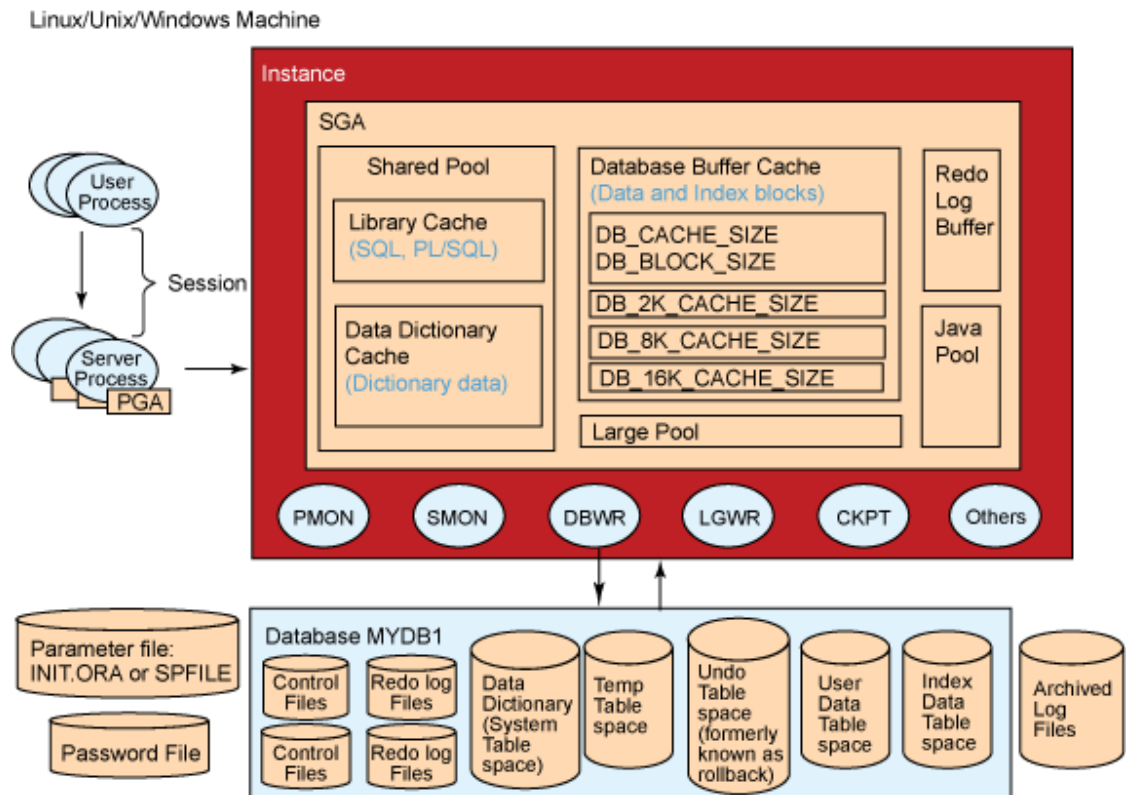


Figure 8. Oracle Architecture Overview

Figure 8, the Oracle memory architecture is briefly illustrated as the following,

- System global area (SGA). This is a large chunk of memory that maintains many internal data structures that all processes need to access. It caches data from the hard disk; buffer data before writing it to the hard disk and holding parsed SQL, etc.
- Process global area (PGA). This is process-specific memory, allocated via C runtime, eg, malloc or mmap. It grows or shrinks at runtime.

- User global area (UGA), the memory that each session can access. If dedicated server, UGA is synonymous with PGA. Otherwise, if share server process, UGA must be started in a memory structure that every shared process has access to (synonymous with SGA).

The Oracle physical process operation is illustrated as follows:

The Oracle server processes perform work on behalf of a client session. We use a dedicated server for these tests so that there is one-on-one mapping between client connection and server process (thread).

Background processes perform mundane maintenance tasks needed to keep the database running, such as maintaining block buffer cache, writing blocks out to data files as needed, or monitoring archive log files. Another advantage of using Oracle is the ease of use of Oracle web console, which gives a convenient interface for performance monitoring. This topic will be discussed in the Chapter 3.

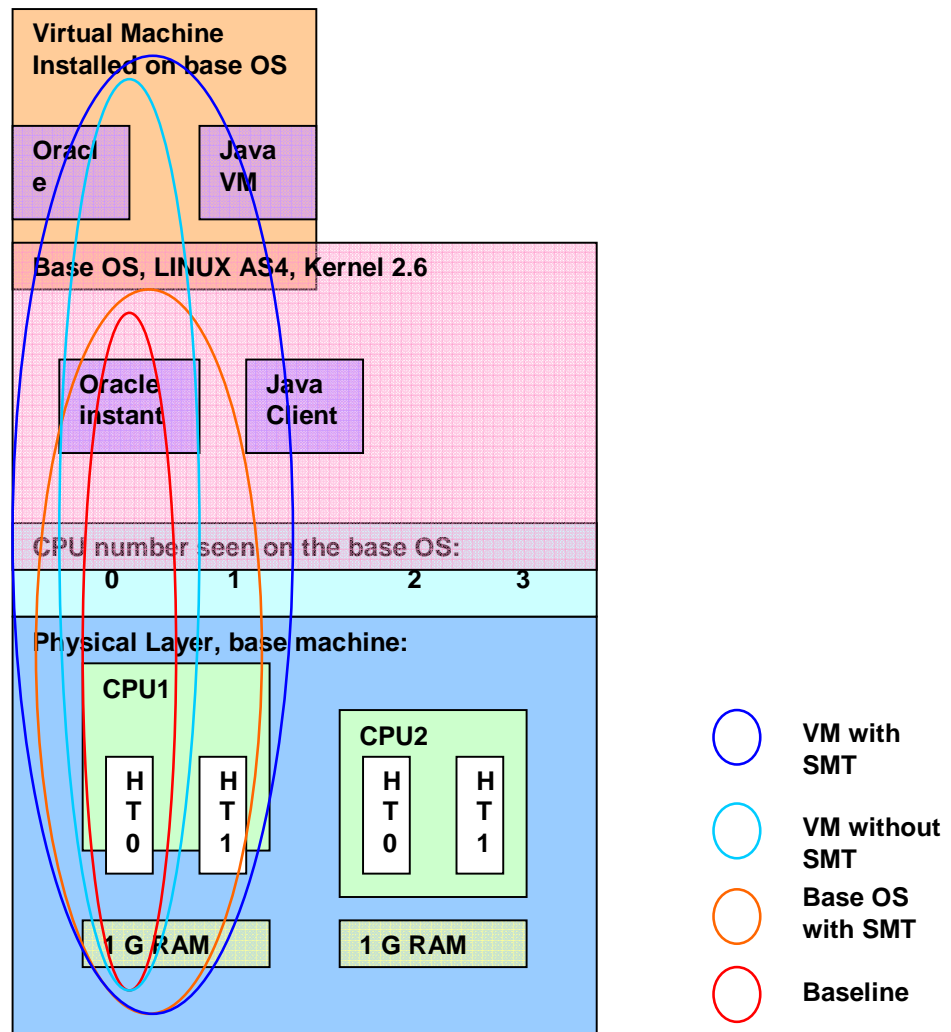


Figure 9. Illustration of test system setting

2.2.6 Baseline: base operating system without virtual machine without SMT

The baseline test is used as the metric for comparison. Figure 9 (circle in Red) demonstrates the setting which consists of the base operating system and the physical layer. The system setting for the baseline test is a Linux machine running on

- 1 physical CPU
- 1 hyper thread (HT) of the 2 HT of 1 physical CPU

- SMT mode of the base operating system is disabled
- 1 GB of RAM
- OS version: RHEL4
- Oracle server and Swingbench test tool

The GRUB configuration is as the following:

```
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol00
    rhgb quiet mem=1G maxcpus=1 noht
    initrd /initrd-2.6.9-42.ELsmp.img
```

2.2.7 Base operating system without virtual machine with SMT

Figure 9 (circle in Orange) demonstrates the setting which consists of the base operating system and the physical layer. The system setting is the following

- 1 physical CPU
- 2 HT of 1 physical CPU
- SMT mode of the base operating system is enabled
- 1 GB of RAM
- OS version: RHEL4
- Oracle server and Swingbench test tool

The following GRUB configuration is used,

```
title RHEL AS4 1GB HT (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol00
    rhgb quiet mem=1G maxcpus=2
    initrd /initrd-2.6.9-42.ELsmp.img
```

2.2.8 One virtual machine on base operating system without SMT

All physical CPUs are used on the base operating system, each physical CPU has SMT mode enabled. One virtual machine installed. Even though the base operating system is operating at full capacity, the virtual server is set to run with 1 logical thread. To do this, VMware will see 2 virtual CPUs (2 hyper thread = 1 physical CPU), but only one will be selected for use in the virtual operating system via GRUB.

- 2 physical CPUs
- SMT on Base OS is enabled and running on all 4 HTs (2 physical CPUs)
- 2 GB RAM is used on the base OS
- VMware ports 1 physical CPU to the client OS
- Client OS sees 2 HT of 1 physical CPU and uses 1 HT
- 1 G of RAM is allocated to the client OS
- OS version: RHEL4 on both host and client OSs
- Oracle server and Swingbench test tool is running on client OS
- Oracle server and Swingbench test tool is **not** running on base OS

The following GRUB setting is for the base OS,

```
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
    rhgb quiet
    initrd /initrd-2.6.9-42.ELsmp.img
```

The following GRUB setting is for the client OS,

```
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
    rhgb quiet mem=1G maxcpus=1
```

```
initrd /initrd-2.6.9-42.ELsmp.img
```

2.2.9 One virtual machine on base operating system with SMT

One virtual machine is running with 2 virtual CPUs. In this virtual machine operating system, the same Oracle server is installed and runs with the same benchmarks.

In this test, the virtual machine will be able to run on all the CPUs it sees.

- 2 physical CPUs
- SMT on Base OS is on and running on all 4 HTs (2 physical CPUs)
- 2 GB RAM is used on the base OS
- VMware ports 1 physical CPU to the client OS
- Client OS sees 2 HT of 1 physical CPU and uses 2 HT
- 1 GB of RAM is allocated to the client OS
- OS version: RHEL4 on both host and client OSs
- Oracle server and Swingbench test tool is running on client OS
- Oracle server and Swingbench test tool is **not** running on base OS

The following GRUB setting is for the base OS,

```
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
    rhgb quiet
    initrd /initrd-2.6.9-42.ELsmp.img
```

The following GRUB setting is for the client OS,

```
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
    rhgb quiet mem=1G maxcpus=2
    initrd /initrd-2.6.9-42.ELsmp.img
```


2.3 **System Setup Procedure**

The chapter describes the setup procedure for all the tests.

2.3.1 Setup Oracle and Swingbench

First, given the test machine, see 2.1 for CPU/system information. On the base machine, install the operating system, install Oracle 10g, and database administration group accounts, install j2sdk, install VMware-workstation RPM and set the recommended Oracle 10G kernel parameters. The Oracle and Swingbench parameter will be the same for all the test cases:

```
fs.file-max = 65536
kernel.shmmax = 2147483648
kernel.sem = 250 32000 100 128
net.ipv4.ip_local_port_range = 1024 65000
net.core.rmem_default = 262144
net.core.rmem_max = 262144
net.core.wmem_default = 262144
net.core.wmem_max = 262144
```

For instance, adding RPMs for Oracle DB (gcc-3.4.6-3.i386, glibc-devel-2.3.4-2.25.i386.rpm, glibc-headers-2.3.4-2.25.i386.rpm, glibc-kernheaders-2.4.9.1.98.EL.i386.rpm)

Add more swap space per minimum Oracle recommendation:

```
dd if=/dev/zero of=/swapfile bs=1024 count=1572864
ls -al /swapfile
mkswap /swapfile
swapon /swapfile
free
```

Extract OracleDB-10G release 2, put Oracle environment variable in place, start listener and then start Oracle. Setup Swingbench environment variable to install Swingbench Order Entry by using oewizard. Java virtual machine will be allocated with 250MB of memory.

2.3.2 Tuning SMT

First, with the current chipset, SMT can be turned off or on without the presence of the operating system, by setting the Advance BIOS setting: Hyper-thread enabled. Depending on the version of kernel, there maybe other parameters needed to be set in the BIOS. For example, kernel 2.4 requires ACPI to be enabled in the BIOS in order to use the logical CPUs. After hyper-threading is turned on, the base OS will be able to see 4 logical CPUs.

Each physical CPU has 2 hyper-threads. If the base OS has all 4 logical CPUs running, VMware will be able to use 1 physical CPU, or 2 hyper-threads. This is due to the fact that VMware workstation can use only 1 physical CPU. This is a limitation of the version of VMware selected. However, all test case configurations fit within these limitations of VMware workstation.

The base OS has 3 settings according to test cases in section 2.2. Setting 0 is used for test cases 3 and 4 where the client OS is running and there is no limitation on number of CPUs and memory allocated. Setting 2 is used for base case, where kernel parameter “noht” is used to ensure hyper-threading is disabled. Setting 1 is used for case 2, “maxcpu=2” means 2 logical CPUs are used. The following is the GRUB file:

```
### 0 ### Original SMP
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
rhgb quiet
    initrd /initrd-2.6.9-42.ELsmp.img

### 1 ### 1GB RAM w/ 1 cpu w/ HT
title RHEL AS4 1GB HT (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
rhgb quiet mem=1G maxcpus=2
    initrd /initrd-2.6.9-42.ELsmp.img
```

```

### 2 ### 1GB RAM w/ 1 cpu w/o HT
title Red Hat Enterprise Linux AS (2.6.9-42.ELsmp)
    root (hd0,0)
    kernel /vmlinuz-2.6.9-42.ELsmp ro root=/dev/VolGroup00/LogVol100
rhgb quiet mem=1G maxcpus=1 noht
    initrd /initrd-2.6.9-42.ELsmp.img

```

To ensure the correct setting, the number of CPUs in use can be seen in `/proc/cpuinfo`. There should be 4 identical CPUs (processor 0 to 3) shown as the following:

```

processor       : 0
vendor_id      : GenuineIntel
cpu family     : 15
model          : 2
model name     : Intel(R) Xeon(TM) CPU 2.00GHz
stepping       : 9
cpu MHz        : 1996.852
cache size     : 512 KB
physical id    : 0
siblings       : 2
core id        : 0
cpu cores      : 1
fdiv_bug       : no
hlt_bug        : no
f00f_bug       : no
coma_bug       : no
fpu            : yes
fpu_exception  : yes
cpuid level    : 2
wp             : yes
flags          : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca
cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe cid xtpr
bogomips       : 3996.67

```

2.4 Conclusions

In this chapter, I have discussed the system setup and the experimental design to provide a clear picture of the operating system configuration with and without VMware and with and without SMT. The setup provides a clear apples-to-apples comparison [34.]. When doing performance comparisons it is important to make sure that the configuration of the systems being compared are as similar as possible. While comparing the performance of client versus host machine, each test case should use the same setup and

the impact of host operating system should be kept to a minimum. For instance, when the virtual machine is configured to use of 1GB of memory and 1 HT, these limits should be only seen on the virtual machine. On the host machine, the base operating system must contain more than 1GB of memory and 1 HT. In this way, we ensure that the base OS has its own resources to run the virtual machine with enough spare capacity to cover the overhead of running the virtual machine without influencing the virtual machine's performance.

3 Performance Metric and Selection

This chapter analyzes the results collected and developed from the system performance model for virtualization with simultaneous multithreading.

3.1 Results collected

This section illustrates which data is used and the data's corresponding weight, or importance.

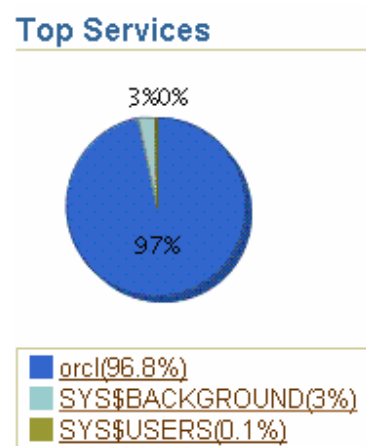


Figure 10. Oracle System Monitor-top

Top Modules (by Service)

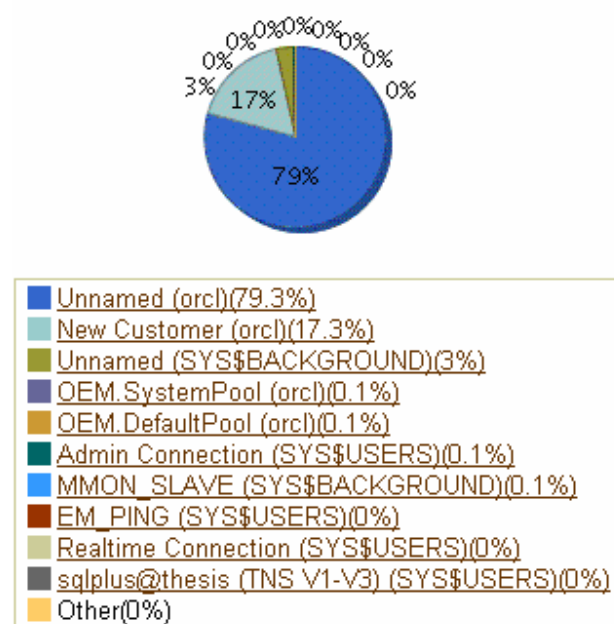


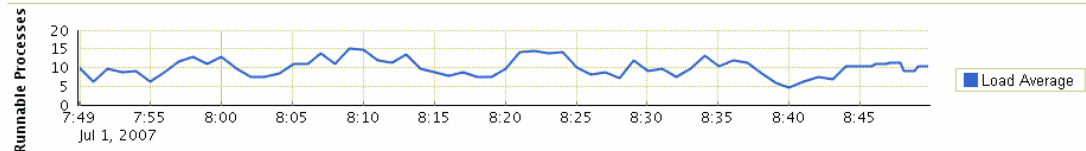
Figure 11. Oracle Monitoring—top modules

Database Instance: orcl

[Home](#) [Performance](#) [Administration](#) [Maintenance](#)

Best viewed using latest SVG plugin

Host



Average Active Sessions

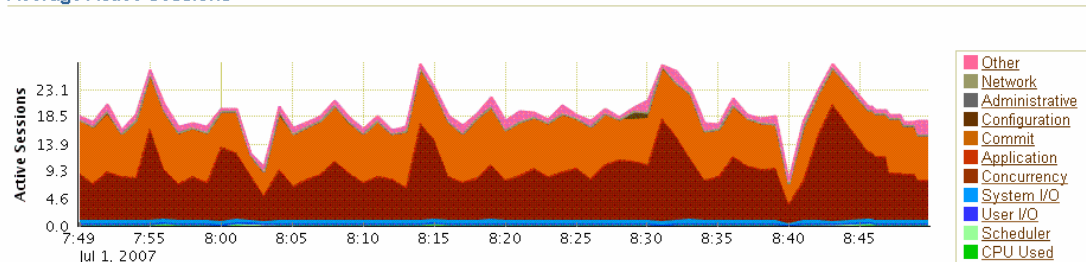


Figure 12. Oracle Console Data Collection

Figures 10 through 12 show that the Oracle console collects data for Oracle performance and the SQL query information. The SQL trace file contains the number of SQL statements executed and the corresponding statistics. It is a good Oracle tracing tool

to discover abnormal SQL statements which take up large amount system resources and result in an abnormal statistical distribution.

From the data collected, including those from vmstat, top and the Oracle console, several obvious patterns emerge as parameter changes are tested. The parameters that were changed include enabling and disabling SMT mode and using or not using VMware. Results for the wait time, as expected, changed when SMT was turned off. Wait time includes in memory undo latch, cache buffer chains and library cache pin.

The tools used and metrics collected included the following:

- vmstat, for cpu usage as seen in the current operating system.
- topstat, to see the top cpu consuming processes
- awr report (auto workload repository), from Oracle, to see the Oracle performance, and throughput (Oracle transactions per second)
- Transactions per minute, same as seen in the Swingbench GUI. (This transaction is different from an Oracle transaction in which each transaction is a complete process consisting of multiple Oracle transactions)
- Swingbench log file (result.xml), mainly for response time distribution.

```

root@thesisVM:~
top - 20:35:29 up 56 min,  2 users,  load average: 9.48, 11.10,  6.71
Tasks: 126 total,  4 running, 122 sleeping,  0 stopped,  0 zombie
Cpu0  : 34.9% us, 31.8% sy,  0.0% ni, 32.7% id,  0.6% wa,  0.0% hi,  0.0% si
Cpu1  : 34.4% us, 32.2% sy,  0.0% ni, 31.9% id,  0.9% wa,  0.6% hi,  0.0% si
Mem:   1034604k total,  914808k used,  119796k free,  17220k buffers
Swap:  3604464k total,    0k used,  3604464k free,  652020k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 3855 root        17   0   643m  32m 8464 S   13   3.2   1:38.37 java
 3722 oracle     16   0   380m  18m 17m R    4   1.9   0:39.80 oracle
 3927 oracle     15   0   365m  65m  63m S    3   6.5   0:14.85 oracle
 3971 oracle     15   0   365m  61m  59m S    3   6.0   0:14.82 oracle
 3997 oracle     15   0   365m  62m  60m S    3   6.2   0:14.91 oracle
 3931 oracle     15   0   365m  68m  66m S    2   6.7   0:14.91 oracle

root@thesis:~
top - 21:05:07 up 2 days,  6:23,  5 users,  load average: 1.62, 1.52,  1.09
Tasks:  74 total,  2 running,  72 sleeping,  0 stopped,  0 zombie
Cpu0  :  2.0% us, 72.4% sy,  0.0% ni, 25.6% id,  0.0% wa,  0.0% hi,  0.0% si
Cpu1  :  0.7% us,  4.0% sy,  0.0% ni, 95.3% id,  0.0% wa,  0.0% hi,  0.0% si
Cpu2  :  2.3% us, 71.2% sy,  0.0% ni, 26.4% id,  0.0% wa,  0.0% hi,  0.0% si
Cpu3  :  0.3% us,  0.3% sy,  0.0% ni, 99.3% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:   2074420k total, 2058036k used,  16384k free,  16620k buffers
Swap:  3604464k total,  144k used,  3604320k free, 1821580k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 5996 root        5 -10 1105m 920m 904m S  147  45.4 37:47.93 vmware-vmx
 6006 root        0 -20    0    0    0 S    5   0.0  2:11.67 vmware-rtc
 5857 root       15   0 42832  28m  15m S    1   1.4   0:48.29 vmware
    1 root       16   0  2064   548  472 S    0   0.0   0:01.10 init
    2 root      RT    0    0    0    0 S    0   0.0   0:00.01 migration/0
    3 root      34  19    0    0    0 S    0   0.0   0:00.00 ksoftirqd/0
    4 root      RT    0    0    0    0 S    0   0.0   0:00.01 migration/1

```

Table I. Top monitoring sessions and host machine monitoring session
Example results as mean of samples data points:

Insert new customer						
No SMT no VM (baseline) w SMT no VM no SMT VM SMT	VMware percentage(Show in Top of Base operating system)	VMware memory	VMware cpu used(%)	Base operating system cpu used (%)	Oracle transaction per second	Swingbench transaction per minute
	None	None	None	85	241	7340
	None	None	None	64	334	10120
	89	35	76	22	190	5895
	162	54	76	32	254	7620
	Number of clients	response time(ms)	Min response time(ms)	max response time(ms)		

No SMT				
no VM	60	199	5	9745
w SMT				
no VM	60	55	4	3077
VM				
NO SMT	60	303	6	6361
VM SMT	60	156	4	5583

Table II. Sample result for database “insert” benchmark

Test Case	1	1	1	2	2	2
Threads	20	40	60	20	40	60
Ave Response	9	91	199	6	59	55
Min Response	4	4	5	4	4	4
Max Response	2461	1534	9745	860	450	3077
awrr Transactions	139.63	208.54	241	156	278	334
FreeMemAvg	279267	15987.1	19138	21940.3	65611.1	17626.3
% Free Mem	0.000279	1.6E-05	1.91E-05	2.19E-05	6.56E-05	1.76E-05
CacheAvg	591555	756614	505325	807908	728615	649602
IObi	74.3267	1.20667	207.917	1.51333	27.88	151.412
IObo	480.153	429.553	674.167	627.513	993.313	897.766
CPUIdleAvg	63.52	45.08	15.5667	80.7333	58.92	42.2092
CPUUsedAvg	36.48	54.92	84.4333	19.2667	41.08	57.7908
CPU Used/Trans	0.261262	0.263355	0.350346	0.123504	0.14777	0.173026
Test Case	3	3	3	4	4	4
Threads	20	40	60	20	40	60
Ave Response	34	187	303	29	82	156
Min Response	6	6	6	1	5	4
Max Response	2729	3077	6361	1663	4801	5583
awrr Transactions	117.84	158	190	120.16	202.82	254
FreeMemAvg	308967	50804.1	17170.3	105088	129204	15901.5
% Free Mem	0.00030897	5.08E-05	1.72E-05	0.000105	0.000129	1.59E-05
CacheAvg	552590	758786	493096	758350	657485	6.71E+05
IObi	63.0067	1.94	32.125	1.18	42.6611	9.29
IObo	337.74	309.84	337.489	433.12	511.317	572.543
CPUIdleAvg	47.54	35.4533	24.8306	57.3267	37.1056	22.6533

CPUUsedAvg	52.46	64.5467	75.1694	42.6733	62.8944	77.3467
CPU						
Used/Trans	0.4451799	0.408523	0.395628	0.355137	0.3101	0.304515
Host View:						
FreeMemAvg	1.69E+05	317490	324913	53203.4	16353.4	16925.6
CacheAvg	1.76E+06	1.54E+06	1.57E+06	1.78E+06	1.86E+06	1.81E+06
IObi	57.74	77.5389	59.6322	0.78	1.65333	3.32993
IObo	533.567	759.933	425.753	721.653	393.587	777.17
CPUIdleAvg	85.2467	65.6889	79.4368	75.2933	82.2133	59.7619

Table III. Details of all results collected





Test Case Representation and Line Color		
	Without SMT	With SMT
Without VM	1 	2 
With VM	3 	4 
Data Collected		
Data Collected and used for calculation in 3.2		
Data Collected in the host OS and is not supposed to be used for comparison		

Table IV. Legend for Table II,III and Color Scheme Used in the Graphs

The data points collected are parsed result from each test. The duration of each test is between one half to one hour. Result collection started after the load was stabilized, approximately fifteen minutes after the test was started. Sampling time interval is 10 seconds and for 1 hour in duration. Thus, there was 360 data points, or samples. The mean of the samples was used. Thus, all data in Table 2 is the mean of the all the data points collected.

3.2 Problem Formulation

3.2.1 Introduction to problem formulation

From Table 1 sample results collected, we can simply derive the following,

Impact on throughput for the disk I/O benchmark, insert new customer to the database:

The SMT factor is about 1.35, which is 35% increase of throughput

$$\frac{\text{BaseOS with SMT}}{\text{Base OS without SMT}} = \frac{334}{254} = 1.31$$

$$\frac{\text{Virtualization with SMT}}{\text{Virtualization without SMT}} = \frac{254}{190} = 1.34$$

The above data show consistency. For instance, with SMT there is a general increase of performance of 30% in database process, even with the virtual machine abstraction. It is consistent with our expectation and with result from other benchmarking done with SMT [6.][35.].

The virtualization factor is about 0.75, which is 25% degradation of throughput

$$\frac{\text{Virtualization with SMT}}{\text{Base OS with SMT}} = \frac{254}{334} = 0.76$$

$$\frac{\text{Virtualization without SMT}}{\text{Base OS without SMT}} = \frac{190}{254} = 0.74$$

The performance degradation of VMware (- 25%) shows consistency. Virtualization with SMT shows a decrease in performance when compared with the base operating system with SMT enabled. The same is true for virtualization without SMT; it is not only less but the percentage decreases is identical to the previous case, when comparing the results with the base operating system without SMT.

Thus, we can generally agree they are independent factors, such that SMT did not change the behavior of Virtual machine degradation and vice versa.

3.2.2 Simple Result Analysis

This section, we focus on the comparison of different values using graphic representation.

3.2.2.1 CPU time used per transaction

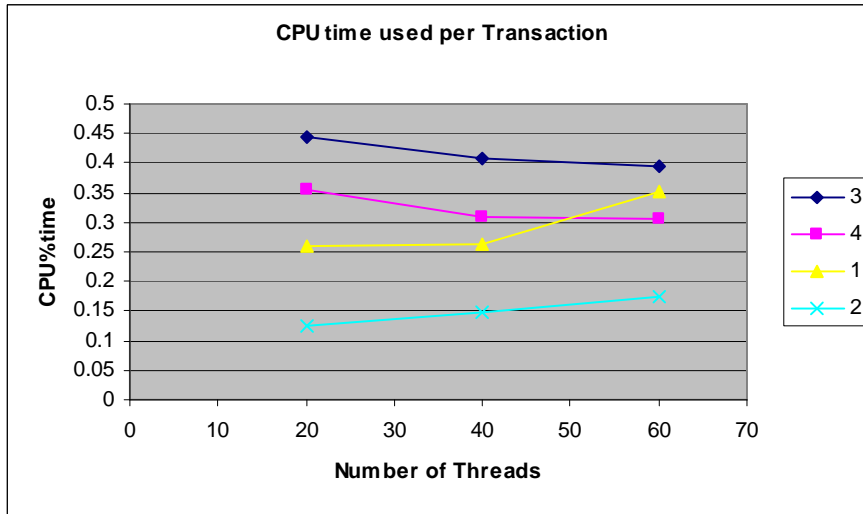


Figure 13. CPU time per each Transaction

The CPU time per each transaction is compared between the four cases, we can interpret the relationship between them as the following: cases without VM, the more the number of threads and the more expensive the average CPU time spent on each transactions. This is seen as line 1 and 2 are going upwards, and line 3 and 4 are decreasing with response to number of threads. The effect of SMT is obvious as line 2 and line 4 are always lower than line 3 and line 1. Notice the interception of line 4 and line 1 indicates that more the number the threads, the more efficient with VM with SMT such that VM with SMT out perform the original case. This also implies more utilized the system, the more efficient it is.

3.2.2.2 Response Time

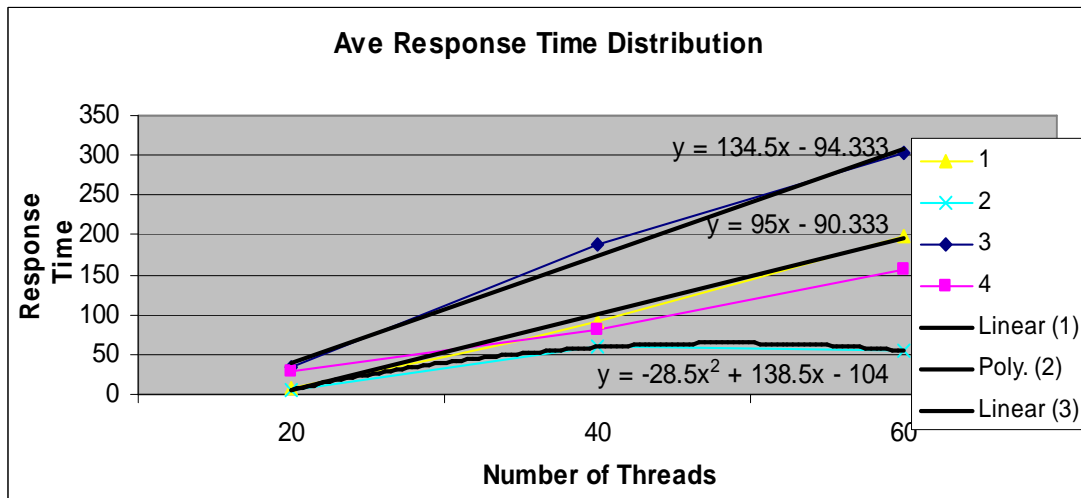


Figure 14. Average Response Time

The response time distribution shows no abnormality. In the graph trend lines are also applied to see the function trend of each case. Case 1, 2 and 3 are all close to linear except case 2 (SMT on original machine). The reason might be due to the response time being close to minimal all the time. As expected, cases with SMT (2, 4) always have lower response times than their counterparts. Interestingly, similar to CPU time used per transaction, response time is better when a high number of threads are used. Case 4 (SMT with VM), the VM impact (longer response time) was bigger with a smaller number of threads; however, with a larger number of threads, the advantage of SMT is shown as the response of case 4 is lower than case 1.

3.2.2.3 Throughput

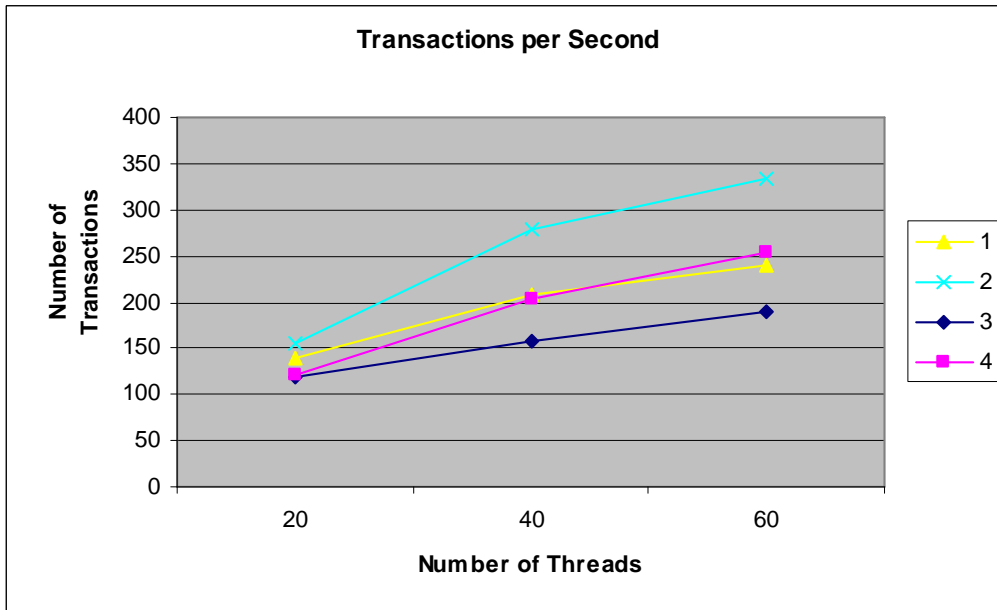


Figure 15. Throughput (Oracle Transactions Per Second)

Graph of transactions per second shows the actual work the process has done. The number of transactions increases with the number of threads. From 20 to 40 thread, all cases have steeper slope than from 40 to 60 which shows, the increase in throughput is faster at lower number of threads and higher number of threads tend to saturate the throughput increase rates. From 40 to 60 threads, case 2 and 4 (with SMT) have same slope and case 1 and 3 (with out SMT) have the same slope. Cases with SMT show a better increase in throughput with large number of threads. Also, this shows the difference between VM stay constant (the height difference between the lines). With the impact of VM, case 3 and 4 has lesser throughput than case 1 and 2. Start with low number of threads, the transactions are close with all cases. However, at 60 threads, we see that even VM decrease the throughput, positive impact of SMT let throughput per threads increase and over throughput of Case4 (with SMT with VM) outperform Case1 (original case with out SMT without VM. More important, due to the steeper slope of

SMT case, the more the number thread tend large the difference of throughput with their counterparts. Notice, we see the pattern of case2, 4, 1, 3 (ordering from best to worse) at high number of threads as we seen in the previous two graphs.

3.2.2.4 Memory

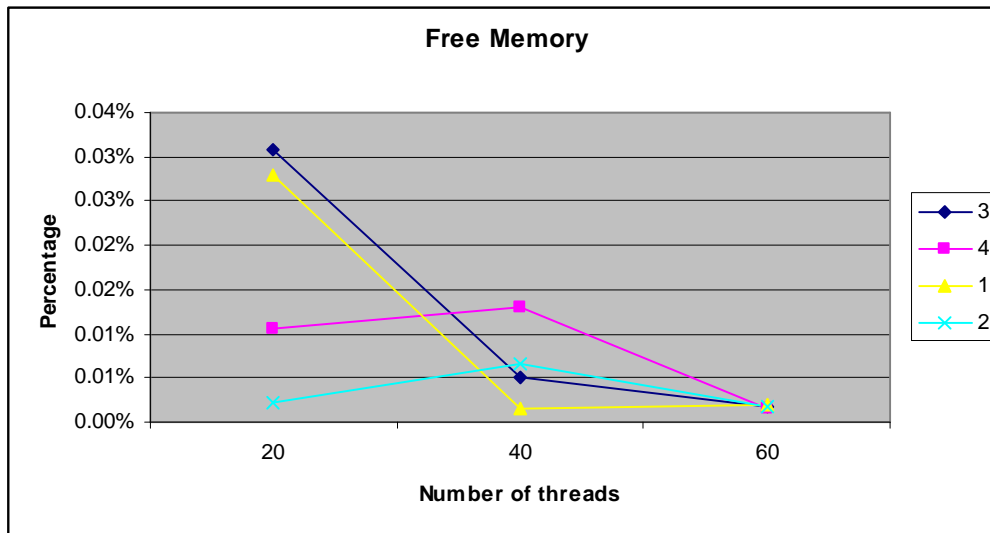


Figure 16. Memory Free in %

The memory usage of the machine shows a better statistic with less number of threads. With 20 threads, there are more obvious difference between the cases, with 60 threads, there are more memory consumed that all cases merge to high memory usage that we cannot get much out from that situation. From observation of 20 thread cases, we see the order 3,1,4,2 (best to worst), interesting, this is exactly the reverse as previous cases as we saw case 2,4,1,3 (best to worst). This might implies that the more the system is utilized, the better the efficiency we can get out from it. Let us look further at 40 and 60 threads cases. There are drastic changes in the memory used. Cases with VM (case 4,2) shows low memory usage increase (even decrease at 40 threads); while, cases without VM has severe increase in memory used and more than their no VM counterparts. At higher number of threads, we see the order as 4,2,3,1(best to worst)

Therefore, we again see the high the usage the better result we can get from virtualization with SMT.

3.2.2.5 Input and output

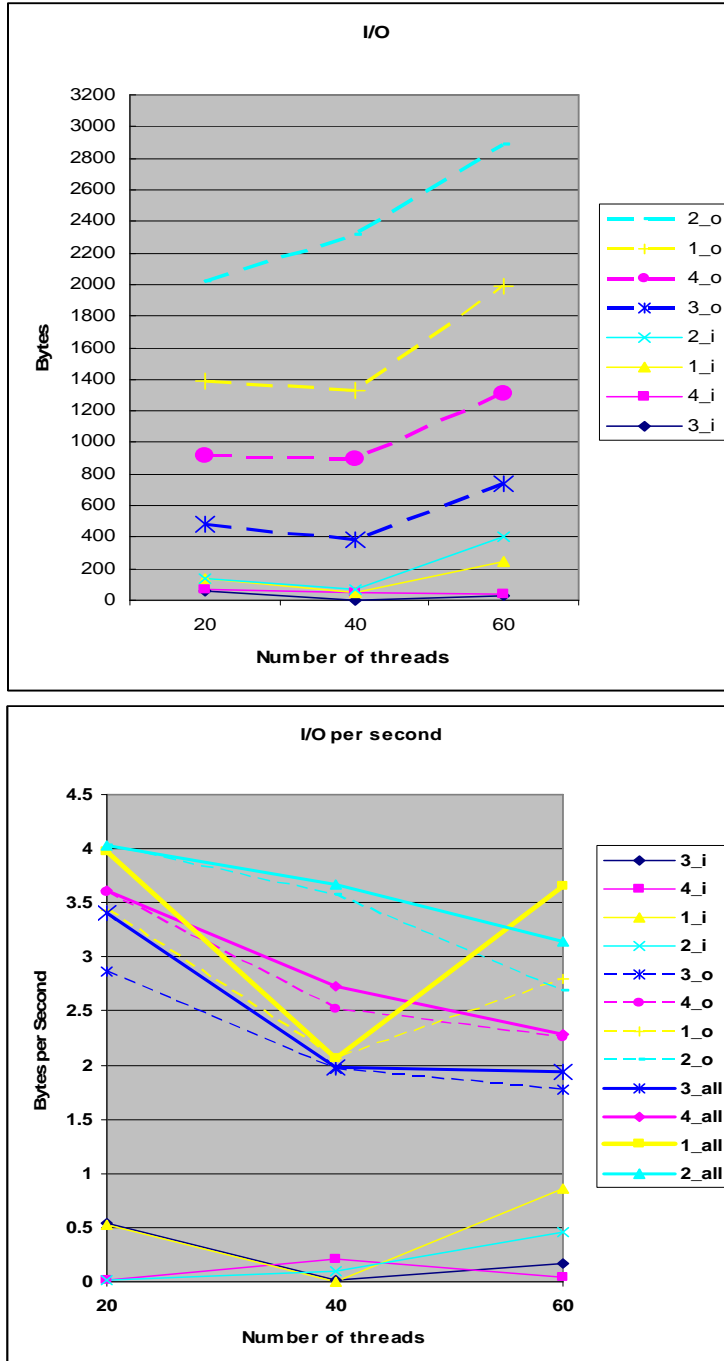


Figure 17. IO Statistics

IO statistics is the bottleneck of VM, we can see from the first IO statistic graph that cases without VM are better performed than cases with VM. The advantage of HT is limited that case 4 with VM with HT is worse than original case. When comparing IO per second, 40 threads cases do not perform as good as 20 and 60 thread cases which they are close in performance.

3.3 Modeling methods used

This section, we want to generalize the results into some models for prediction and further analysis. We first start with looking at available SMT models and VM models separately. [15.][31.][29.][32.]

3.3.1 Previously suggestion methods

There are several models suggested for SMT or VM. The following are two of them.

In micro spectrum, there is a model suggested about SMT in [6.]. The model estimates the overall performance give a description of the processor and the characteristics of the workload. Performance is given by the overall number of instructions executed per cycle (IPC).

$$IPC = \sum_{w=1}^G P_w IPC_w \quad (\text{eqt. 1.})$$

Where P_w is the probability of w ready-to-issue instructions in the global window, IPC_w is the expected IPC for these w instructions, and G is the size of the global window. IPC_w models the effect of structural hazards. The rationale behind this division is that structural hazards are primarily dependent on the hardware architectural

configuration while control and data hazards are primarily dependent on the workload. This model is further derived into model of structural hazards using Markov Chain.

For performance of virtualization, we can use the model suggested in [32.], vconsolidation, a Intel virtualization multiplication value, to calculate the performance of virtualized environment. Weights are the importance distribution of each guest operating system and are defined with fixed number. For instance, more memory and physical CPUs are assigned to guest operating system one than guest operating system two. The workload performance is the performance actually behaved in the guest operating system with the standard workload. The weight is the pre-defined work load of each virtual machine and the *workloadPerf* is the independent performance of each of them. The result is useful for comparing different configuration.

$$\sum_{i=1}^N Weight[i] * WorkloadPerf[i] \quad (eqt. 2.)$$

Thus, this model can predict the how much usage totally used for comparing different guest operating system configuration and the various utilizations. For instance, the first guest operating system can be the web server and the second guest operating system can be a development box.

3.3.2 Modeling for SMT and VM Combined

To show the effect of SMT and VM, we can simply use multivariate linear regression to generate the models. Regression is used in generating functions out from variables. For instance, $Y = Xb$, where Y is the result matrix, X are variables the and b is the coefficients vector. By using the given values of Y and X , we want to find the coefficient vector b , such that $\tilde{Y} = Xb$ and $Y - \tilde{Y} = \varepsilon$, where ε is minimized.

3.3.2.1 Simple Introduction to Regression Analysis

The general form of a simple linear regression is $y_i = \alpha + \beta x_i + \varepsilon_i$ where α is the intercept, β is the slope and ε_i is the error term, which picks up the unpredictable part of the response variable y_i . The error term is usually taken to be normally distributed (Gaussian distribution). The x 's and y 's are the data quantities from the sample or population in question, and α and β are the unknown parameters to be estimated from the data. Estimates for the values of α and β can be derived by the method of ordinary least squares. The method is called "least squares," because estimates of α and β minimize the

sum of squared error estimates for the given data set,
$$\hat{\beta} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2} \text{ and}$$

$\hat{\alpha} = \bar{y} - \hat{\beta}\bar{x}$ where \bar{x} is the mean of the x values and \bar{y} is the mean of the y values.

[16.][19.][20.][14.][27.]

3.3.2.2 Using the Multivariate Regression in the Data Collected

Average response time (Y_1), throughput per minute (Y_2), memory (Y_3) and CPU time used per transaction (Y_4) can be represented as functions of number of threads, presents of virtual machine and presents of SMT. Each Y can be represented via multivariate regression.

$$Y = X b ,$$

$$Y = b_0 + b_1 x_1 + b_2 x_2 + b_3 x_3 + b_4 x_1 x_2 + b_5 x_1 x_3 + b_6 x_2 x_3$$

x_1 = number of threads, x_2 = 0 for no SMT, 1 for SMT, x_3 = 0 for no VM, 1 for VM

$$X = [x_{ones} \quad x_1 \quad x_2 \quad x_3 \quad x_1 x_2 \quad x_1 x_3 \quad x_2 x_3]$$

$$= 1 \quad 20 \quad 0 \quad 1 \quad 0 \quad 20 \quad 0$$

1	40	0	1	0	40	0
1	60	0	1	0	60	0
1	20	0	0	0	0	0
1	40	0	0	0	0	0
1	60	0	0	0	0	0
1	20	1	0	20	0	0
1	40	1	0	40	0	0
1	60	1	0	60	0	0
1	20	1	1	20	20	1
1	40	1	1	40	40	1
1	60	1	1	60	60	1

$$Y_{responseT} = -90.583 + 4.7563x_1 + 81.833x_2 - 3.5x_3 - 3.5x_1x_2 + 1.9625x_1x_3 - 26x_2x_3$$

$$R^2=0.9896$$

$$Y_{throughput} = 91.2825 + 2.6277x_1 - 9.545x_2 - 4.425x_3 + 1.7289x_1x_2 - 0.9171x_1x_3 - 22.5633x_2x_3$$

$$R^2=0.9842$$

$$Y_{memory} = (9.128 + 0.263x_1 - 0.955x_2 - 0.442x_3 + 0.173x_1x_2 - 0.092x_1x_3 - 2.256x_2x_3) * 1E^{-8}$$

$$R^2=0.9766$$

Error! Objects cannot be created from editing field codes.
 $R^2=0.9856$

The square term x_n^2 were not used, as the coefficient of these terms tends to close to zero. Also for x_2 and x_3 , the power will have no effect at all (since their values are 0 or 1). Thus, the close equations with small residual we can get are the above. All of them have 95% confidence interval.

Therefore, we can predict \tilde{Y} by substituting number of threads, presences SMT and VM to found out the best choice. Further, instead of using the above statistics individually, we can balance the response time, throughput, memory and CPU usage as a united performance entity, \tilde{P}_1 , use all case 1 as the base value, for instance,

$$\tilde{P}_1 = \sum_{k=1}^n \left(\frac{w_k}{n} \% \right) = \sum_{k=1}^n (c_k \tilde{Y}_k) = 1,$$

$$c_k = \frac{(\frac{w_k}{n} \%)}{\tilde{Y}_k}$$

k is the each component factor (CPU, throughput, memory, response time), $n=4$ for there is 4 component factors. C_k is the coefficient of each component, such that it is weighted with w_k . For example, there are 4 component factors, $n=4$, weight of each factor is $w_k=1$ and $\tilde{Y}=23$, $23C_k=25\%$, thus, the coefficient $C_k=0.011$. To calculate the improvement and deficiency, we can simply use a new \tilde{P}_i value, and compare it with the base case \tilde{P}_1 to adjust for the best value. New \tilde{Y}_{k_new} will be used and using the C_k we just computed, we can get the new \tilde{P}_i . For example, \tilde{P}_2 is 1.05 and \tilde{P}_1 was 1, there is 5 % performance gain; if \tilde{P}_3 is 0.97, then, \tilde{P}_3 has 5% performance decreases.

Notice: for memory used and response time, C_k should be negative, e.g., $-(C_k)$ to indicate the better the less response time and the better the less memory used.

3.3.3 Load Balancing and Load Distribution

Deriving the above, we can further determine the load distribution of machine with SMT and virtualization. First, we want know about how traffic of load is distributed, for instance, does virtual machines communicate with themselves internally? If so, how resources are distributed among them?

Eq.2 in section 3.3.1 for virtualization together with the model using 3.3.2, we can use $WorkloadPerf[i]=\tilde{P}_i$ for each guest virtual machine. $Weight[i]$ as the resources being divided up to each machine. For the availability and stability of base operating system,

$\sum_{i=1}^N Weight[i] < 0.9$, so that the base operating system has 10% resource available for running the VM manager and such; for a less powerful machine, the base operating system would need more resource relatively. The internal communication between

virtual machine can be managed by virtual network interface vnet1 and vnet8 resides in the host operating system, its operation relies on how many resources are left after assigning them to the virtual machine. Thus we want to take base operating system and the I/O statistics of the virtual machine into account:

$$W = PerfBaseOS + \sum_{i=1}^N Weight[i] * P_i, \text{ where } P_i \text{ is performance of each guest operating}$$

system,

$$\tilde{P}_i = \sum_{k=1}^n (c_k \tilde{Y}_k), \text{ where } \tilde{Y}_1 = \tilde{Y}_{Input/Output}$$

$$PerfBaseOS = ava (1 - \sum_{i=1}^N Weight[i]), \text{ and}$$

To determine *PerfBaseOS*, we can simply use the machine's performance without load as the base, e.g. *ava*=1, then, there are more inter virtual machine traffic, availability, *ava*, will be smaller, say 0.9. Therefore, by comparing values of different *W* (whole system) settings, we can choose the best performance effective model with load balancing among the virtual machines.

Secondly, how threads of SMT can be distributed among different virtual machines? Obviously, if the base operating system is Linux kernel 2.6, we want to distribute one physical CPU to one virtual machine, as described in Section 2.2.2 scheduler would assign threads of the same program to the same CPU, maximizing utilization of resources such as CPU caches and instruction buffers. Thus, putting the resource available as the same physical CPU for each virtual machine avoids possible performance degradation due to resource being pulled from other physical CPUs.

4 Thesis Conclusion

4.1 *Virtualization Challenges and SMT Advantages*

Virtualization introduces additional levels of abstraction and additional overhead, thus, it is important to optimize the overhead via appropriate configuration. Unlike a non-virtualized system which has abundant resources, the resource limitations in virtualization usually drive up context switching rates occasionally at multiple levels of abstraction. Therefore, it is important to know and test beforehand the system's physical resources available.

Second, scheduling across virtual machines is using assumed to be equitable and consistent; it actually depends on the operating system that is running and schedulers' design. For instance, Linux 2.6 kernel has a better scheduler than 2.4 which accommodates SMT better. In this case, VM can make more use of the benefits of SMT.

Third, how will the resources be partitioned for the VMs are important, see load balancing and distribution (3.5). Some virtualization monitors will provides various options to map physical CPUs to virtual CPUs and to create affinity between certain sets or allow a more general pool of recourse to be shared amongst all VMs. However, in the case of SMT, it is not advice to separate a single CPU's two hyper-threads among two different virtual machines since it might not increase performance. The branch prediction unit becomes less effective when shared, because it has to keep track of more threads, with more instructions, and will therefore be less efficient at giving an accurate prediction. This means that the pipeline will need to be flushed more often due to mispredicts, but the ability to run multiple threads more than makes up for this deficit. The penalty for a mispredict is greater due to the longer pipeline used by an SMT

architecture (by 2 stages), which is in turn due to the rather large register file required. However, there has been research into minimizing the number of registers needed per thread in SMT architecture. This is done by more efficient operating system and hardware support for better deallocation of registers, and the ability to share registers from another thread context if another thread is not using all of them.

4.2 Performance Determination and Analysis

As stated in Chapter 3, we have generalized a model for performance analysis of SMT and suggested methods for comparison, such that, we have in comparing the impact of different parameters, we have the “apple to apple” comparison, see [34].

4.3 Suggestions

There are several suggestions to improve accuracy in the performance comparison:

Load distribution of the machine is different at different period of time, e.g. usage can vary from day to night. This need to taken into calculation for different models.

The number of data collected. During all 12 official tests, the durations were about 1 hour. Each test has around 360 data points. The number of data can increase to ensure the statistic accuracy.

The distribution of the data points, in the tests, normal distribution was used. However, in some rare case, the data might no necessary has statistic meaning which need to be aware of.

Number of components taken to the consideration. There are several major factors to be used, such as CPU, memory and I/O. However, for different test case we might want to test other components, for example, the X gui factor or graphic response speeds for games, UI interactive rate.

5 Future Work

This chapter suggests future work can be done and the technology under development.

5.1 Multiple virtual machines with SMT mode

This is the case of multiple virtual machines on the base operating system, so that virtual machines share the resources. However in this setting, there can be several combination of performance analysis.

Case 1

All the virtual machines are running but no application on top are running. This is to see the performance overhead of multiple virtual machines.

Case 2

This is the case of one virtual machine is running application and the rest of the virtual machines do not run applications. This case shows how ideal virtual machines affect the virtual machine that is running application.

Case 3

In this case, multiple virtual machines are running applications. Performance degradation is expected. We want to see how much decrease is here with respect to the case with one virtual machine running application and others idle. [30.]

5.2 VMware with Oracle

VMware with Oracle in process see [37.][38.]. Since virtualization causes degradation, the integration of technology between VMware and Oracle would have a beneficial impact on the perspective of virtualization.

5.3 Discussion on IBM P595 Micro partitioning

Starting with the IBM P595 server, [28.]the advance properties of SMT provide an enhanced micro-partition technology for virtualization, called Advanced Power Virtualization (APV). For IBM machines, p5 Hypervisor is the virtualization engine behind the APV technology. This technology divides a physical processor's computing power into fractions of processing units and shares them among multiple LPARs(Logical Partitions). For example, we could allocate as little as 0.10 processing units as opposed to dedicating an entire CPU. There are two main advantages, better utilization of physical CPU resources and more partitions (not limited by number of physical CPUs). Unlike the current model we have, which is to bind the guest operating system to each physical CPU, micro-partitioning further utilize the available resources. Thus, the future work for virtualization and SMT, can be investigation on this APV technology, which is expecting to have a better performance model.

Reference

- [1.] D. E. Tullsen, S. J. Eggers, H. M. Levy. *Simultaneous Multithreading: Maximizing On-Chip Parallelism*. The 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, June, 22-24, 1995, 392 - 403.
- [2.] D. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. *Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor*. In Proceedings of the 23rd Annual Intl. Symposium on Computer Architecture, pages 191--202, May 1996.
- [3.] S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, and D. M. Tullsen, *Simultaneous Multithreading: A Platform for Next-Generation Processors*, IEEE Micro, vol. 17, no. 5, pp. 12--19, Sept./Oct. 1997.
- [4.] R. P. Preston et al.. *Design of an 8-wide superscalar RISC microprocessor with simultaneous multithreading*. ISSCC Digest and Visuals Supplement, February 2002.
- [5.] H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, S. R. Kunkel, *Characterization of simultaneous multithreading (SMT) efficiency in POWER5*
- [6.] M. J. Serrano, *Performance Estimation in a Simultaneous Multithreading Processor*, Fourth IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'96)
- [7.] Hensbergen, *P.R.O.S.E.: partitioned reliable operating system environment*, ACM SIGOPS Operating Systems Review, April 2006
- [8.] R. Sailer, et al, *Secure Hypervisor approach to trusted virtualized systems*, IBM Research Report RC23511 in 2005
- [9.] M. Mergen et al, *Virtualization for high performance computing*, IBM Research Report 2006
- [10.] *IBM Journal of Research and Development issue 49-4/5 : PR5 and Packaging* Volume 49, Number 4/5, 2005
- [11.] Y. Wei, S. Son et.al, *QoS Management in Replicated Real-Time Database*, 24th IEEE International Real-Time Systems Symposium (RTSS'03) p. 86
- [12.] S. Hwang, N Jung et. al, *Dynamice Scheduling of Web Server Cluster*, Proceedings of the 9th International Conference on Parallel and Distributed Systems, 2002
- [13.] V. Cardellini et. al., *Dynamic Load Balancing on Web Server Systems*, IEEE Internet Computing, May 1999
- [14.] D. Xu, *Multivariate Statistical Modeling and Robust Optimization in Quality Engineering*, Ph.D Dissertation, October 2001
- [15.] G. Marin, J. Mellor-Crummey, *Cross-Architecture Performance Predictions for Scientific Application Using Parameterized Models*, ACM SIGMETRICS Performance Evaluation Review, June 2004

- [16.] J. Kleijnen, *Validation of Models: Statistical Techniques and Data Availability*, 1999
- [17.] P. Dibble, *Migrating to Linux kernel 2.6*, Linux Devices.com
- [18.] R. Love, *Linux Kernel Development*, Novell Press, 2nd Edition
- [19.] J. Werfel, MATLAN, Statistics, and Linear Regression,
<http://hebb.mit.edu/courses/9.29/2004/lectures/optional01.pdf>
- [20.] A. Gullickson, *Introduction to Multivariate Regression*, Introduction to Social Data Analysis, http://www.columbia.edu/~ag2319/teaching/G4074_Outline/
- [21.] VMware Workstation .5.5, release note,
http://www.VMware.com/support/ws55/doc/releasenotes_ws55.html
- [22.] VMware Workstation 5.5, requirements,
http://www.VMware.com/support/ws55/doc/intro_hostreq_ws.html#wp1000805
- [23.] VMware Workstation 5.5 Support Document,
http://www.VMware.com/support/ws55/doc/ws_devices_2way_vsmp.html
- [24.] Intel Processor Identification,
<http://support.intel.com/support/processors/tools/piu/sb/CS-015823.htm>,
- [25.] Intel Processor Code Name,
http://en.wikipedia.org/wiki/List_of_Intel_codenames, by Wikipedia
- [26.] J.Sugerman, G. Venkitachalam and B. Lim, *Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor*, Proceedings of the 2001 USENIX Annual Technical Conference
- [27.] M. Heath, Scientific Computing, An Introductory Survey, 2nd Edition, Mc Graw Hill
- [28.] M. Bush, Configuring a Virtual I/O Server, Sys Admin: The Journal for Unix and Linux Systems Administrators, July 2007
- [29.] Booting Server Virtualization Performance, IT @ Intel Brief, Feb 2007
- [30.] E. Bolker, Y. Ding, Virtual Performance won't do: Capacity planning for Virtual Systems, BMC Software
- [31.] E. Bolker, Measuring and Modeling Hyper-threaded Processor Performance, U.Mass Boston Presentation, Sep 2003
- [32.] J.P. Casazza et. al, Redefining Server Performance Characterization for Virtualization Benchmarking, Intel Technology Journal, August 2006.
- [33.] N. Carr, Linux Kernel 2.6 Features in Red Hat Enterprise Linux, Technical Brief Red Hat Inc, 2002.
- [34.] Performance Tuning and Benchmarking Guidelines for VMware Workstation, VMware Technical Resources 2007
- [35.] TPC BenchmarkTMC Full Disclosure Report for IBM eServer p5 595

- [36.] Thomas Kyte, Expert Oracle Architecture—9i and 10g Programming Techniques and Solutions, APress
- [37.] Installation Guide for Oracle with VMware,
http://www.Oracle.com/technology/products/oem/extensions/plugin-vmware_esx.html
- [38.] Guideline for VMware user using Oracle, <http://www.vmware.com/Oracle>

Appendix

A. Terminology

Abbreviation	Explanation
MT	Fine-grained multi-threading
SMT	Simultaneous multi-threading
HT	Hyper thread / Hyper threading, which is essentially SMT and can be used interchangeably
CMP	Chip multiprocessing
SMP	Symmetric multiprocessing / Symmetric multiprocessors
client OS	Client operating system located on the virtual machine
host OS	Host operation system at which the virtual machine is located
VMware	Virtualization technology developed by VMware company
Hypervisor	Virtualization technology developed by IBM
RHEL	RedHead Enterprise Linux
ESX server	Virtualization technology developed by VMware company which essentially is the host OS
VMM	Virtual machine monitor
LV	Low voltage
SVR5	Unixware System V Release 5
NPTL	Linux's Native Posix Threading Library
SGA	Oracle: System global area
PGA	Oracle: Process global area
UGA	Oracle: User global area
Awr	Oracle: Auto work repository (used to get the Oracle transactions per second)

Filename: noel_thesis_5.doc
Directory: C:\Documents and Settings\Noel\My Documents\Research
Template: C:\Documents and Settings\Noel\Application
Data\Microsoft\Templates\Normal.dot
Title: Abstract
Subject:
Author: n
Keywords:
Comments:
Creation Date: 1/4/2008 1:15:00 PM
Change Number: 8
Last Saved On: 1/4/2008 1:41:00 PM
Last Saved By: n
Total Editing Time: 29 Minutes
Last Printed On: 1/4/2008 1:41:00 PM
As of Last Complete Printing
Number of Pages: 62
Number of Words: 11,530 (approx.)
Number of Characters: 65,721 (approx.)