ENHANCEMENT OF CACHE UTILIZATION BY ISOLATION AND THIN PROVISIONING OF HYPERVISOR CACHE

A Thesis by

Karthik Mylar Balasubramanya

B.E.,Visveswaraiah Technological University, India, 2008

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

ENHANCEMENT OF CACHE UTILIZATION BY CACHE ISOLATION AND THIN PROVISIONING OF 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 Electrical Engineering.

_____
Ravi Pendse, Committee Chair

_____
Abu Asaduzzaman, Committee Member

_____
Janet Twomey, Committee Member

DEDICATION

TO LORD GANESHA

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

ACKNOWLEDGEMENTS

I sincerely thank my thesis advisor, Dr. Ravi 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 gave me a level of discretion in deciding my area of research. I would like to thank members of the committee for their effort and time in reviewing this thesis.

I extend my gratitude towards Amarnath Jasti, Vijay Ragothaman and Nagaraja Thantry for sharing their valuable technical know-how with me.

I would like to thank all my friends who have helped me in the smallest of way to sail through the tides of thesis and reach the shores of Master's.

ABSTRACT

Storage resources are being consolidated and have led to the increased demand in sharing storage, depending on the type of application or class of customers. In such a scenario, cache consolidation becomes increasingly necessary. An analysis of cache utilization at various levels in a Storage Area Network such as the server cache, the virtual machine cache, and the array controller cache was done, drawing conclusions from the response time and execution time. A proposal is made for dynamic cache allocation algorithm in a virtualization environment, which takes into account the usage trends and varies the allocation depending on the type of the workload.

The dynamic cache allocation algorithm helps in adapting the cache space available for a particular virtual machine based on the number of requests and the cache miss affected, with respect to a threshold value which is individual to each VM. The caching mechanism proposed in this work is an isolated, unshared, dynamic cache, where caching is done on the hypervisor instead of the virtual machines. Also, we prove by experimental results that a static allocation of the cache is not suitable for a varying workload in a virtualized setup.

TABLE OF CONTENTS

CHAPTER 1 – INTRODUCTION

1.1 Overview

The ever-growing trend of Information Technology has called for a whole new way of data storage and management. The constant increase in the amount of data that an organization procures and maintains has led to a great deal of innovation, which has mainly come in the form of storage area networks (SAN) [1] and virtualization [2]. A large part of the investments and advancements are being made in these mentioned areas. While virtualization has made it possible to consolidate many individual servers into one single physical machine, SAN have allowed the storage to be remote and more organized, thereby servicing the consolidated servers with a lesser overhead. Thus, only the execution part of the operation will be local while the computation will be remote. This helps in achieving greater efficiency and performance, with a minimal use of hardware. The most significant aspect of this approach is the reduced carbon footprint.

With the advent of SAN and virtualization, every aspect of storage gains significance, of which cache is of prime importance. Caching is employed at various levels along the path of the server to storage. It can be local, like in a server level caching, or at a remote level, like in a array controller. Since many servers are consolidated, the placement and allocation of the cache is extremely important. The cache being an expensive resource, care should be taken for judicious and efficient usage.

1.2 Storage

The perception of data over the last decade has changed completely. It has become instrumental in deciding the fate of a business in terms of access. There has been an overwhelming increase in the storage requirements due to the globalization of business, for

which internet has become the key. There are minimal windows that are available for data backup and recovery which has made it necessary for facilitating cost-effective and efficient ways for storage management. Traditional storage poses the following challenges:

- Data sharing across a large number of users simultaneously across the globe.
- Support and deployment for data-intensive vital applications across a network which is ever increasing.
- Effective and efficient management of stored data without a large expenditure of money.

The following approaches can be considered to address the above mentioned issues:

- Direct Attached Storage (DAS)
- Network Attached Storage (NAS)
- Storage Area Network (SAN)

DAS has a setup configured so that the storage devices are connected directly to the host computers. The storage to computer processor communication happens through protocols such as the Fiber Channel (FC) and the Small Computer System Interface (SCSI). This setup is similar to our laptops or desktop computers where in the storage is connected to the processor using a SATA bus.

NAS has its protocols, like NFS and CIFS, which work in tandem with channel protocols like SCSI. That is, the traditional protocol stack is used for communication between the host and the remote storage. Therefore the NAS file system is exposed to issues such as file security, file integrity, and consistency.

1.2.1 Storage Area Network (SAN)

SAN can be defined as “Any high performance network that enables communication between storage and computer systems” per Storage Networking Industry Association (SNIA) [3]. Storage devices are managed and accessed by a network. The servers use protocols such as Fiber channel (FC) or iSCSI to access the storage devices.

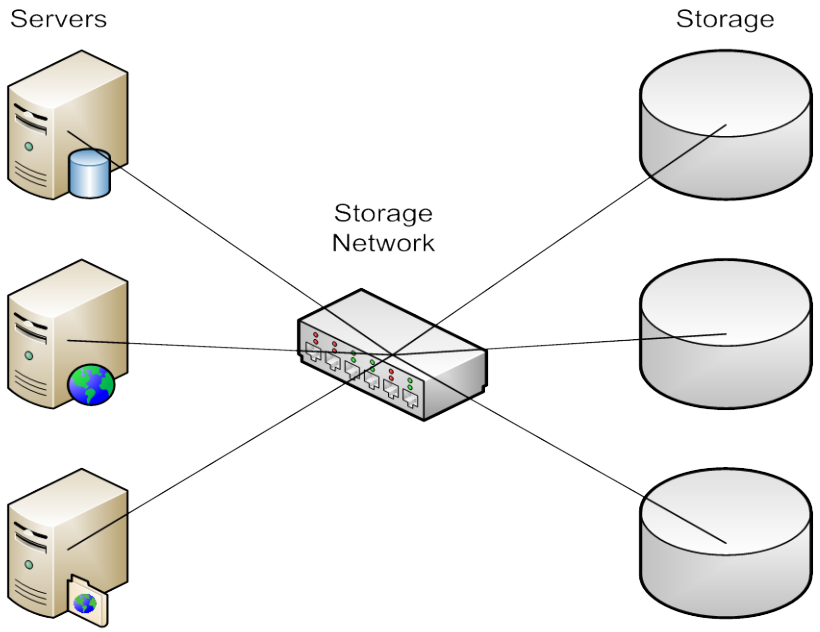

Figure 1. Typical SAN setup

SAN has the following advantages [4]:

1. ***Availability***: The same storage can be accessed over multiple paths in a SAN. This negates the case of a link down scenario, wherein if a path fails the server can choose to communicate through other available paths.
2. ***Consolidation***: SAN helps to consolidate the storage into shared and highly available resources.

3. ***Scalability***: The separation of storage from server makes the SAN environment more scalable where in newer storage devices can be made a part of the network and can have connectivity to the server seamlessly.
4. ***Improved Management***: The consolidation of storage brings about an overall improvement in the storage management. Since the storage and the server are separate, the bandwidth can be divided among them optimally. This also helps in reducing management costs.
5. ***Bandwidth***: There can be an effective use of bandwidth and the servers can perform faster as they are not tied to the storage devices. This helps in achieving a quicker response time for requests.

1.3 Virtualization

The varied scope of virtualization makes it hard to define it in a sentence. In simple words virtualization can be thought of a technology used to create virtual copies of the existing physical hardware or a fully functional software environment, be it a storage device or an Operating System (OS). The virtual machine created makes use of the existing hardware to make it appear like a multiple logical resource. Therefore, many applications which might require independent servers can be emulated by Virtual Machines using a single host machine.

Virtualization presents the following advantages [5]:

1. ***Server Consolidation***: Virtualization allows multiple server applications to run on a single physical machine. So, a number of physical machines running different operating systems can now be under a single platform; this might lead to the physical host having

more processor speed and memory, but will use less power than the individual hosts. This greatly improves the utilities costs and real estate expenditure.

2. ***Testing and Development***: A particular application can be isolated and tested in a controlled environment, leading to lesser recovery times and helping us overcome severe crashes.
3. ***Load balancing***: Virtualization helps in resource balancing by moving the virtual machines that over utilize the server application to under-utilized servers. This ensures an optimal utilization of the server resources.
4. ***Reliability and Security***: Crashes on the system due to corruption of memory which might be caused by device drivers can be negated. Also, I/O resource isolation improves security and availability of a particular application.

Virtualization comes in various flavors:

1. ***Server Virtualization***: In this type, the users do not have access to the resources. In a way it is hid from them. Users will not have any information on how the resource management is being done. This type of virtualization helps in increasing the sharing of resources efficiently. There are different forms of server virtualization:
    - OS virtualization
    - Para-virtualization
    - Hardware emulation

    In OS virtualization, a operating system is run on top of an existing OS. The applications that are being run on the hosted OS do not have access to other applications running on another virtual OS.

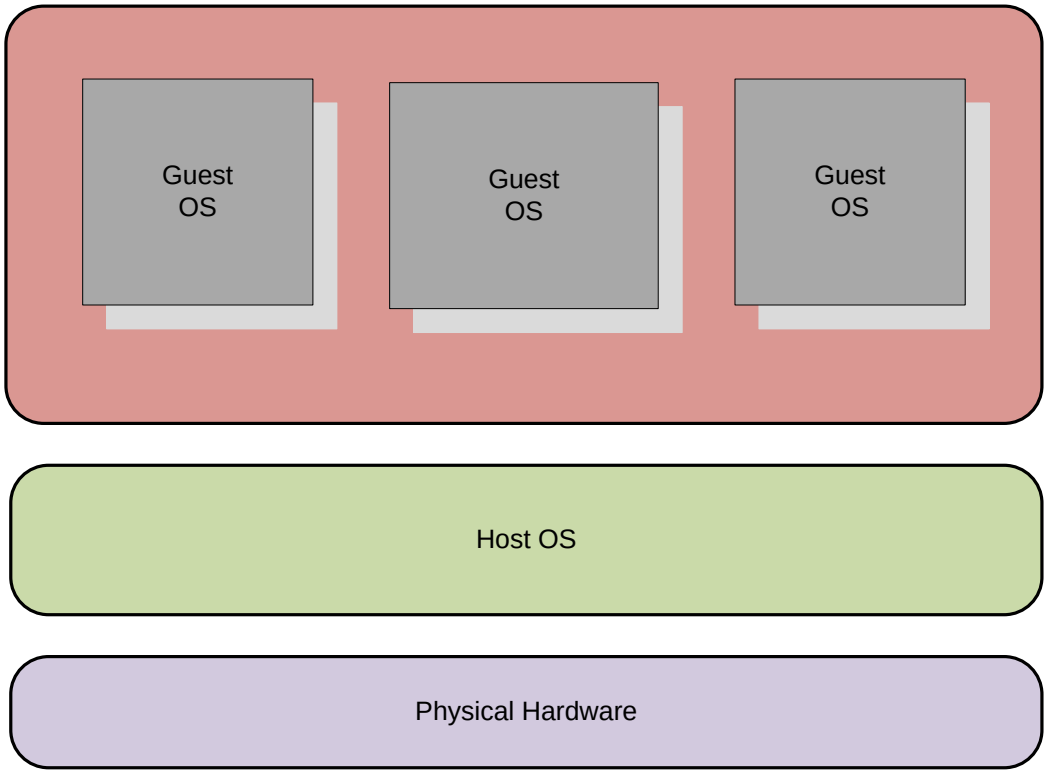

Figure 2. OS virtualization

Para-virtualization has a software interface between the existing physical hardware and the virtual machine. The guest VMs will be aware of the fact that they are running in a virtualized environment.

Hardware emulation occurs where in the virtual machine monitor hosts the guest OS. The VMs and the VMM as a whole can be migrated from one physical host to another. For this the entire physical resources are virtualized by the VMM, and therefore giving an impression to the guest OS that it is running on a standalone host.

2. ***Full virtualization***: In this type, the entire hardware is simulated and presented to the virtualized environment. In this case too, the VM will not be aware that it is running in a virtual setup. Since the users are isolated this type of virtualization is more secure.

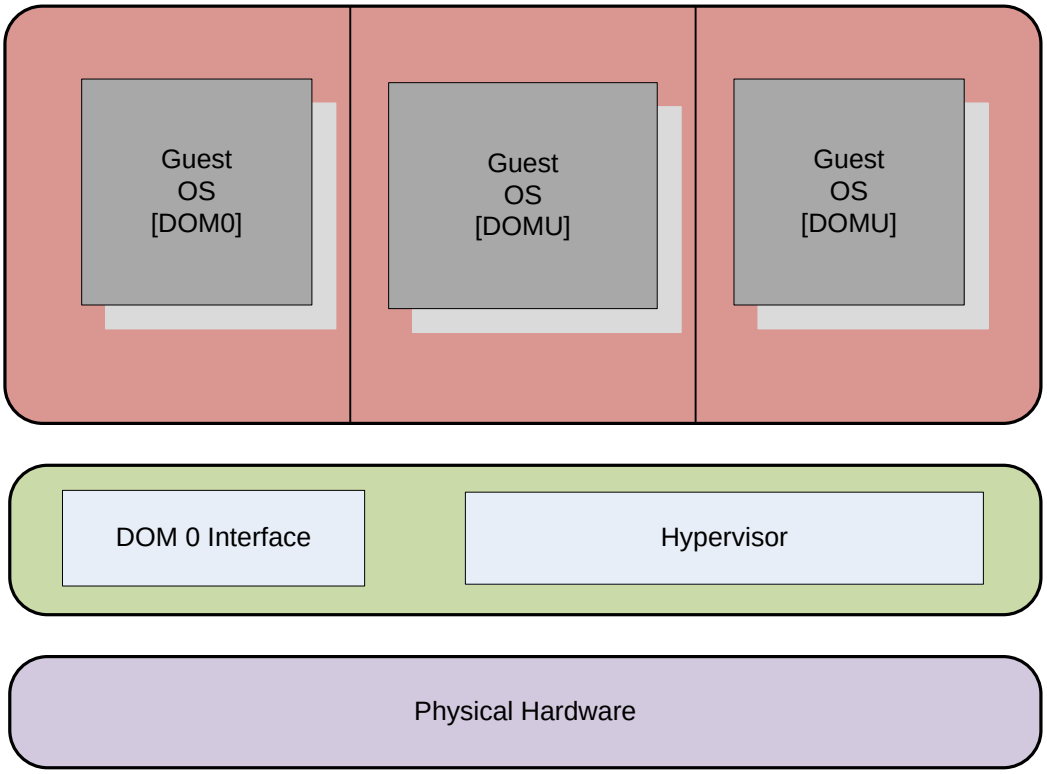

Figure 3. Paravirtualization

CHAPTER 2 – CACHE

2.1 Introduction to the Cache

Small high speed memories that are placed between the processor and the main memory to speed up the memory accesses are referred to as *Caches*. The access time for a cache is much less than is the access time required to access the main memory. Caches are by default very small and can hold only a small part of data contained in the main memory. Therefore if a memory access is made, the cache is first consulted to check whether or not it contains the required data. Otherwise an access to the main memory is made. If the data cannot be found in the cache, it is referred to as a *miss.* If it is found in the cache, then it is referred to as a *cache hit* and the main memory need not be consulted. In case of a cache miss, data has to be fetched from the main memory. During this process of data fetching, a copy of the block containing the accessed data is made in the cache as well, presuming that the data might be required again in the future. The cache performance is measured in terms of the cache hit ratio, which is the total number of hits divided by the total number of accessions.

Some of the parameters that have to be considered while designing a cache are:

- The cache capacity, which is the total number of bytes that the cache can hold.
- Associativity, which is an indicator of the number of locations in a cache that a particular block can be loaded. An associativity of one signifies the fact that the block can be stored in a single location. A cache is said to be fully associative if it has the ability to store the blocks in any location of the cache.
- Block size, the number of bytes that can be loaded from and written to the memory.

Generally, more than one cache is typically placed between the processor and the main memory. The smallest and the fastest cache will be nearest to the processor and the slowest will be nearer to the main memory, thus forming a hierarchy. The associativity of these caches varies, depending on the requirements.

Cache optimization should inculcate the following aspects: [6]

- The probability of finding the referenced memory target in the cache should be maximized.
- The access time for a cache hit should be minimized.
- The overhead that is involved in updating the main memory should be minimized.
- The delay that occurs due to a cache miss has to be minimized.

Cache hit ratios depend on a program's stream of memory references. If a particular program exhibits temporal locality, then it is most likely that the accessed data might be required again. In case of spatial locality, the data might not be required in the future at all, in which case caching that data would be a waste.

2.2 3C model for Cache Misses

A cache miss can occur due to the following: [7]

- Cold misses: This is when the memory block is accessed for the first time.
- Capacity misses: If the cache capacity is lesser than the amount of data accessed it leads to a capacity miss.
- Conflict misses: If a set which is resolved by the reference address is full it causes a conflict miss.

2.3 Caching in Virtual Machines

When virtual machines are created on an existing physical host, the resources are assigned to each virtual machine appropriately. The physical memory, the primary function of which is caching, is also assigned to a virtual machine. The caching policies are decided by the operating system that is running on a particular VM. The hypervisor which is used to create the virtual machines is aware of all the requests made by a VM referencing a memory location. It will also act as a resource manager for the virtual machines. How the hypervisor manages the cache is beyond the scope of this research.

In this thesis we are proposing a hypervisor level cache which is dynamically allocated among the virtual machines based on the needs of the individual VMs. The hypervisor keeps track of all the requests that are being made by a particular VM and knows if it is deprived of cache or if it has cache in excess which other VMs might need.

CHAPTER 3 – LITERATURE SURVEY

3.1 Related Work

The memory allocation policies in technologies like VMware, [8][9] Xen, [10] and VirtualBox [11] partition the physical memory among the different VMs and allow performance isolation. The VM data access pattern at the hypervisor level is very challenging to determine due to the lack of knowledge about the whole process.

Goyal *et al.* [12] propose a dynamic caching algorithm called CacheCOW which dynamically allocates the cache space among different workloads running on the servers. The algorithm proposed by the authors provides a storage system cache level QoS and also increases the throughput by increasing the cache hit rate. The cache is allocated dynamically based on the observed response time, temporal locality of reference and arrival pattern for each class.

P. Lu and K. Shen [13] employ a hypervisor level exclusive cache which allows transparent tracing of data access that would assist in determining the VM page miss ratio. They come up with a solution to manage large chunks of VM memory without incurring an increase in the overall count of page faults. The authors propose a technique that the hypervisor uses to manage the part of the VM memory so that all accesses that miss the remaining part of the VM memory can be traced by the hypervisor. In this mechanism, the hypervisor manages its memory for exclusive caching, i.e., the pages that are evicted from the VMs are cached. The pages entering from the VM and not from the secondary storage are cached. They use the ghost buffer technique to predict the page miss rate on VMs where the memory is beyond its current allocation.

O. Tickoo *et al.* [14] have identified the challenges required to model a virtual machine in a data center in order to study the performance of the VM and the VMM. The authors show in their work that the VM performance depends heavily on visible and invisible resource interference caused by other VMs. Their study shows that the performance of the VMs decreases by 30% when invisible resources such as caches are shared by the VMs and the cache resource of the VMs are invisible to the VMM. Hence, the effects of invisible resources need to be obtained by using performance counters.

. Stephen T. Jones *et al.* [15] have explored techniques to infer the cache and memory usage in VMs by Virtual Machine Monitors (VMMs). They have developed a prototype implementation of these techniques in XEN VMM, which they call “Geiger”. Geiger is used to accurately determine when pages are inserted into and evicted from a system’s buffer cache. Geiger output can then be used to make decisions about memory allocations to the VMs and it also enables a VMM to implement useful VMM level services. Specifically, the authors show that, using Geiger, the VMM can observe the guest OS interactions with virtual hardware such as the MMU and storage devices and also detect when the pages are inserted and evicted from the operating system buffer cache.

In [16], the author studies various caching schemes on the KVM hypervisor and discusses the advantages and disadvantages of double caching in virtualized environments. The paper proposes solutions to overcome the disadvantages of caching on both the hypervisor and virtual machines. The author first proposes a mixed caching approach where caching is pushed on the hypervisor side and the cache on the VMs is monitored and shrunk frequently. Another algorithm is proposed wherein a ballooning driver is cooperatively used to control the page cache.

Haogang Chen *et al.* [17] have identified memory as a bottleneck of application performance in virtualized setup. The authors propose a scheme called REMOCA (Hypervisor Remote Disk Cache) for memory intensive or I/O intensive workloads. In their scheme, a virtual machine can use the memory resources on other physical machines for caching purposes. The main goal of the research is to reduce the number of disk accesses, thereby reducing the average disk I/O latency. The authors have implemented REMOCA within the hypervisor by intercepting guest events such as page evictions and disk accesses. The authors state in their research that, with an increase in the number of VMs and demands of applications, the main memory will become an extremely limited resource and it will be very critical to improve the memory resource efficiency in virtual machine systems.

R. Prabhakar and co [18] propose the dynamic but shared cache management scheme *Maxperf,* which works in a multi-server architecture in managing the cache that is being allocated to the different servers. Applications that are concurrently running are assessed and the cache required is continuously calculated.

This thesis, proposes to eliminate caching on the virtual machines, and instead implement an unshared and dynamic cache on the hypervisor. A dynamic cache allocation algorithm based on a thin provisioned model is implemented on the proposed hypervisor cache which results in efficient cache utilization and a reduction in the response time of the virtual server.

CHAPTER 4 – DYNAMIC CACHE ALLOCATION MODEL

This section describes the model for dynamically allocating the cache resources among the VMs. As explained earlier, the hypervisor manages the cache and is the central entity which decides cache allocation to the VMs. The model and the algorithm used by the hypervisor to decide the cache allocation is as explained below. Table I represents the parameters used in the model. The following parameters are used as indicators to allocate and de-allocate hypervisor cache to the VMs dynamically:

1) The number of IO requests a VM receives in a particular time interval, $2[t\ ,\ t+\ \Delta t]$

2) The Cache Miss Percentage of a VM in the time interval, $2[t\ ,\ t+\ \Delta t]$

TABLE I
PARAMETERS USED IN THE MODEL

Parameter	Description
C	Total available cache size on the hypervisor
N	Number of active Virtual Machines
$\mathbf{C_{Alloc}}$	Cache allocated to each VM
$\mathbf{C_{Threshold}}$	Threshold cache usage for each VM
$\mathbf{R_{Threshold}}$	Threshold of number of I/O requests on the VM
$\mathbf{C_{MissTh}}$	Threshold of number of cache misses for the VM
$\mathbf{R_n}$	Number of I/O requests on the VM in $[t, t+\ \Delta t]$
$\mathbf{C_{Used}}$	Amount of cache used by a VM in $[t\ ,\ t+\ \Delta t]$
$\mathbf{C_{Miss}}$	Number of cache misses on the VM in $[t, t+\ \Delta t]$
$\mathbf{C_{Remain}}$	Amount of cache remaining on each VM

Initially, the total available cache *C* is divided and equally allocated to each of the active virtual servers. The hypervisor owns a small portion of the cache space for itself and is not allocated to the VMs [3]. If the portion of cache owned by the hypervisor is represented by C_h, then the available cache to allocate to the VMs can be given by:

$$C_{vm} = C - C_h$$

The cache space initially allocated to the VMs is given by:

$$C_{alloc} = \frac{C_{vm}}{N}$$

At any time instant 't', the cache remaining on the virtual server, C_{Remain} , can be given by:

$$C_{Remain} = C_{alloc} - C_{Used}$$

The model developed in this research is based on the following assumptions:

1) Each virtual machine plays the role of a server. Example, a VM can be a database server, a web server and so on.

2) The workload characteristics of each VM are known a priori.

3) The values of $C_{Threshold}$, $R_{Threshold}$ and C_{MissTh} are calculated when the virtual server has heavy workloads running on it. Hence, these values are taken as a reference to make decisions about allocating and de-allocating cache space.

The dynamic caching algorithm proposed in the research is based on a thin provisioning technique. A virtual server which owns excess cache in addition to the threshold, in time interval $2[t\ ,\ t + \Delta t]$ acts as a cache donor and a virtual server which has less cache than the threshold borrows the cache space from the donor. Two consecutive time intervals are considered in order to get reliable values, since readings in a single time interval could be misleading. The virtual server cache allocation and de-allocation are decided based on the following conditions in time interval $2[t\ ,\ t + \Delta t]$:

- If $R_n > R_{Threshold}$ and $C_{Miss} > C_{MissTh}$ then cache needs to be allocated to the virtual server and the virtual server acts as a borrower only if $C_{Used} > C_{Threshold}$for two consecutive time intervals $2[t, t + \Delta t]$

- If $R_n < R_{Threshold}$ and $C_{Miss} < C_{MissTh}$ then, cache can be de-allocated from the virtual server and the VM acts as a donor only if $C_{Remain} > 2C_{Threshold}$.

- If $R_n \geq R_{Threshold}$ and $C_{Miss} \leq C_{MissTh}$ then no action will be taken because it indicates that the virtual server still has cache remaining

- If $R_n < R_{Threshold}$ and $C_{Miss} > C_{MissTh}$ then, the cache replacement policy has to be looked into, because this scenario is independent of cache space.

Based on the above conditions, if the algorithm decides that cache needs to be allocated to a virtual server, then,

The pseudo code for dynamic caching using thin provisioning is as shown below.

Main()

{

Get the threshold cache value = $C_{Threshold}$;

Get the threshold request count = $R_{Threshold}$;

Get the threshold cache miss value = C_{MissTh};

{

***Initial_Cache_Alloc**(C, N, C_h)*

{

$C_{vm} = C - C_h$ *;*

$C_{alloc} = \frac{C_{vm}}{N}$*;*

}

For *every virtual server*

{

For *time interval* $2[t\ ,\ t + \Delta t]$

{

Cache_alloc*()*

{

If *(* $R_n > R_{Threshold}$ ***&&*** $C_{Miss} > C_{MissTh}$ *)*

{

If *(* $C_{Used} > C_{Threshold}$ *)*

{

CacheAdd()

}

}

}

}

}

Cache_DeAlloc(C_{alloc}, C_{Used})

{

If *(* $R_n < R_{Threshold}$ *&&* $C_{Miss} < C_{MissTh}$ *)*

{

$C_{Remain} = C_{alloc} - C_{Used}$;

If *(* $C_{Remain} > 2C_{Threshold}$ *)*

de-allocate $C_{Threshold}$*;*

}

}

}

CacheAdd*()*

{

Compute highest value from Cache_DeAlloc()

}

}}

CHAPTER 5 - SIMULATION AND RESULTS

5.1 Test Setup

The model developed in this research was used to evaluate the performance of virtual servers when dynamic caching using thin provisioning is used. The model was evaluated both with without dynamic caching, and the experimental results show that dynamic caching using thin provisioning reduces the response time of the server and also leads to efficient cache utilization. In order to evaluate the dynamic cache allocation model, a virtual box hypervisor was installed on a host PC and 3 virtual servers were created. The specifications of the host PC are as follows:

- Dell PowerEdge 2600
- 3.20GHz processor
- 4GB RAM
- Windows 2003(OS) and will be the Host for the three Virtual Machines

The host is connected to a storage array through a Fiber Channel connection. The storage array is managed from the host machine through the software called SANtricity. Disk volumes with sizes of our choice can be created and managed on the storage array through SANtricity.

The specifications of the three virtual machines created are as follows:

- 20GB of assigned hard disk from the storage array.
- 1 GB of RAM allocated for each VM

- Windows Vista Professional OS

Each of the VMs act as dedicated servers. The servers in consideration are the File server, the Web server and the Database server. FileZilla software has been used in both the server and the client for the file server VM. MySQL software was used as the database management tool on the database server. Due to the constraints of creating a web server for a localized environment, the authors had to resort to IOmeter software to generate the required load for a web server.

The reasons for considering these servers as candidates for the experiments are that the workloads on them depict varying percentages of reads and writes and different levels of cache utilization. Hence, the proposed model can be better analyzed for different workloads. Cache performance of each VM is measured for different amounts of loads and parameters. Generally caching is done at various levels, be it the processor cache, RAM or the database buffer. For the sake of experiments in this research, the authors consider caching on the VMs and see how the performance varies for different servers.

Windows Task Manager and a performance monitor are the tools being used for the measurement of cache utilization by a process at any given point in time. The cache utilization and response time for the servers are recorded for the three servers. Cache performance for a File server is measured while there is a file transaction happening between the server and the client. Similarly, for a database server, the cache performance is measured when the server is processing database queries for different request sizes of data. For a web server, the cache performance is measured for different IOmeter loads. The test files that are being used are of uniform sizes across the 3 servers.

The requests from each of the servers go through the hypervisor through which the cache is controlled. The algorithm proposed in this research works on controlling the dynamic provisioning of the cache while maintaining the optimum performance metrics which are pre-determined and are as per the industry standards. The proposed model is evaluated for the workloads on different servers and the response time and cache utilization are evaluated. The results are promising and are evaluated for each parameter separately as explained below.

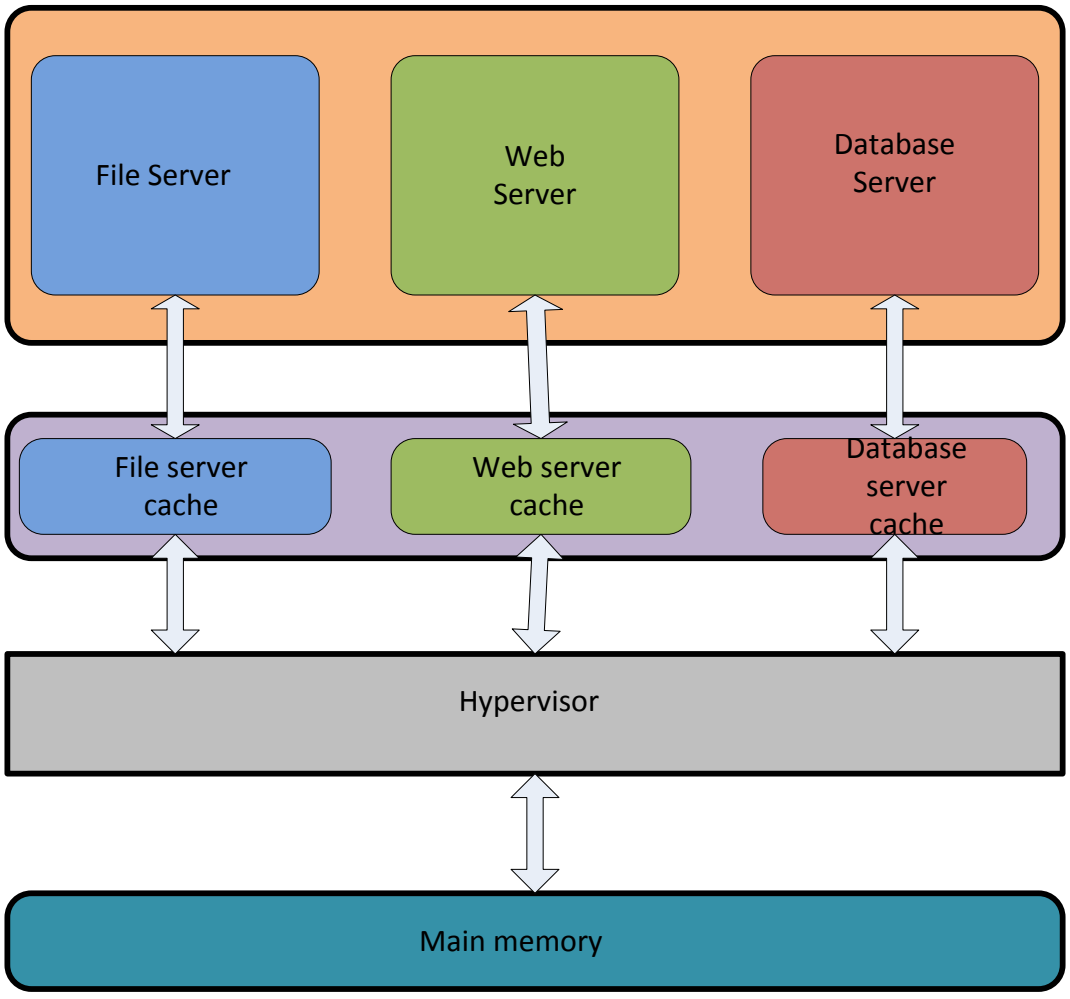

Figure 4. Testbed

5.2 Results

5.2.1 Cache Performance analysis at storage array

The read percentage was varied in steps of 25 from 0 to 100 on the IOmeter. Also, the read block sizes that are being considered are 4K, 16K and 32K. The cache hit percentage was gathered for each reading using the SANtricity software where in the simulations was run for 5 minutes for every reading. Later the cache block size was varied from the default 4K to 16K for the concerned volumes using SANtricity software.

Graphs for Cache hit vs. Read percentage were obtained for the VM and the Host for the different cache block sizes.

The average read response time reading from the IOmeter for every simulation signifies the latency involved for every read operation. This response time was plotted against the read percentage to get the latency graphs for the VM and the host for different sizes of read blocks and cache block.

We compare the differences in the cache performance for the native and virtual machines with respect to the cache variation in the array controller level in a SAN environment.

Cache Hit vs. Read Percentage

Figures 5 and 6 show how the cache hit percentage varies as the read percentage is varied. Although the pattern looks similar for both the host and the VM, it can be seen that for a read block size of 4K, the cache hit percentage is significantly more for a cache size of 16K rather than 4K. This can be explained by the fact that for a cache size of 16K, 4K blocks of data have a lesser probability of being swapped in and out of the cache and hence more data can be cached,

increasing the hit percentage. Whereas, for read block sizes of 16K or greater, swap in and swap out probability is more so the cache hit percentage does not have any significant difference when compared to that of a cache size of 4K. It can also be seen that the cache hit percentage for the VM is slightly lower than that of the native machine. Thus, it can be seen that there is a scope for improvement in the way cache allocation takes place at the VM level.

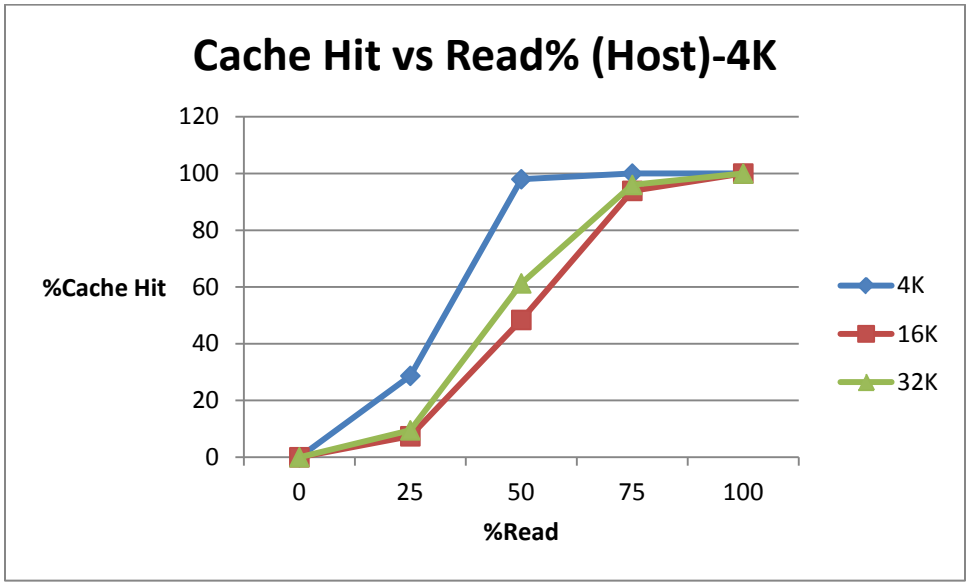

Figure 5(a) – Comparison of cache hit percentages for different read blocks of data with varying read percentages on the host machine for cache block size of 4K

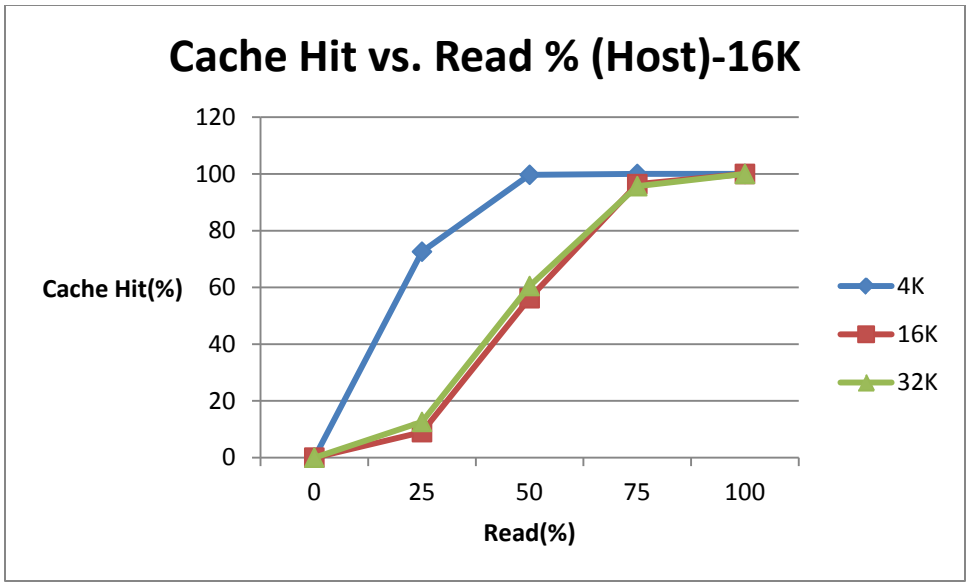

Figure 5(b) – Comparison of cache hit percentages for different read blocks of data with varying read percentages on the host machine for cache block size of 16K

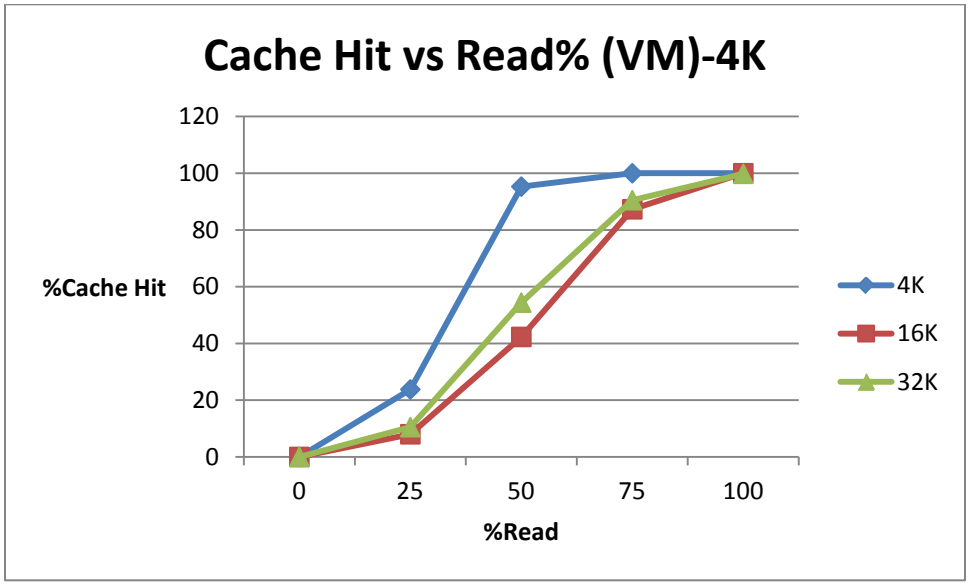

Figure 6(a) – Comparison of cache hit percentages for different read blocks of data with varying read percentages on the virtual machine for cache block size of 4K

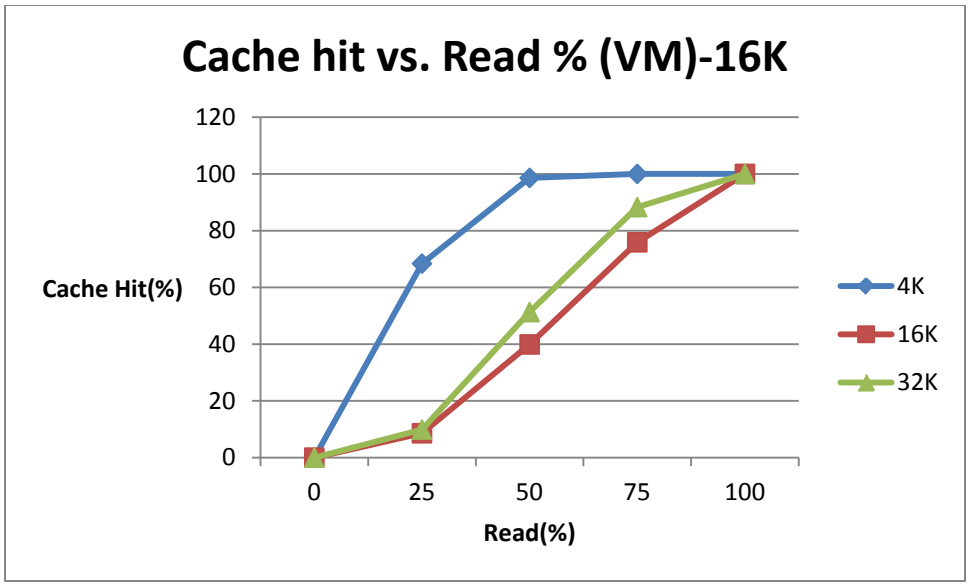

Figure 6(b) – Comparison of cache hit percentages for different read blocks of data with varying read percentages on the virtual machine for cache block size of 16K

Latency

Figures 7 and 8 compare the latency that is seen on the host and the VM when a particular data is being read from the storage. It can be seen that the pattern is same for both the host and the VM. But the latency when the data is being accessed from the VM is significantly higher when compared to the host. Also, latency reaches its peak for all blocks of data when the read percentage is lesser than the written percentage. This shows that a lot of additional overhead is added for data access from the VM. The latency does not vary much, even though the cache block size at the array controller level has been varied, which shows us that the read response time is independent of the cache block size. This signifies that there is a great deal of processing that takes place at the VM level, of which VM caching plays a prime role. Hence, eliminating the caching at the VM level and employing it at the hypervisor will lead to the decrease in this overhead and help in achieving better response times.

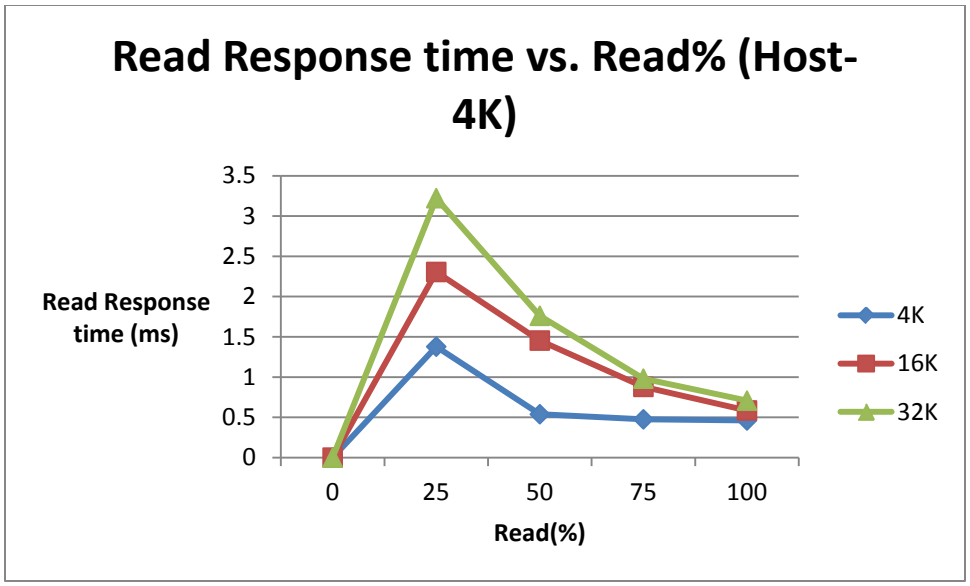

Figure 7(a) – Comparison of read response times for different blocks of data with varying read percentages on the host machine for cache block size of 4K

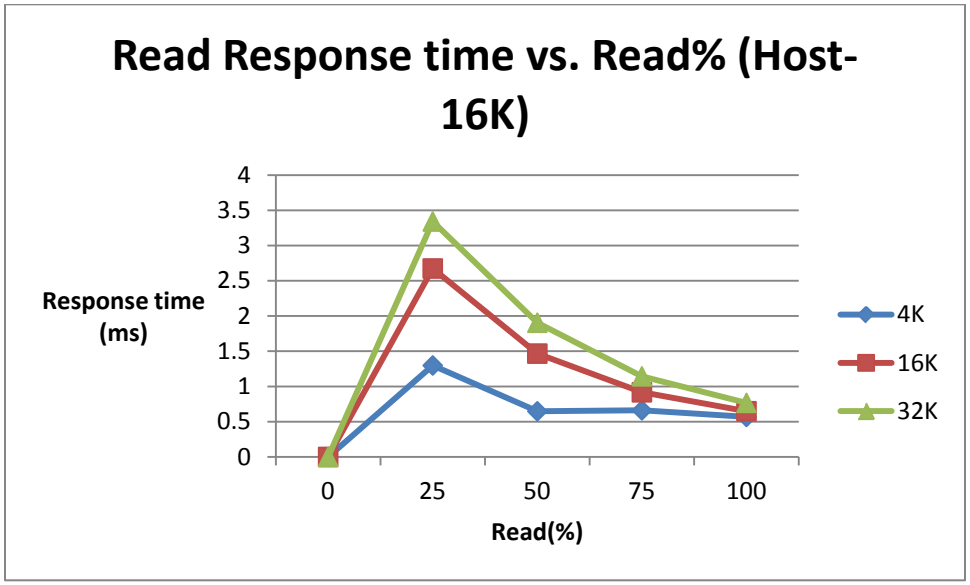

Figure 7(b) – Comparison of read response times for different blocks of data with varying read percentages on the host machine for cache block size of 16K

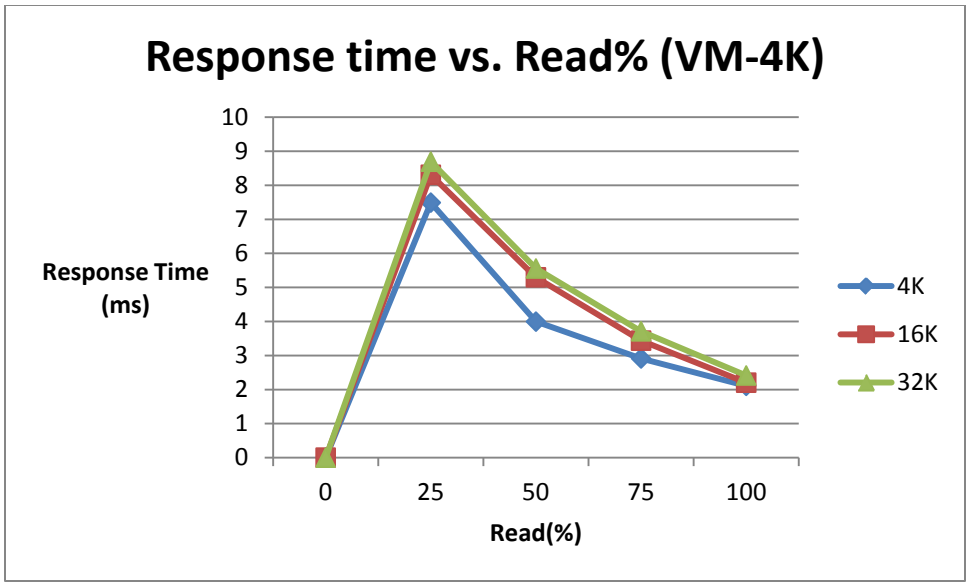

Figure 8(a) – Comparison of read response times for different blocks of data with varying read percentages on the virtual machine for cache block size of 4K

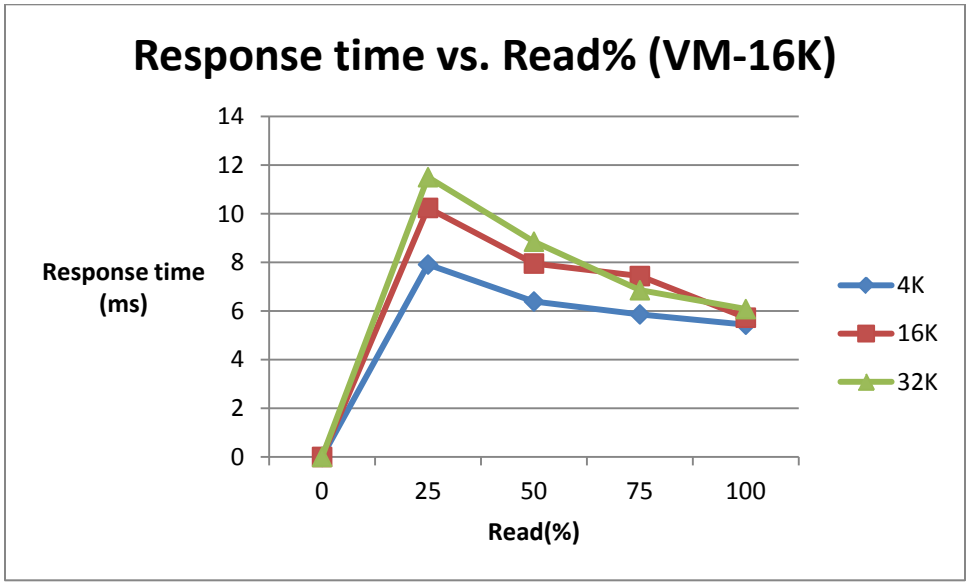

Figure 8(b) – Comparison of read response times for different blocks of data with varying read percentages on the virtual machine for cache block size of 16K

5.2.2 Evaluation of Cache Utilization with respect to File Size

The thin provision model is designed to dynamically allocate the cache, and use the cache efficiently and effectively among the VMs. The caching policy is not considered in this aspect of the calculation. Most caches usually have the LRU cache replacement policy in place. For the first part of the experimentation, the case of static cache allocation is considered. In the static allocation technique, every VM gets a definite and fixed amount of cache, irrespective of its workload. If the cache becomes full, evictions start taking place based on the cache replacement policy and new blocks are brought into the cache.

Figure 9 shows the utilization of the cache as the request block size varies. Eventually the cache in all three servers reaches its maximum limit.

The thin provisioning model was applied to the virtual servers and the cache was allocated dynamically to the VMs according to the model. With the thin provisioning model in place from figure 10, it can be observed that the cache utilization is optimal and for the same file size, the cache is not limited. Another interesting thing to observe is that there is no converging point for the thin provisioning model. This is because cache is allocated on the go, on a per-need basis and the VM does not run out of cache space quickly. Fig. 10 shows the improvement achieved by the dynamic caching algorithm.

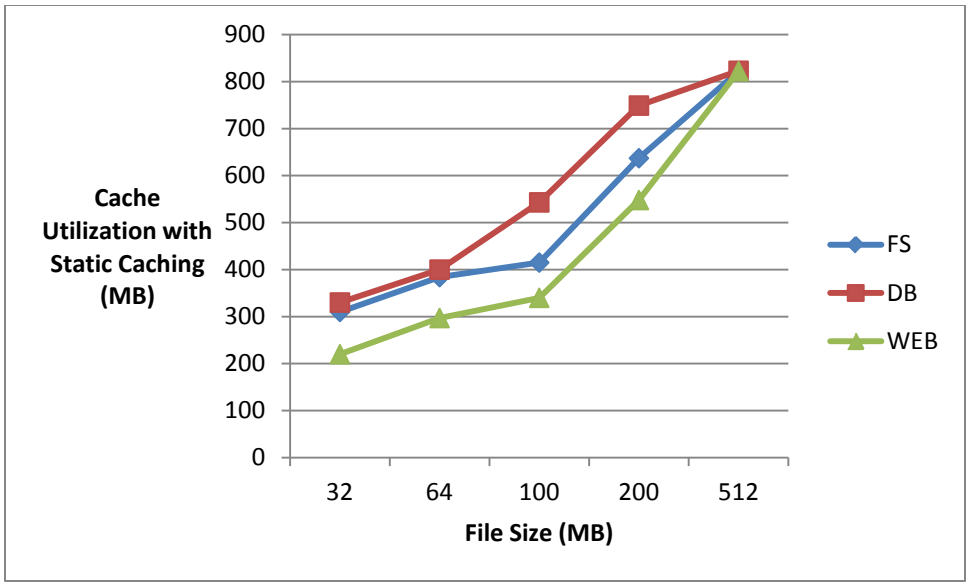

Figure 9 – Cache Utilization with static caching for varying file sizes on different servers

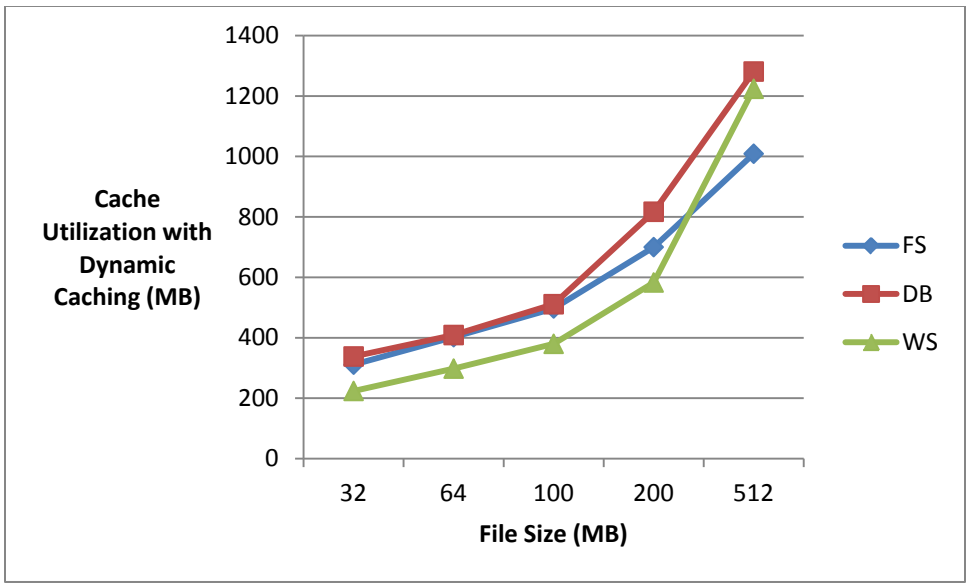

Figure 10 – Cache Utilization with dynamic caching for varying file sizes on different servers

5.2.3 Evaluation of Response Time of the Virtual Server

The response time of the virtual server can be defined as the time taken for the virtual server to service the application requests. Response time is a major factor that decides the performance of any system. A lot of parameters affect the response time, of which the cache and processor speed are the most significant ones. When cache was statically allocated, the file server took only 8.5 minutes to process the workload, whereas, the database server took 80 minutes for the same. This clearly indicates that the cache used by the file server is lying dormant while that of the database server was exhausted. Similarly while the database server was processing a smaller block of data, the file server was almost exhausted of its cache resources. Therefore, a balance has to be achieved in caching among the virtual servers, so that the overall performance of caching and the performance of the entire system improves.

Figure 11 shows that the dynamic allocation of the cache improves the response time of the server, by adequately balancing the cache among the three servers. Dynamic allocation also helps reduce the number of capacity cache misses, because cache is allocated to the VM when there is a need for it, unless there is no cache space to allocate. Hence, the chance that a VM might run out of cache quickly is greatly reduced.

Figure 11 compares the response times of the servers with statically allocated cache to those with thin provisioning. The database server with thin provisioning has shown a remarkable decrease of 25% in the response time, the file server has shown a 17% decrease and the web server has shown a 14% reduction in the response time when dynamic caching was in place.

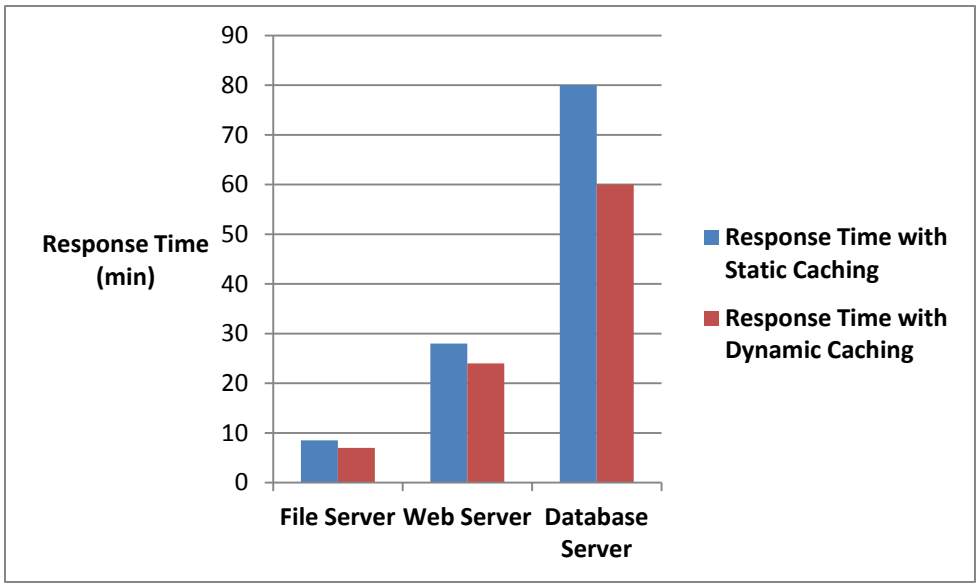

Figure 11 – Comparison of response times with static and dynamic caching as seen by the considered servers

5.2.4 Evaluation of Cache Utilization with respect to Time

The performance of the three servers considered varied considerably when they were tested with the thin provision cache allocation model. Figures 12, 13 and 14 show the cache utilization with respect to time for statically allocated and dynamically allocated caching mechanisms for the three servers. It can be seen that the file server, database server and the web server reach the maximum cache capacity much sooner in the static cache technique when compared to the dynamic allocation model. The cache does not reach a threshold in the dynamic allocation model because more cache can be allocated based on the requirements of the server and the availability of extra cache space. Hence, the dynamic caching using thin provisioning can help improve the performance of caching and the entire system.

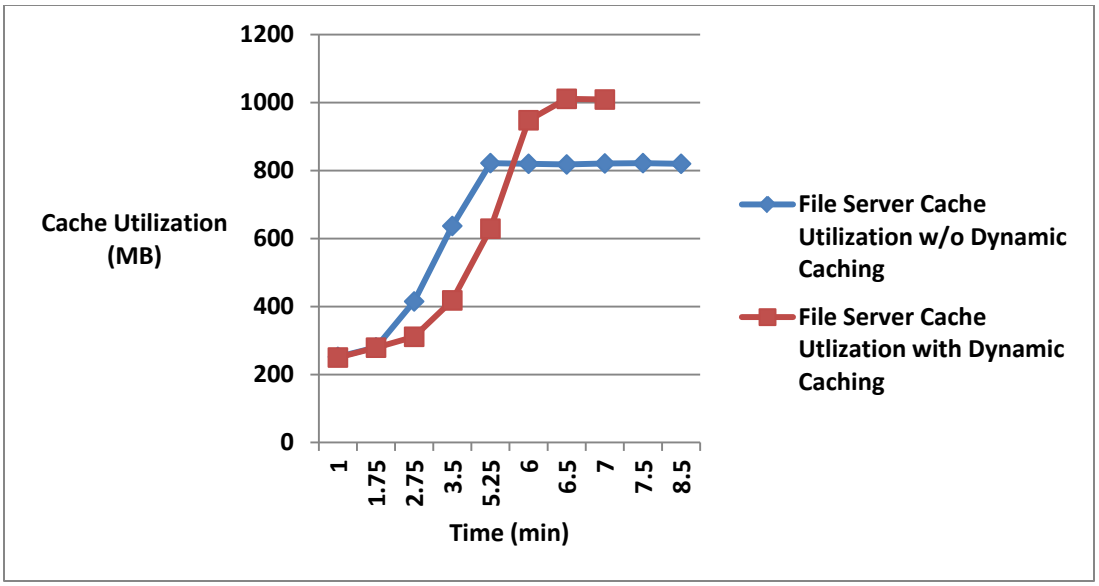

Figure 12 – Comparison of cache utilization with and without dynamic caching w.r.t time for a file server

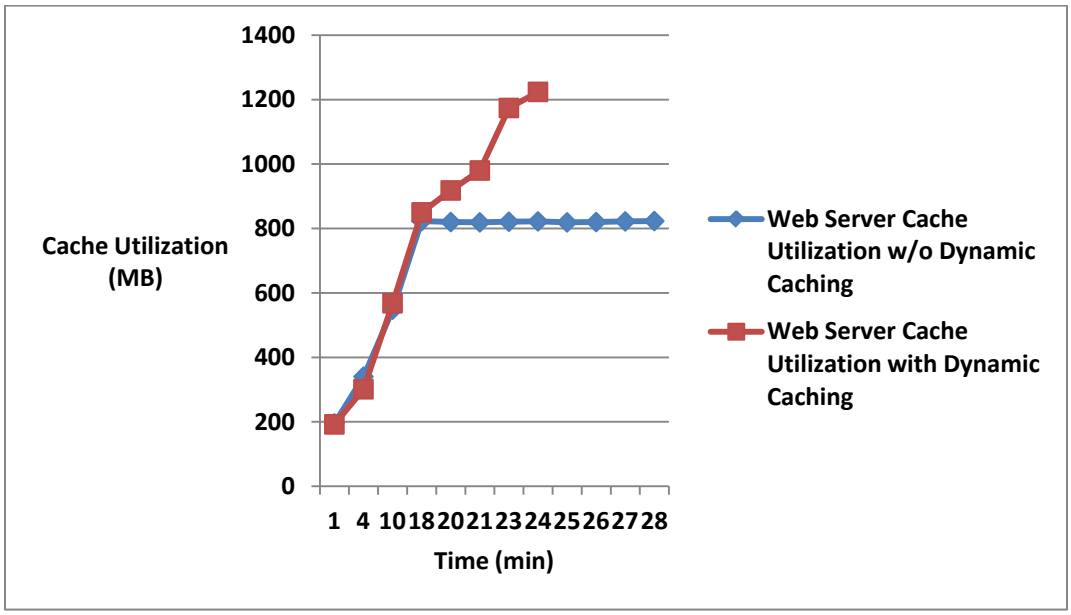

Figure 13 - Comparison of cache utilization with and without dynamic caching w.r.t time for a web server

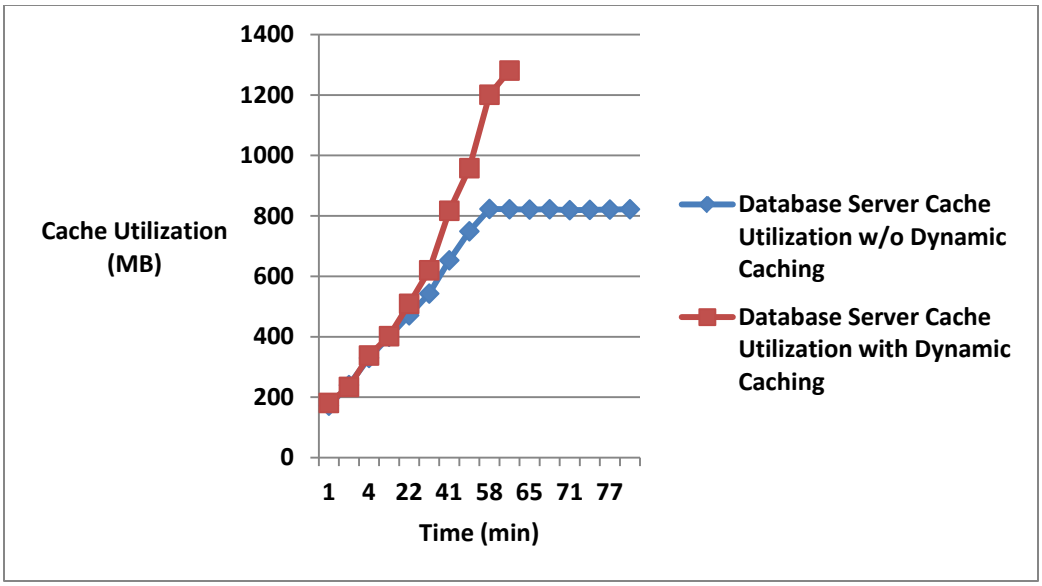

Figure 14 - Comparison of cache utilization with and without dynamic caching w.r.t time for a database server

CHAPTER 6 – CONCLUSION AND FUTURE WORK

6.1 Conclusion

A proposal to implement caching explicitly on the hypervisor and eliminate caching on the virtual servers has been made. Also, a novel cache allocation mechanism using thin provisioning was implemented, in which the hypervisor allocates cache to the VMs dynamically on a per-need basis. Compared to the static cache allocation techniques currently used, a thin provisioned cache model can efficiently utilize an expensive resource such as cache by ensuring fairness in allocation. Experimental results also show that the dynamic cache model can improve the response time of the virtual server. Thin provisioned model is capable of achieving rapid response to the changes in the workload characteristics. According to the characteristics of the thin provisioned cache model, when the cache resides in the hypervisor, it can take an intelligent call on how the cache has to be managed, depending on the workloads. This vastly helps in achieving better response times, as seen by the results.

6.2 Future Work

Since the hypervisor manages the cache of the VMs, it can result in overhead on the hypervisor. Moreover, since the whole of the cache resides on the hypervisor, it is necessary to analyze the throughput. This is left as a future work and further research is being done on the same.

REFERENCES

REFERENCES

[1] Introduction to Storage Area Networks, IBM, International Technical Support Organization, July 2006

[2] Virtualization Overview, VMware white paper http://www.vmware.com/pdf/virtualization.pdf retrieved July 2011

[3] Storage Virtualization, http://www.snia.org/sites/default/files/sniavirt.pdf retrieved October 2011

[4] Design and Implementation of a SCSI Target for Storage Area Networks, Ashish A. Palekar and Robert D. Russell, May 2001

[5] Thomas Burger, The advantages of using Virtualization Technology in the Enterprise, http://software.intel.com/en-us/articles/the-advantages-of-using- virtualization-technology-in-the-enterprise/, Aug 2010.

[6] Alan Jay Smith, Cache Memories http://www.eecs.berkeley.edu/~knight/cs267/papers/cache_memories.pdf accessed on 9/21/2011

[7] Types of cache misses: The Three C's www.cs.umd.edu/class/fall2001/cmsc411/proj01/cache/cache.pdf accessed on 9/20/2011

[8] VMware Whitepaper, "Understanding Full Virtualization, Paravirtualization and Hardware Assist", 2007.

[9] VMware Whitepaper, "VMware Infrastructure Architecture Overview", http://www.vmware.com/pdf/vi_architecture_wp.pdf, 2006.

[10] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, " XEN and the Art of Virtualization", *In Proc. 19th ACM Symposium on Operating System Principles*, 2003, pp. 164-177.

[11] Virtualbox Technical Documentation, https://www.virtualbox.org/wiki/Technical_documentation referred August 2011

[12] P. Goyal, D. Jadav, D. Modha, and R. Tewari, " CacheCOW: QoS for Storage System Caches", *In Proc. 11th Intl Conf. on Quality of Service,* 2003, pp. 498-515.

[13] P. Lu and K.Shen, " Virtual Machine Memory Access Tracing with Hypervisor Exclusive Cache ", *In Proc. USENIX Annual Technical Conference,* 2007.

[14] O. Tickoo, R. Iyer, R. Illikkal and D. Newell, " Modeling virtual machine performance: Challenges and Approaches", ACM SIGMETRICS Performance Evaluation Review, 2009, 55-60.

[15] S. T. Jones, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, "Geiger : Monitoring the buffer cache in a virtual machine environment", *In Proc. 12th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 14-24.

[16] Balbir Singh, "Page/ Slab cache control in a virtualized environment", In *Proc. Linux Symposium*, 2010.

[17] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, " XEN and the Art of Virtualization", *In Proc. 19th ACM Symposium on Operating System Principles*, 2003, pp. 164-177.

[18] R. Prabhakar, S. Srikantaiah, C. Patrick and M. Kandemir, " Dynamic Storage Cache Allocation in Multi-Server Architectures", *In Proc. Conf on High Perf. Computing Networking, Storage and Analysis*, 2009.

[19] Ismail Ari , Melanie Gottwals , Dick Henze, "SANBoost: Automated SAN-Level Caching in Storage Area Networks", in Proc. Int. Conf. Autonomic Computing, 2004

[20] Ismail Ari, Melanie Gottwals, Dick Henze, "Performance Boosting and Workload Isolation in Storage Area Networks with SANCache", *IEEE/NASA Conference on Mass Storage Systems and Technologies (MSST)*

[21] IOmeter user guide, www.iometer.org accessed June 2011

[22] VirtualBox user manual, www.virtualbox.org accessed May 2011

[23] File Caching, Microsoft Windows

[24] Mark Friedman, Odysseas Pentakalos, "File Cache Performance and Tuning"