PROBABILITY BASED CACHE REPLACEMENT ALGORITHM FOR THE HYPERVISOR CACHE

A Thesis by

Abhiram Madhugiri Shamsundar

BE, Vishveswaraya Technological University, India, 2006

Submitted to Department of Electrical Engineering and Computer Science
and faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

July 2012

PROBABILITY BASED CACHE REPLACEMENT ALGORITHM FOR THE HYPERVISOR CACHE

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science with a major in Computer Networking.

_____
Ravi Pendse, Committee Chair

_____
Janet Twomey, Committee Member

_____
Abu Asaduzzaman, Committee Member

DEDICATION

To my family, for their continuing support and patience; to all my friends for their significant advice and time throughout the completion of my thesis.

ACKNOWLEDGEMENTS

I would like to sincerely thank my thesis advisor, Dr. Ravindra Pendse for his devoted motivation and supervision throughout my career at Wichita State University. His guidance helped me complete my thesis successfully.

I take this opportunity to thank Vidya Suryanarayana for her constant support and guidance throughout my thesis. Her suggestion and advice helped me understand the technology and gain more knowledge. I would like to thank members of the committee for their effort and time in reviewing this thesis.

I would like to extend my gratitude towards to Amarnath Jasti, Vijay Ragothaman, Mahesh and all those who directly or indirectly helped motivate me with my research.

ABSTRACT

Virtualization is one of the key technologies which help in server consolidation, disaster recovery, and dynamic load balancing. The ratio of virtual machine to physical machine can be as high as 1:10, and this makes caching a key parameter, which affects the performance of Virtualization.

Researchers have proposed the idea of having an exclusive hypervisor cache at the Virtual Machine Monitor (VMM) which could ease congestion and also improve the performance of the caching mechanism. Traditionally the Least Recently Used (LRU) algorithm is the cache replacement policy used in most caches. This algorithm has many drawbacks, such as no scan resistance, and hence does form an ideal candidate to be utilized in the hypervisor cache. To overcome this, this research focuses on development of a new algorithm known as the "Probability Based Cache Replacement Algorithm". This algorithm does not evict memory addresses based on just the recency of memory traces, but it also considers the access history of all the addresses, making it scan resistant.

In this research, a hypervisor cache is simulated using a C program and different workloads are tested in order to validate our proposal. This research shows that there is considerable improvement in performance using the Probability Based Cache Replacement Algorithm in comparison with the Traditional LRU algorithm.

TABLE OF CONTENTS

Chapter Page

TABLE OF CONTENTS (cont.)

Chapter Page

CHAPTER 1 - INTRODUCTION

1.1 Storage

Storing data on the network is the order of the day and is continuously expanding. Close to 75% of all hardware expenditure is spent on the resources necessary for storage [3] in an enterprise storage scenario. Storage automatically becomes a focal point of interest where expenses need to be cut down and hence there has been considerable amount of research in this area.

There are three major variations in storage:

a) Direct Attached Storage (DAS)
b) Network Attached Storage (NAS)
c) Storage Area Network (SAN)

1.1.1 Direct Attached Storage

The storage setup where the storage devices such as hard disks, tape devices, Redundant Array of Independent Devices (RAID), are directly connected to the client computer processors is known as Direct Attached Storage. These devices are connected via cables/adapters/buses using various protocols such as a Fiber Channel (FC), a Small Computer System Interface (SCSI), and a Serial ATA (SATA). Sometimes, such a setup is also referred to as a Capacitive Storage or Server Attached Storage. The simplest example of this can be the hard disks connected to the CPU of a desktop/laptop computer via SATA buses. One of the important points to note here is that storage is local to the computer processor, so each vendor implements their storage variations. There is no fixed standard for this or we can term the standards as weak.

Another disadvantage associated with DAS is the maintenance of such an implementation, where large numbers of DAS systems are connected.

1.1.2 Network Attached Storage (NAS)

To overcome the disadvantages posed by DAS, where data storage is bound to individual computers, network storage came to light. The two variations for this method of storage model are the Network Attached Storage (NAS) and the Storage Area Network (SAN).

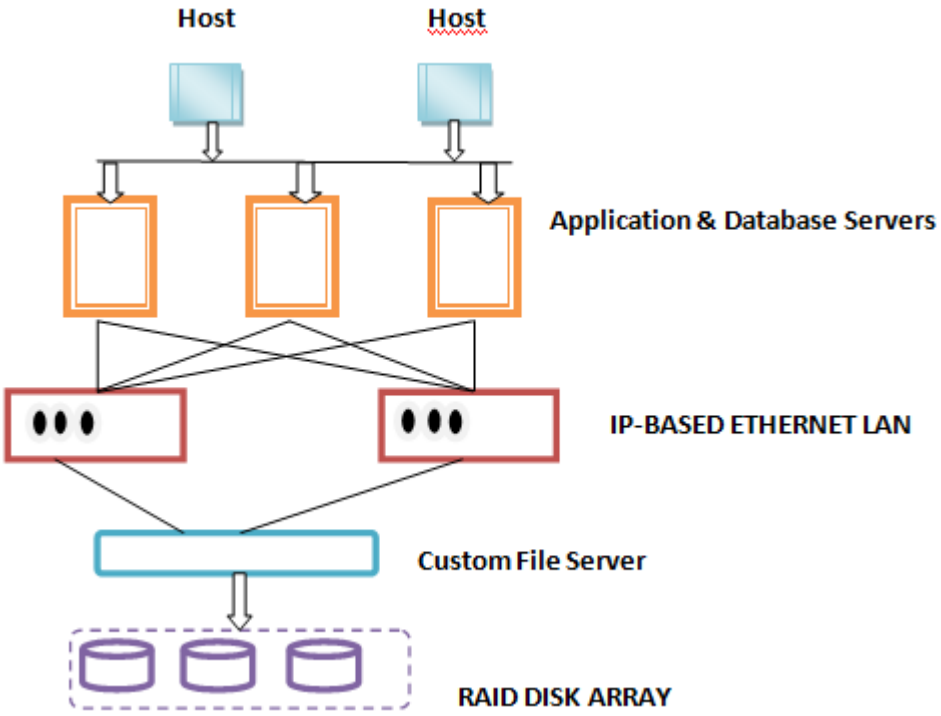

Figure 1: Diagrammatic representation of NAS technology

NAS is based on the existing IP and Ethernet technologies, which with the help of NAS protocols such as NFS or CIFS, and it replaces the general file server with a network attached storage system [5]. NAS implements file-level I/O operations, which involve a higher level of abstraction. Although this comes out as an overhead, it also brings in the advantage of ease of use and implementation. Windows and UNIX systems are equipped with support for NAS

protocols such as Network File Systems (NFS) and Common Internet File Systems (CIFS). NAS usually consists of an internal processor and/or storage disks [2] (If it does not have storage disks, it can then also be presented to a SAN with the help of the Host Bus Adapter (HBA) of the NAS device.) [4] Figure 1 shows the implementation of NAS technology.

1.1.3 Storage Area Network

Storage Networking Industry Association (SNIA) describes SAN as "Any high performance network that enables communication between storage and computer systems" [1].SAN, unlike NAS, is employed in a dedicated high-speed network. SAN may use a Fiber Channel (FC) or an iSCSI (the new technology where the SAN employs Ethernet as well) to provide any-to-any connectivity between clients and the storage devices. Input/output (I/O) is very similar to DAS and is block-oriented and thus provides better performance in comparison to NAS. Figure 2 shows the implementation of SAN technology.

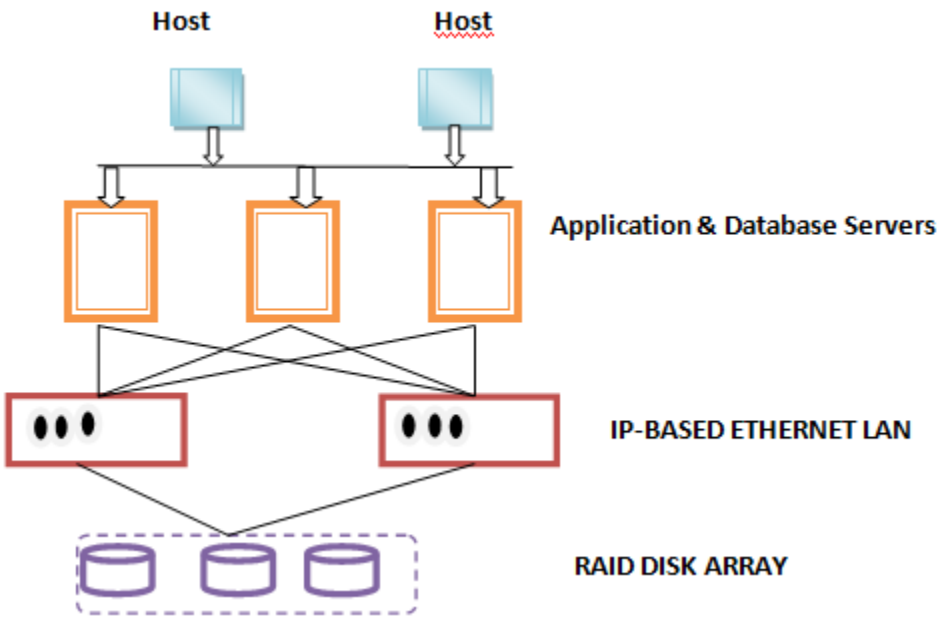

Figure 2: Diagrammatic representation of SAN technology

1.1.3.1 Advantages of SAN

SAN brings in many advantages: scalability, improved performance in comparison to DAS, data isolation, redundancy, and uptime, all of which can be managed through from a central interface. [6]

a) Scalability: Disks can be added as the demands increase and more storage space is required. It makes things easier from an implementation perspective.

b) Improved performance in comparison to DAS: the SAN has its own dedicated high speed network for data flow and does not interfere with LAN traffic, which generally forms the bottleneck which raises performance issues.

c) Data Isolation: Since data does not reside on the enterprise LAN, data is safe and isolated.[12]

d) Uptime: Addition of additional disks does not require any downtime, they can be added without bringing the production system down, leading to 100% uptime.

1.1.3.2 Disadvantages of SAN:

Although SAN brings in a lot of advantages, there are a few drawbacks which need to be addressed, including single point of failure, performance (which needs to be improved further), disaster recovery, need for cheaper options for data archiving, and, most important of all, poor utilization of Disk space. [7]

a) Single point of failure: If there is a failure in any elements in the path from storage to client or if there is a single disk failure amongst the array of disks, it would lead to huge downtime which is totally unacceptable for production environments.

b) Performance Improvement: Storage system (without virtualization) performance is complex to manage in real time.

c) Disaster recovery and data backup: Not many options available for disaster recovery and data backup options are a costly affair.

Virtualization addresses all the above mentioned problems for storage and hence "Storage Virtualization" becomes a very important aspect which needs more attention. The next section presents a detailed discussion on Virtualizaton

1.2 Virtualization

1.2.1 Introduction

Virtualization is broad term used for various virtualization techniques employed on different platforms including, but not limited to, cpu, hardware, server, etc. Virtualization in its broadest sense refers to utilization of common hardware to serve different clients, where the client may be either aware or unaware of the fact that it is utilizing shared resources.

1.2.2 History of Virtualization

Virtualization was a concept developed by IBM in the early 1960s, for IBM mainframes. In this design, the mainframe computers, which usually had large amount of RAM and hard disk space, would logically be divided into virtual machines to utilize the hardware available on the expensive mainframe computers. But with the evolution of modern computers, desktops and laptops, development from a virtualization perspective was at a standstill. Later, the disadvantages of laptops and desktops started surfacing, with problems like no disaster recovery, backup issues, high maintenance costs, and increasing management expenses. To overcome these

problems, VMware came out with the concept of virtualizing the existing x86 platform, where this platform could be used to provide isolation in a shared hardware infrastructure. Since this was released as open source, it led to wide implementation and development of this technology. Once this technology gained recognition, leading players in the market leaped towards adopting it, Microsoft bought Connectix and released MS virtual PC in 2004. Similarly, others hopped onto the bandwagon as well, a few of the companies which are worth mentioning are - Xen, Xen Source, OpenVZ, SWSoft, OpenSolaris. Below are a few very important motivational factors to move into the virtualized world:

a) Monetary reasons: By implementing virtualization, we are saving on hardware and expenses that come with hardware (maintenance, electricity etc).
b) It helps us in moving towards a cleaner and greener environment.
c) The admin overhead is considerably reduced as the number of physical devices to manage is reduced.
d) During times before virtualization, we could see that the hardware would always be underutilized. This could easily identified by abundant CPU idle cycles on the physical hardware. But virtualization makes better use of the CPU and hardware available and, in turn, again saves on money.
e) Since the emergence of virtualization, the procedure to install new software on each physical machine has become outdated and life has been made easier with the help of virtualization where installing on one Virtual machine can be replicated from there on.

1.2.3 Classification of virtualization

There are different ways to classify virtualization, one of the ways is to classify it based on how much the guest OS (Virtual Machine) might interact with the hardware. [9]

a) Full Virtualization

b) Para Virtualization

c) Hardware assisted Virtualization

1.2.3.1 Full Virtualization

Before delving into the details of full virtualization, we would need to understand the basics of hypervisor/Virtual machine Monitor (VMM). The hypervisor or VMM is a program which governs the guest operating systems or virtual machines by scheduling their tasks on the physical hardware.[13] In Full Virtualization the hypervisor acts and intercepts the attempts from the Guest Virtual Machine to access the physical hardware and the hypervisor emulates that hardware's functionality within the software. Thus, this layer of abstraction is transparent to the guest virtual machine so the guest VM will never realize that it is residing in a virtualised world. This layer of abstraction comes at a cost: is slowness of the VM. Since all the VM's have to get their requests through the Hypervisor, this will cause slowness and hence affect the performance. Some of the examples for full virtualization are: VMware, Intel VT (Virtualization technology)

1.2.3.1.1 Working of full virtualization on Intel x86 processors

In case of Intel x 86 processors, there are 4 rings identified as 0,1,2,3 which indicate the privileges, Operating systems are on Ring 0 as it needs to execute the instructions on the hardware, whereas, user applications run on ring 3. In full virtualization, the hypervisor or VMM provides all the resources like virtual BIOS, virtual memory and virtual devices. So the VM will never know about its existence in a virtualized world. This is possible due to binary translation.

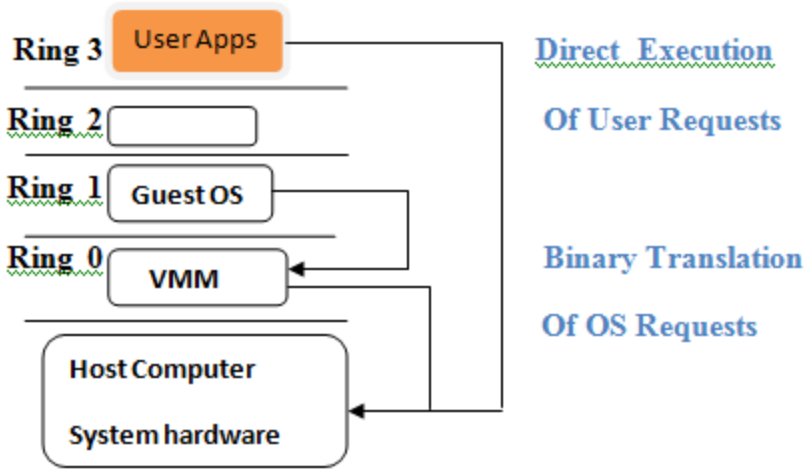

Figure 3: Full Virtualization

Advantages of full virtualization:

a) Guaranteed Isolation

b) Ease of Migration

c) Any virtual machine can be run on the hypervisor with no modification.

d) Does not require the hardware itself, to support virtualization.

Disadvantages of full virtualization:

a) Overhead on the OS/hypervisor

b) Affects the performance

1.2.3.2 Para Virtualization

In this form of virtualization, the guest VM will know that it is working in the virtualized world and it is sharing resources from the same hardware where other VMs are running as well. The Guest VM resides 3 levels above the actual hardware. The Hypervisor in this case resides on a layer above the Host OS in contrast to residing the in the OS in the full virtualization scenario.

Fig 4 shows the working of para-virtualization in x86- architecture. Para-virtualization involves modifying the guest OS kernel in order to modify all the non-virtualizable instructions. The hypervisor, as shown in the figure, provides "hypercalls" memory management interrupt handling and other such activities. The most important thing to take home from the para-virtualized architecture is that the overhead from the virtualization perspective is considerably lower when compared to full virtualization. One of the key players in this field is Xen, which provides para-virtualization support by partially abstracting a portion of the underlying hardware to the hosted operating system which exposes some aspects of the machine as "drawbacks" on the Guest VMs. Para-virtualization runs faster because the guest OS is modified to work at ring 1. The Xen hypervisor is one of the few popular implementations of para-virtualization.

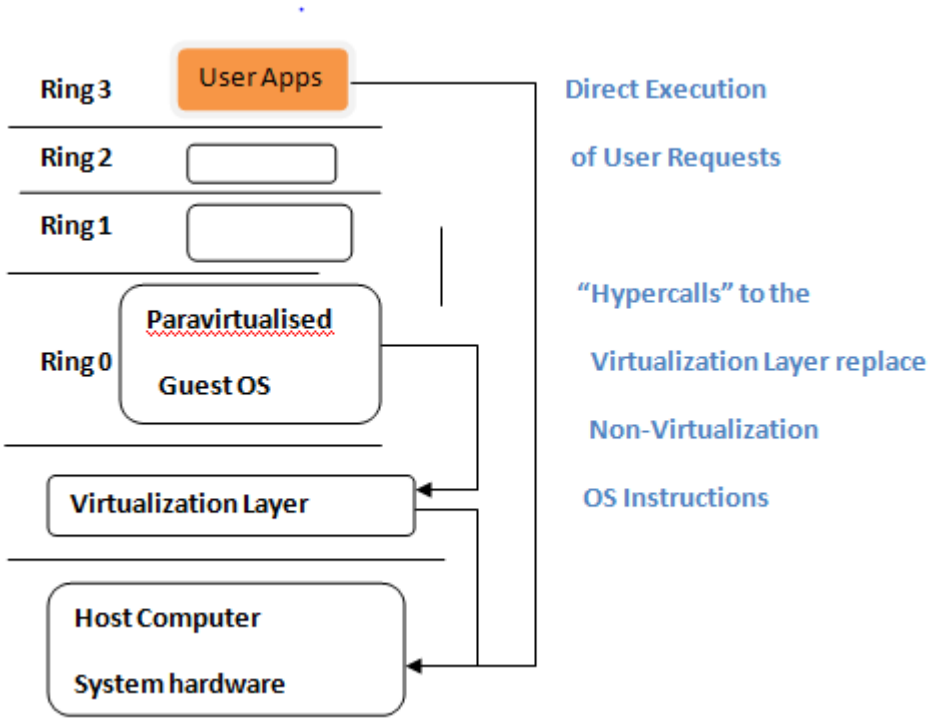

Figure 4: Para-Virtualization

Advantages:

a) Faster and hence better performance
b) Works well with all Linux systems.
c) The hypervisor will not have to intercept any instructions

Disadvantages:

a) Cannot work with Windows, as this involves modifying the windows kernel, but Windows is proprietary.

1.2.3.3 Hardware assisted Virtualization

Hardware-assisted virtualized virtualization works on the basis of running the Guest VMs at ring 0. One of the key players in this is XenServer, and after the introduction of virtualization in their x86 and AMD platforms, there is a DM0, which is also a VM, but has special privileges which can initiate device drivers to perform I/O tasks on the guest VM's behalf. Currently Intel's VT-x and AMD-V support this type of virtualization. [11]

Hardware-based virtualization provides the best of both worlds, which is full virtualization and para-virtualization. [10] Since there is no need for rewriting the instructions, it is definitely faster than full virtualization and since its hardware based, it is faster than para-virtualization as well. The way it works is that hardware virtualization gives us the freedom to create a root mode (trusted) and a non-root mode (un-trusted) and still work separately from 0 to 3 rings as per the ring architecture. So the root mode is occupied by the hypervisor and the guest VMs are placed in the non-trusted mode. The binary translations do not come into picture in this case as the sensitive and privileged calls are set as a trap to the hypervisor. Some of the major

vendors who provide support for this type of virtualization are VMware ESX server, XenSource and Microsoft.

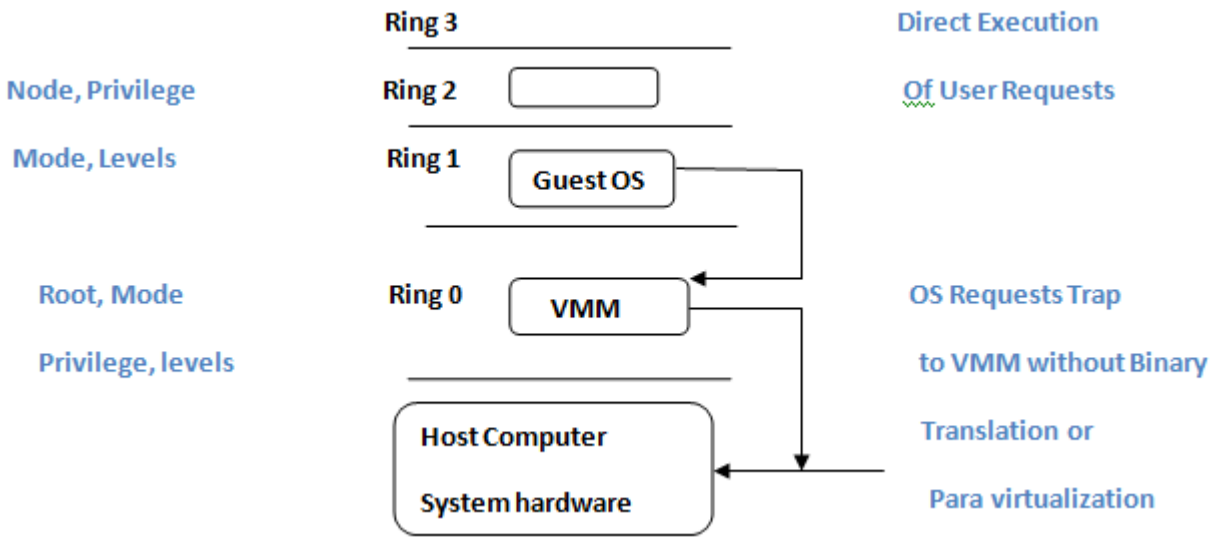

Figure 5: Conceptual Working of Hardware Assisted Virtualization

Advantages:

a) High performance

b) Lower latencies

Disadvantages:

a) Requires hardware support

1.2 Drawbacks of Virtualization

Listed below are a few hindrances and drawbacks of virtualization as a whole, which is preventing it from occupying the market completely: [8]

a) Since we are cutting down on the number of physical resources, this would mean our dependency on fewer hardware resources increases, which is not very useful in terms of reliability

c) In spite of having the luxury of buying lesser hardware for implementation, there are still the costs of licenses for individual software that we use and will increase over time.

d) There are many security concerns at this point of time. Since multiple customers have their data on the same physical machine, the customers do not feel safe about their data.

e) Server virtualization has its own issues as well: using server virtualization for applications which have unpredictable resource requirements could increase the danger of crashing the VM in case of a sudden surge in data requirements.

f) It is difficult to troubleshoot if there is an issue with the setup

g) As resources are divided, the performance levels come down.

h) The migration of the VM across different processors from different manufacturers is not possible.

i) If we create multiple VMs to cater multiple applications on the same hardware, the performance level might be reduced if the applications are too demanding for the CPU.

CHAPTER 2- CACHE

2.1 Introduction to Cache

In computers, the programs and the data are stored in the memory; such computers are called as Von Neuman Computers. Such computers face a drawback; memory access becomes a bottleneck because the instructions are stored in the memory and the computer processor has to make memory references for each instruction to be executed. Multiple instructions and executions, which are common in modern computers, cause a bottleneck known as the Von Neuman Bottleneck. [14] Caching is one of the most important tools used to cut down the effects of this problem. The cache is a small, temporary storage which stores local copies of selected data and is accessible at high speeds. The cache is placed on the route between the processor and the memory which is configured to intercept all instructions and handle them accordingly. Figure 6 shows a conceptual representation of where a cache is placed.

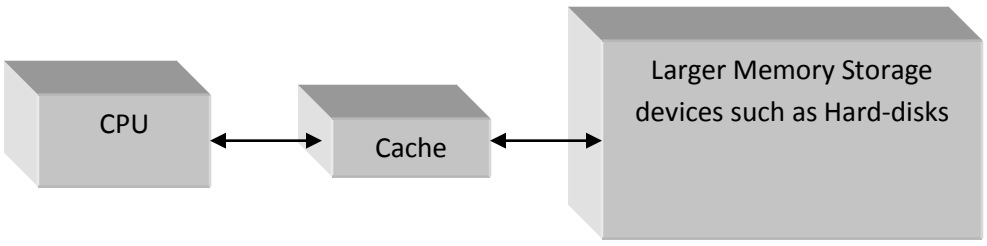

Figure 6: Conceptual representation of caching mechanism

2.2 Features of the Cache

There are few very important features of the cache which makes it work efficiently:[14]

1) Size: The cache is usually very small and hence is very economic. The size of the cache ranges from than less than 1 percent to a maximum of 10 percent of the data store

(Example: hard-disk). Therefore it is crucial to decide upon what data items are to be stored on the cache.

2) Speed: Since the cache is on-board temporary memory, the amount of time required to fetch the data from the cache is much less, compared to fetching data from a large memory storage device.
3) Activity: The cache is very active and has to intercept each and every request, look it up to see if the requested data is present in the cache, and if not, it must retrieve the same storage device, and finally decide whether or not to keep the data item in the cache.
4) Transparency: The cache is transparent to both the processor and the storage device. There are no changes required, neither on the processor end nor the data store end.
5) Automatic: The cache is configured to run a particular algorithm (Example: Least Recently Used, First In First Out) for decision-making purpose and does not require an intervention or any instructions for it work.

Caching can handle data related to both hardware, software, different types of data such as textual, non-textual, and application specific items such as web pages, and database entries. It can also handle data types of different sizes: a byte, a word, a segment, or even a complete program and so is very flexible in nature. A few of the most important terminologies used in caching are cache block, cache line, cache set, tag, associativity, valid bit, dirty bit, cache hit, and cache miss. [15]

2.3 Important Terminologies in Cache

a) Cache Block: This is the storage unit of the caching mechanism; it is the smallest unit that can be transferred back and forth between the main memory and the cache.
b) Cache line: The cache memory is usually divided into units known as the cache line, and this term is used interchangeably with cache block.

c) Cache Set: Every single row in a cache memory is called the Cache set. Based on the associativity, the number of blocks per row or set is determined. There are three different types of associativity: direct mapped, fully associative and set associative.

d) Associativity: The cache set can further be divided into sectors and the associativity is determined based on how many sectors are made.

   i) Direct mapping: In this type of caching mechanism, each block of address in the data store (main memory) has only one location where it can be stored. If two or more are addressed in the main memory map to the same location in the cache, then only one of them can reside any given point of time.

   ii) Fully Associative: In this type of caching mechanism, any block of address in the main memory can map to any location on the cache memory, which gives more flexibility. However, finding the requested block of address in the cache becomes a complex operation, since it reside anywhere on the cache.

   iii) Set Associative: In this type of caching mechanism, the block of address is restricted to particular sectors. Figure 7 provides a good idea of the different terminologies used in the cache.

e) Tag: For groups of data, a unique identifier is used to locate blocks residing on the cache. Since data might be residing in different parts of the cache, a tag makes it easier for identification.

f) Valid Bit: As the name indicates, based on the binary value 1 or 0, it shows if data in a block is valid (1) or not (0).

g) Dirty: It is an identifier which indicates if the data residing in the cache is modified in comparison to the corresponding data residing in the main memory.

h) Cache Hit: If the data requested from the processor is present in the cache memory, then it is considered as a cache hit.

i) Cost of Cache Hit: It represents cost incurred to retrieve a data block requested by the processor which is present in the cache.

j) Cache Miss: If the data requested from the processor is not present in the cache memory and needs to be fetched from the main memory then it is considered to be a cache Miss.

k) Cache Miss Cost: It is the cost incurred to retrieve a data block requested by the processor which is not present in the cache and needs to be retrieved from the main memory. This cost is always higher than the Cache Hit Cost. Therefore, we would prefer to have a lot of cache hits, hence improving the performance.

	Sector 0								
	Block 0					Block1			
Set 0	Tag	V	D	Word	Word	V	D	Word	Word
Set 1	Tag	V	D	Word	Word	V	D	Word	Word
Set 2	Tag	V	D	Word	Word	V	D	Word	Word
Set 3	Tag	V	D	Word	Word	V	D	Word	Word
Set 4	Tag	V	D	Word	Word	V	D	Word	Word
Set 5	Tag	V	D	Word	Word	V	D	Word	Word
Set 6	Tag	V	D	Word	Word	V	D	Word	Word
Set 7	Tag	V	D	Word	Word	V	D	Word	Word

Sector 1								
Block 0					Block 1			
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word
Tag	V	D	Word	Word	V	D	Word	Word

Figure 7: Conceptual Representation - Working of a 2 Way set associative cache

(Since it is a 2 Way set associative, we see the cache divided into two sectors)

l) Locality: It is referred to as an inclination to reference data blocks that are placed closer to recently referenced data blocks or data blocks that were recently referenced themselves.

i) Temporal Locality: It is described as the tendency of a memory location which was referenced once as likely to be referenced numerous times in the near future.

ii) Spatial Locality: It is described as the tendency that once a memory location has been referenced, then the program is likely to reference a data block near this particular data block.

2.4 Caching in Virtual Machines

In [18], the authors list the advantages of moving specific functions to a hypervisor. Virtualization is one of the most important implementations in the world of storage and server management. It brings in several advantages, such as increased flexibility of resource allocation, isolation of data and service, and allowed migration of workload as well. All these advantages are possible because of the Virtual Machine Monitor (VMM), or Hypervisor, in a virtualized environment. The Hypervisor is a software layer that performs the process of virtualizing the hardware interface reflecting the underlying machine architecture [19]. The Hypervisor acts as the key resource manager for all the Virtual Machines which it will host. Hence this can be attractive place for a number of features such as caching, scheduling and monitoring [18]. Such features are generally implemented at the Operating System (OS) level, but there are many potential benefits which we can reap by implementing them at the Hypervisor level.

a) The services which are implemented on the Hypervisor can be ported to many Virtual Machines running on the same Hypervisor.
b) New features can be implemented at the Hypervisor level; this cannot be possible at the Operating System level if the legacy system is closed source.
c) Also, the Hypervisor is what controls the resource allocation and hence should be the ideal place where features such scheduling, caching, and monitoring should be implemented.

But this also brings a disadvantage known as the semantic gap. The services provided by the virtual machine function below the abstractions provided by the guest OS and the applications. This lack of knowledge about the higher levels in the Hypervisor is known as the semantic gap [20]. One other disadvantage of the virtualized world includes the overhead that it brings in. This might include interception of the systems calls and re-directing it to the right guest OS. But the advantages outweigh the disadvantages of using the Hypervisor and virtualization.

CHAPTER 3 – LITERATURE SURVEY

3.1 Related Work

3.1.1 Caching at the Hypervisor level

The authors of [17] look at the need for caching at the Hypervisor level. Following are important reasons which stress the importance of integrating the cache at the hypervisor level.

a) The guest OS may not have a large cache buffer and therefore might be limiting the performance of the virtualized system. Hence, a cache at the Hypervisor can exceed the natural addressing limits of the guest OS.
b) It increases performance, when the cached element is shared among several other guest Operating Systems, making this a viable option, instead of accessing the main memory whenever there is a requirement for a shared element. A similar concept of storing pages in intermediate software was demonstrated using a prototype known as Disco [21]. The demonstration shows considerable improvement in the performance using virtualization.
c) If there is a remote storage server cache, the hypervisor can explicitly communicate with it and exchange cache information as necessary.

Since the Hypervisor and caching at the hypervisor can lead to very important advantages, it is essential for this hypervisor cache to be very efficient in terms of performance. The most popular algorithm which is used is the Least Recently Used and this has its disadvantages as discussed before. The Hypervisor cache is an exclusive cache, which means, the cache holds data are not present in the cache of the virtual machine [17]. The locality of the data blocks in this cache is lesser due to the fact that the hypervisor is not the first level of caching. Hence the LRU algorithm will not be the most ideal policy to implement. In this thesis,

we propose an algorithm which would take the probability of a memory access into consideration for caching and replacement policies without significant overhead on the processor.

3.1.2 Cache Performance

Douglas Comer [14] stresses the performance of the cache as one of the key areas to concentrate on. Cache performance directly impacts the speed of the processor, with growing importance of storage; cache plays a vital role keeping up with the ever-increasing storage size and helping improve the performance of the processors. Cache performance can be analyzed using the cache hit, cache miss, and Cost of cache parameters.

The cache hit ratio (HR) is defined as the ratio of number of requests which were hits (RH) to that of the total number of requests (TR).

Hit Ratio = (Number of requests which were hits/Total number of requests)

*Hit Ratio Percent = (RH/TR) * 100*

The cache miss ratio (MR) is defined as the ratio of number of requests which were misses (RM) to that of the total number of requests (TR).

Miss Ratio = (Number of requests which were misses/Total number of requests)

*Miss Ratio percent = (RM/TR) * 100*

The cost of access can be written in terms of cache hit cost, cache hit ratio, and cache miss ratio.

*Cost of Access = (HR * Cache Hit Cost) + [(1 – HR) * Cache Miss Cost]* → *Equation 1*

OR

*Cost of Access = (MR * Cache Miss Cost) + [(1 – MR) * Cache Hit Cost]* → *Equation 2*

(Note: HR = 1 – MR)

From the above equations, the Cache Miss Cost is not something that can be changed, since the cost to retrieve a block of data from the main memory remains the same. The Cache Hit cost can change based on the location of the cache (on-board or off-board). Hence the only parameter that can be improved upon to increase the cache performance is the Hit Ratio (Equation 1) or Miss Ratio (Equation 2). A higher Hit Ratio or lower Miss Ratio will mean higher improvement in the performance of Cache. Caching mechanisms implement different replacement policies in order to achieve higher Hit Ratios. The replacement policy is called upon when the cache is full and a new response to a request arrives. The replacement policy can either evict a current resident on the cache to accommodate the new response, or can just ignore the new response. The decision of evicting or ignoring is based on these policies. There are different kinds of policies/algorithms used based on the need. A few of the most popular are Least Recently Used, First In First Out, Least Frequently Used, Most Recently Used and Random.

Replacement Policies

a) Least Recently Used (LRU): In this policy, the cache tracks the usage of data blocks and the block with the oldest usage time is discarded. Two important features of this policy are speed and is adaptive in nature. The drawback of this algorithm is that it is not scan resistant. Scan resistance refers to pollution of the cache when there is continuous memory access of single use memory blocks whose size is higher than the cache size [16]. This is one of the most popular algorithms used.

b) First In First Out (FIFO): In this policy, the cache acts as a queue. The data blocks in the cache are evicted in the order in which they were populated when the cache is full. Most important feature of this policy is that it is very fast with very little overhead. The drawbacks of this policy: it is non-adaptive in nature and is not scan resistant.

c) Least Frequently Used (LFU): In this policy, the cache keeps a counter of the number of times a data block is accessed. Hence the data blocks with lower count value are discarded first. The important features of this policy: adaptive in nature and is very fast. The drawback of this feature is that it is not scan resistant.

d) Most Recently Used (MRU): It is exactly opposite to that of LRU, but it comes with an overhead of keeping track of the most recently used data blocks. An advantage of this feature is that the policy is fast and is scan resistant, unlike LRU. But the drawback of this policy is that it is not adaptive in nature.

e) Random: It is a unique policy where the data blocks in the cache are discarded at random. It is very fast and has very little overhead in comparison to other protocols. But the most important drawback being that it is non-adaptive in nature.

In [24], Adams and Agesen have compared virtualization in both software and hardware environments. Since Memory Managed Unit (MMU) was not supported in the hardware, they look at a technique known as Binary translation in the kernel code for the guest VMs. This technique offers better performance in terms of memory tracing which involves caching. In this technique, the binary translator captures the execution trace of the guest Virtual machine code. This ensures the translator cache code will provide good cache locality when the code execution follows similar paths.

In [25], the authors look at the overheads that come along with the use of virtualization with high performance networks in multi-core platforms. Two of the most important problems that hinder the performance are the added domain to process I/O and the additional scheduler which is a part of the VMM for scheduler domains. The authors propose an improvement to caching procedure in the virtualized world, where the last level of cache of DomU is shared with the Dom0. The authors are of the opinion that this technique will improve the cache locality while accessing packets and also lower inter-domain communication costs. The above techniques along with certain hardware-related changes result in a better performance of virtualized networks in high performance networks.

Kourai, K., in his research in [26], suggests the use of a warm-cache reboot at the VMM level in order to improve the performance immediately after a reboot. After a reboot, the cache is wiped out and hence the performance of the VM goes down. In order to avoid this, the author has developed "Cachemind" on top of the XEN hypervisor. The I/O between the DomU and the Dom0 is handled by the VMM. The warm-cache reboot mechanism uses the Physical-to-Machine (P2M) mapping table, bitmaps (reuse), and cache mapping tables. All the memory contents from the DomU are stored in the P2M, through Cachemind, even during the reboot. The cache-mapping table is used to restore the cache to DomU once the reboot is complete and to maintain cache consistency the bitmaps are reused. This mechanism gives a boost to our thesis by bringing in another advantage of quick cache re-use even after the reboot, if the cache is maintained in the Hypervisor or the VMM. Hence this particular research does prove the importance of having the cache at the Hypervisor level, which is the basis of our thesis as well.

Khalid and Obaidat, in their work [27], suggest a new cache replacement algorithm which provides better performance, compared to the popular LRU, MRU, and FIFO policies.

This policy is known as the Khalid Obaidat Replacement Algorithm (KORA), and works on the principle of identifying and discarding dead line in the cache. The key to their algorithm is to look at the history of memory address accesses without impacting the performance of the cache or consuming additional time and space. This work proves a very important point of considering the history of memory access to improve the hit ratio in the caching mechanism. Our work in this thesis also takes access history into account to better the performance of the cache in comparison to the traditional LRU mechanism.

In [28], the authors propose an enhancement to the existing Re-reference Interval Prediction (RRIP) policy [29]. The RRIP policy was proposed in order to avoid cache pollution when memory blocks are referenced with the distant re-reference interval. The LRU algorithm is not scan resistant, where the cache is polluted if the memory accesses are not repetitive in nature for a continuous stretch for more than the size of the cache. In such a case the cache will lose all of its important data since it does not consider recency in the picture. Hence, RRIP was proposed as a solution to this problem. In spite of the above advantage, RRIP limits the performance of memory accesses which are recency-oriented. If there are several memory accesses with the same distant recency, then the RRIP selects the block to be evicted randomly. This might cause a recently accessed memory block to be evicted. To overcome this issue, the authors propose a strategy which considers both re-reference prediction value as well as access recency value in the picture during the eviction process. This policy is termed as the Dynamic Adaptive Insertion and Reference Prediction (DAI-RRP). The key point to be noted here is that the access history and access recency are to be considered if we have to improve the performance of the caching process. The core idea of our thesis of using access history to predict and evaluate the cache replacement algorithm reaps us good benefits.

CHAPTER 4 – TEST SETUP, SIMULATION AND RESULTS

4.1 Goals

- Simulate a hypervisor cache scenario using software.
- Come up with a probability-based replacement policy for the hypervisor cache.
- Compare the results of the performance of the new cache replacement policy with the traditional and widely used LRU algorithm.
- Use different workloads to test the flexibility of this cache.

4.2 Test Setup and Simulation

The testbed used is a software simulation of an actual cache and is programmed using the language C. The program consists of the four following steps in order to simulate the cache [23].

a) Process the arguments passed and check to see if they meet the requirements.

b) Build the cache based on the input values provided.

c) Take in reference inputs, and simulate cache.

d) Print the results and the analysis.

4.2.1 Workloads used for testing

To prove the flexibility of the algorithm, different workloads were used to test the cache.

- Array Sort: A C program which performs the sorting of an array provided as an input in ascending order. This involves lot of swapping operations and thus would be an ideal candidate to be used in benchmarking for caching operations.
- Fast Fourier Transform (FFT): A C program which performs FFT; FFT is an algorithm used to reduce the number of computations using the discrete Fourier transform algorithm.

- GZIP: The GNU zip, is a utility used for compression which generates a compressed file with an extension ".gz"
- Mp3: The last workload which is used to perform testing is the reading of a music file with the extension ".mp3". This is a continuous stream of music that is read from the main memory and involves considerable of I/O operation.

Each of these operations is executed on the following hardware:

- Virtual Box Hypervisor
- CentOs Linux Virtual Machine
- RAM assigned to the VM – 512MB

The Cache simulation program accepts the memory traces and simulates the cache based on these memory traces. So, in order to obtain the memory traces of the above programs, we use the Valgrind and Lackey benchmarking tool [31]. This tool takes a program command line execution as the input, executes the program, and provides the memory accesses during the execution of this program as a separate file. This file contains all the load and store operation values, and the load and store correspond to read and write operations in caching terminology. In this thesis we only concentrate on read values, hence a perl script is used to obtain only the load values from the log file output of the Valgrind-Lackey tool. This is then provided as an input into the cache simulation program. The cache simulation program is run for different cache sizes - 2KB, 4KB, 8KB, 16KB, 32KB and 64KB and different associativity – 2, 4, 8 and 16.

To realize the benefits of the Probability based cache replacement algorithm, we use the widely used and popular LRU replacement algorithm as a benchmark. The test is run once using the LRU algorithm, and once with the Probability-based algorithm; the miss ratio for these

algorithms is plotted to analyze the results. The proposed probability-based algorithm uses the recent history of access to calculate which block of data needs to be evicted during the process of cache replacement. In this algorithm, the current probability of access is compared to the overall past access probability maintained by the access history buffer in order to decide the block of data to be evicted. This is an improvement to the traditional LRU-based algorithm which processes the cache replacement on the basis of the "least recently used block to be evicted" method.

4.3 The Pseudo-code for the Probability based Algorithm

```
probAlgorithm() {
        probIndex = get the current size of the memory
        BlockQ *blockBase = get all the blocks of the temporary memory
         iterate through the complete memory
         while (blockIp) {
                 i++;
                 total number of hits for all the blocks = number of hits for each of the blocks or
        the block in question + total number of hits for all the blocks  ;
                blockIp = move to the next block;
         }
          Average value of the memory being hit or accessed = total number of hits for all the
        blocks / (1.6% of the memory size allocated);
          return the average value
 }
int GetLRUWay(int index) {
```

```
        BlockQ *blockBase = GetSetBlocks(index);
        BlockQ *blockIp = blockBase;
        int i = 0, way = 0, idea = 0;
        int probAverage = probAlgorithm();
       A probability value of this block being hit  = Number of times this block is been hit/
(1.2% of the memory size allocated);
        int min = 1000;
   while (blockIp) {
      if (((blockIp->timeStamp) < min && (A probability value of this block being hit <
Average value of the memory being hits or accessed )) {
        min = blockIp->timeStamp;
        way = i;
        dummyMethod3 (blockIp->index, blockIp->tag);
      }
     i++;
     blockIp = blockIp->next; \\(Move to next block of address)
}
        return way;
}
```

Int main () \\ Main Cache function starts here

if (number of blocks >= cache size) {

GetLRUWay(index of the block) \\ To make the decision of which block needs to evicted from the cache

} Else {

Continue to populate the cache

}

4.4 Graphs

Plot of varying associativity

The graph below (Figure 8) is a plot of miss ratios observed by varying associativity for different workloads, namely Array Sort, GZIP, FFT, and MP3. This is plotted for both the Probability based Cache replacement algorithm (indicated in the legend with dotted/dashed lines) and the LRU based cache replacement algorithm (indicated in the legend with solid lines)

The observations from the above graph are delineated below:

- Array Sort: At an associativity of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 13.11 where as the LRU based algorithm indicates a miss percentage of 17.37. This accounts for a 4.25% improvement due to the new algorithm.
- FFT: At an associativity of 4, the Probability-based cache replacement algorithm indicates a miss percentage of 6.45 whereas the LRU based algorithm indicates a miss percentage of 7.6. This accounts for a 1.15% improvement due to the new algorithm.
- GZIP: At an associativity of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 12.2 whereas the LRU based algorithm indicates a miss percentage of 14.0. This accounts for a 1.72% improvement due to the new algorithm.
- MP3: At an associativity of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 18.3 whereas the LRU based algorithm indicates a miss percentage of 20.55. This accounts for a 2.15% improvement due to the new algorithm.

Overall, the Probability based Algorithm provides improvement at associativity 2 and 4.

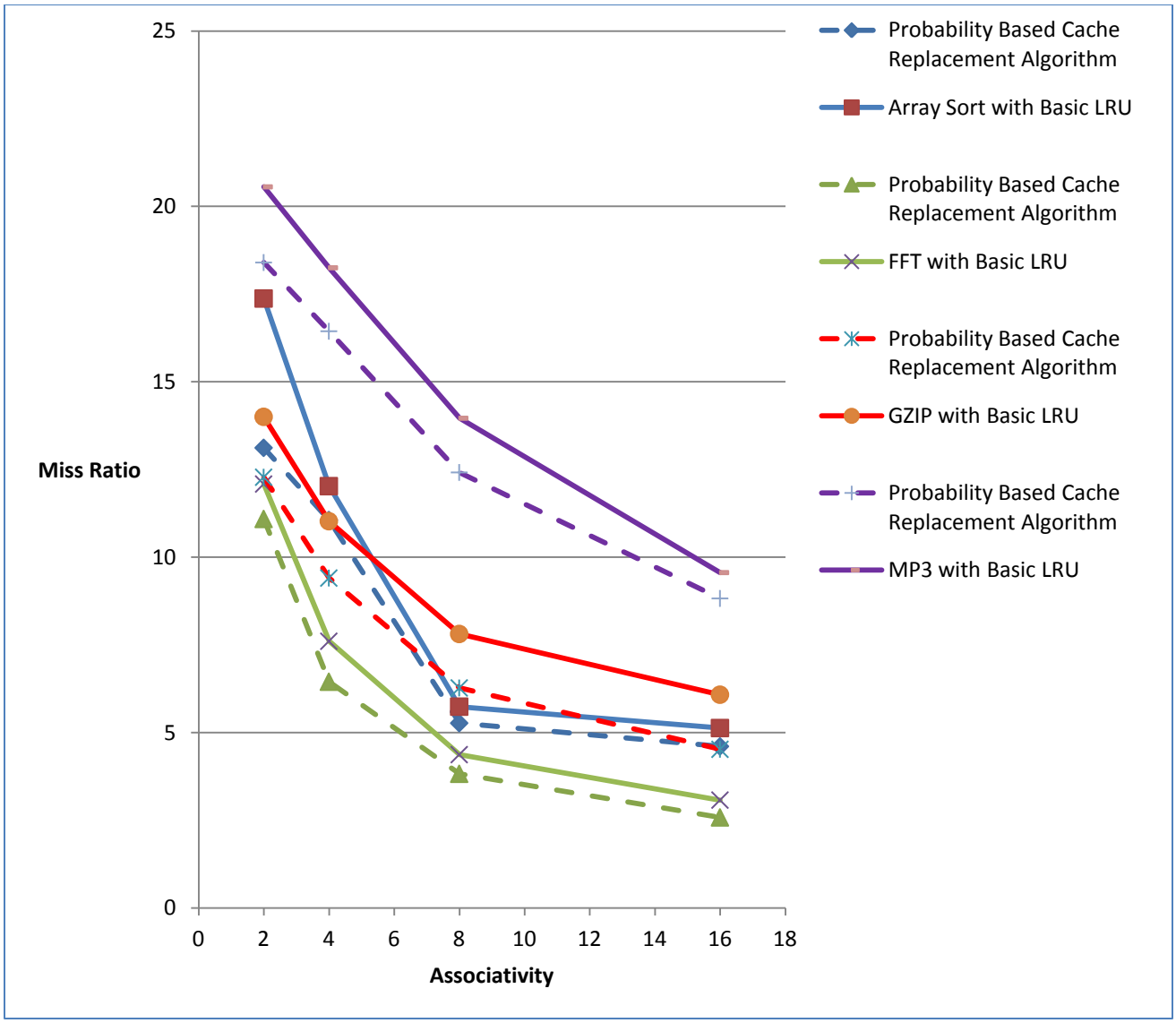

Figure 8: Graph showing Varying Associativity with "Probability Based Cache Replacement Algorithm" and "Basic LRU" against Miss Percentage

Plot of Varying Cache Size:

The graph below (Figure 9) is the plot of miss ratios observed by varying cache size for different workloads, namely Array Sort, GZIP, FFT and MP3. This is plotted for both the Probability-based Cache replacement algorithm (indicated in the legend with dotted/dashed lines) and the LRU based cache replacement algorithm (indicated in the legend with solid lines)

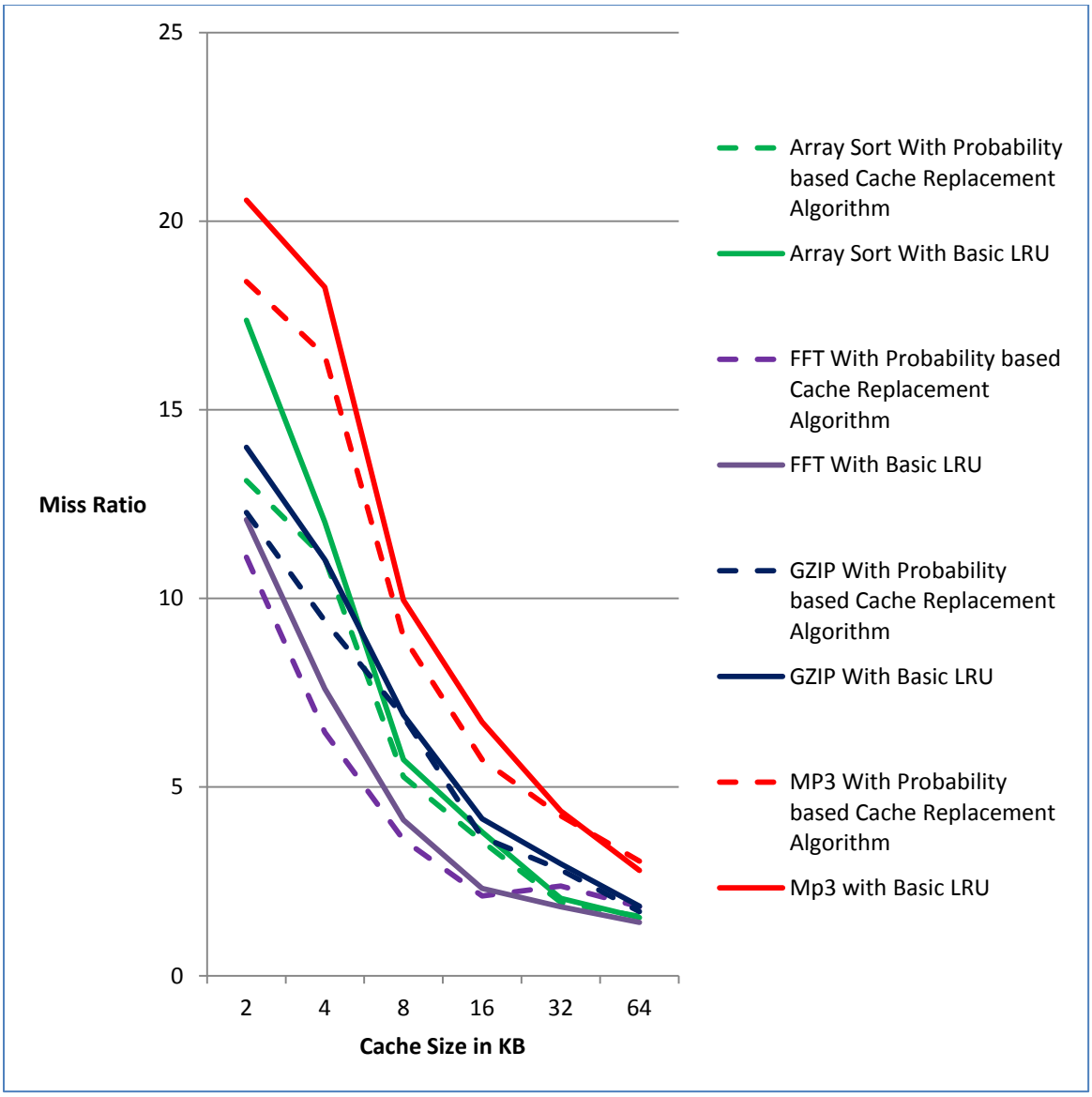

Figure 9: Graph showing Varying Cache Size with "Probability Based Cache Replacement Algorithm" and "Basic LRU" against Miss Ratio

The observations from the above graph are as below:

- Array Sort: At a cache size of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 13.11 whereas the LRU based algorithm indicates a miss percentage of 17.37. This accounts for a 4.25% improvement due to the new algorithm.

- FFT: At a cache size of 4, the Probability-based cache replacement algorithm indicates a miss percentage of 6.45 whereas the LRU-based algorithm indicates a miss percentage of 7.6. This accounts for a 1.15% improvement due to the new algorithm.
- GZIP: At a cache size of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 12.2 whereas the LRU based algorithm indicates a miss percentage of 14.0. This accounts for a 1.72% improvement due to the new algorithm.
- MP3: At a cache size of 2, the Probability-based cache replacement algorithm indicates a miss percentage of 18.3 whereas the LRU based algorithm indicates a miss percentage of 20.55. This accounts for a 2.15% improvement due to the new algorithm.

CHAPTER 5: CONCLUSION AND FUTURE WORK

5.1 Conclusion

Caching at the Hypervisor level gives many advantages such as more address space for caching, and avoiding duplicate memory traces across multiple VM's. We also have to keep in mind that the LRU algorithm which is generally used for caching is not the most ideal candidate, as it is not scan resistant, which leads to a need for better replacement algorithm. The above implementation indicates that cache replacement need to be based on the history of access and replacement policies should be based on probability. The implementation shows improvement in miss percentages when the new Cache replacement policy is used. The Probability-based cache replacement algorithm is scan resistant unlike the existing LRU. Hence significant changes are to be brought in to the existing LRU replacement policies to bring about performance improvement in caching.

5.2 Future Direction

This implementation considers only read values as input to the cache simulator. To obtain more accurate values, we can also consider the write and instruction memory traces along with the read traces. This will ascertain the benefits of using Probability based cache replacement algorithm. This can also be implemented as a part of hypervisor which can be built from the kernel up. An ideal choice for the same can be the Xen Hypervisor.

REFERENCES

REFERENCES

[1]What is storage area Network
http://www.snia.org/education/storage_networking_primer/san/what_san [Cited Aug/2011]

[2]Demystifying Storage Networking
http://www-03.ibm.com/industries/ca/en/education/k12/technical/whitepapers/storagenetworking.pdf [Cited Aug/2011]

[3] A Storage Architecture Guide http://www.storagesearch.com/auspex-art-2001.pdf [Cited Sep/2011]

[4] Storage Area Network Architectures http://pmcs.com/whitepaper-processor-mips-sonet-ethernet/ [Cited Sep/2011]

[5] The Role of Network Storage in DB2 Warehousing
http://www.information-management.com/issues/20021101/5955-1.html [Cited Dec/2011]

[6] Storage Virtualization for Dummies
http://www.hds.com/at/go/virtualisierung/download/hds_storage_virtualization_for_dummies.pdf [Cited Sep/2011]

[7] Storage Virtualization http://www.snia.org/sites/default/files/sniavirt.pdf [Cited Aug/2011]

[8] Wikibon Virtual Infrastructure Adoption Projections http://wikibon.org/blog/20-key-research-notes-from-the-wikibon-community/ [Cited Nov/2011]

[9]How Virtualization Works and Its effect on IT
http://garraux.net/files/cit112_virtualization.pdf [Cited Sep/2011]

[10] Understanding Full, Para and Hardware Assisted Virtualization
http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf [Cited Aug/2011]

[11] XenServer http://community.citrix.com/display/adi/XenServer [Cited Sep/2011]

[12] Virtual Machine Security Guidelines
http://benchmarks.cisecurity.org/tools2/vm/CIS_VM_Benchmark_v1.0.pdf [Cited Jan/2012]

[13] Hypervisor http://searchservervirtualization.techtarget.com/definition/hypervisor [Cited Oct/2011]

[14] Essentials of Computer Architecture, Comer. Douglas.

[15] The Basics of Caches http://cseweb.ucsd.edu/classes/su07/cse141/cache-handout.pdf [Cited Nov 2011]

[16] Dueling CLOCK: Adaptive Cache Replacement Policy based on The CLOCK Algorithm http://www.date-conference.com/proceedings/PAPERS/2010/DATE10/PDFFILES/07.7_2.PDF [Cited Dec/2011]

[17] Virtual Memory Access Tracing with Hypervisor Exclusive Cache http://static.usenix.org/event/usenix07/tech/full_papers/lu/lu.pdf [Cited Dec/2011]

[18] Stephen T. Jones, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, "Geiger: monitoring the buffer cache in a virtual machine environment", *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems.*

[19] Memory Resource Management in a VMWare ESX Server http://www.vmware.com/pdf/usenix_resource_mgmt.pdf [Cited Sep/2011]

[20] When Virtual is better than real http://www.cs.toronto.edu/~demke/OS_Reading_Grp/pre-s2002/chen_virtual_hotos8.pdf [Cited Nov/2011]

[21] Disco: Running commodity Operating Systems on Scalable Multiprocessors http://zoo.cs.yale.edu/classes/cs422/2011/bib/bugnion97disco.pdf [Cited Jan/2012]

[22] Informed Prefetching and Caching http://www.pdl.cmu.edu/PDL-FTP/TIP/SOSP15.pdf [Cited Nov/2011]

[23] Building a One-Level Cache simulator with C http://www.keyvan.ms/building-a-onelevel-cpu-cache-simulator-with-c [Cited Sep/2011]

[24] Adam. K., Agesen. O,. "A Comparison of Software and Hardware Techniques for x86 Virtualization"

[25] Liao.G., Guo.D., Bhuyan.L., King.S., "Software Techniques to Improve Virtualized I/O Performance on Multi-Core Systems"

[26] Kourai .K., "Fast and Correct Performance Recovery of Operating Systems Using a Virtual Machine Monitor"

[27] Khalid, H., Obaidat, M.S., "Performance evaluation of a new cache replacement scheme using SPEC, " *Computers and Communications, 1996., Conference Proceedings of the 1996 IEEE Fifteenth Annual International Phoenix Conference on* , vol., no., pp.144-150, 27-29 Mar 1996

[28] Xi Zhang, Chongmin Li; Haixia Wang; Dongsheng Wang; , "A Cache Replacement Policy Using Adaptive Insertion and Re-reference Prediction," *Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on* , vol., no., pp.95-102, 27-30 Oct. 2010

[29] A. Jaleel, K.B. Theobald, S.C.S. Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," *The 36th Intl. Symp. on Computer Architecture, France*, pp.60–71, June 2010

[30] The GZIP file format http://www.gzip.org/format.txt [Cited Nov/2011]

[31] Valgrind's Tool Suite http://valgrind.org/info/tools.html [Cited Nov/2011]