Performance Evaluation of Split Disk Cache

by

Adarsh Jain

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 4, 2013

Keywords: Disk, Cache, Metadata, Split, Disksim

Abstract

The performance gap between the processor and the memory speeds has been ever increasing. Whereas processor speeds have improved by 60% annually, secondary storage speed has only improved by 10% annually. Although there have been many efficient techniques introduced to minimize this speed gap, it still remains a bottleneck in various commercial implementations. Since secondary memory technologies are much slower than the main memory, it is challenging to match its speed to that of the processor.

Usually, hard disk drives include semiconductor disk-caches to improve their performance. A hit in the disk-cache eliminates the mechanical seek time and rotational latency. To further improve performance a split disk-cache, which is split between metadata and data, was proposed earlier. In this thesis, we evaluate the performance of such a disk-cache via extensive simulations on DiskSim. The simulations were run on standard benchmarks, Cello and hplajw, and synthetic benchmarks using normal, Gaussian and Poisson distributions. The split point between data and metadata regions was varied to find an optimum split point. The simulation runs for the standard and synthetic benchmarks show improvement up to 6% in hit ratio when the metadata region in the disk cache is between 10-30%. Since metadata is smaller in size, but accessed frequently, such a result seems reasonable. Although the performance improvement is small, it is important, because of the high access latency at the level of the disk. As such, our raw projections show that the overall response time improvement is expected to be between 15-20%. Such performance improvement could particularly be significant for long running processes.

Acknowledgments

There are many people in Auburn who deserve my gratitude for helping me pursue my M.S dreams. Foremost among them is Dr. Sanjeev Baskiyar, who has truly been an amazing adviser. Without his timely inputs and continuous support, this thesis would never have been possible. I shall forever remain indebted to him for his guidance in my research and my career. I would also like to thank Dr. Xiao Qin and Dr. Alvin Lim for serving as members of my advisory committee. I also owe much gratitude to Dr. Daniela Marghitu, Dr. David Umphress, Dr. Jeff Ku and Dr. Kai Chang for shaping my graduate student career for the better. I would also like to acknowledge the efforts of Ms. Michelle Wheeles, Ms. Jo Lauraitis, Ms. Carol Lovvorn and Ms. Penny Christopher in helping me keep my school and immigration paper work in order. My thanks also go out to my colleagues at Shelby 3139 and Shelby 2113. In particular, I would like to thank the group of Swaroop Anupindi, Chengjun Wang, Vibudh Mishra, Samuel Haque, Matthew Swann, Joseph Ledet and Bradley Smith for suggestions and help. I am also deeply indebted to the families of Dr. Dave Sree and Mr. Nagaraj Ejantkar for ensuring that I missed none of the festivals celebrated back home. In addition to these families, I would also like to thank my brother Amith Jain, my friends Sanjay Kulkarni, Abilash Kittanna, Santosh Kulkarni, Harish Rao, Vijay Sheshadri, Harsha Banavara, Pratap Simha, Deepika Rao, Rakshith Venkatesh Prateek Hejmady, Nitilaksh Hiremath, Ajit Chavan, Vikalp Narayan, Aditya Singh, Digvijay Gholap, Poojita Puligundla, Kanika Grover and Swathi Dumpala, for all their support, laughs and companionship at Auburn. I would also like to thank Abhishek Rao, Akshatha Prasad, Roohi Sahota, Ajit Prabhu, Deep Kanwar Singh, Madhani Penathod Subair, Mahalaxmi, Vinay Kumar, Ashwini Yathiraj, Vadiraj, Harish Sathish, Rohan Prasad, Naveen KS, Karthik Ramesh, Sachin, Karthik Krishnan, Mithun Rajanna for their constant support.

Above all, I would like to express my deepest gratitude to my family for their love, compassion and support in my endeavor. It was the constant emotional support from my parents Rekha Jain and Subhash Chandra Jain which motivated me and helped me immensely in achieving this dream. Together they define my existence and it is to them that I lovingly dedicate this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the processor speeds continue to increase, the challenge to constantly supply data to the processor also continues to increase. Processors grow at a rate of 60% every year where as the memory technology grows at 10% [9]. One major bottleneck in meeting the increased processor demand is the speed at which data and instructions are fed to the processor from the storage devices. Although implementation of high-speed cache system and various other techniques have improved to fill the performance gap, secondary storage access speed is a matter of concern especially for big data. The access time for secondary devices is considerably high compared to main memory. Table 1.1 shows typical access time for various memory technologies. It is clear from the table that hard disk is 10^5 times slower than the main memory. As the gap between memory and disk drives increases to 6 orders of magnitude with the gap increasing up to 50% every year, ta number of optimization techniques is required narrow this gap. A small improvement at disk cache level would reduce the response time by a great margin [11]. Hence a minimal improvement in hit ratio for disk cache would improve the overall response time substantially. The total access time when the data is not present in the disk cache and the physical disk has to be accessed is given by Hospodor [7]

$$t_{access} = \mathrm{t}_{overhead} + \mathrm{t}_{seek} + \mathrm{t}_{rot\_latency} + \mathrm{t}_{xfer} \tag{1.1}$$

where,

- t_{access} : Total access time

Memory Device	**Access Time (seconds)**
Registers	10^{-9}
Cache	10^{-8}
Main Memory	10^{-7}
Disk Cache	10^{-5}
Hard Disk Drive	10^{-2}

Table 1.1: Access Time in seconds

- $t_{overhead}$: Time required to decode the request from the processor and identify the required data region
- t_{seek} : Head seek time
- $t_{rot\_latency}$: Rotational delay
- t_{xfer} : Transfer time

In case of a cache hit, the total access time is,

$$t_{access} = t_{overhead} + t_{xfer} \tag{1.2}$$

For simplicity, we consider the transfer time for cache to be equal to transfer time through the magnetic disk although the latter is much higher as it involves rotation of the disk to read from the sectors [7]. The access time is reduced by a factor of 100 [See Table 1.1] when there is a cache hit.

Many mechanical techniques have improved the actual response time by 8% annually over the last decade via reduction in seek time by about 8% and increase in rotational speed by nearly 9% annually. Additionally, improvement in areal density by 40% annually has resulted in reduction in response time by 8% every year. In total, there has been a 15% yearly improvement from disk technology [4] [1].

Disk cache is a buffer in the disk system that holds recently accessed portions of the disk memory. The storage system with a unified disk cache is shown Figure 1.1. In this system, file system/database cache represents the logical cache and the disk cache represents

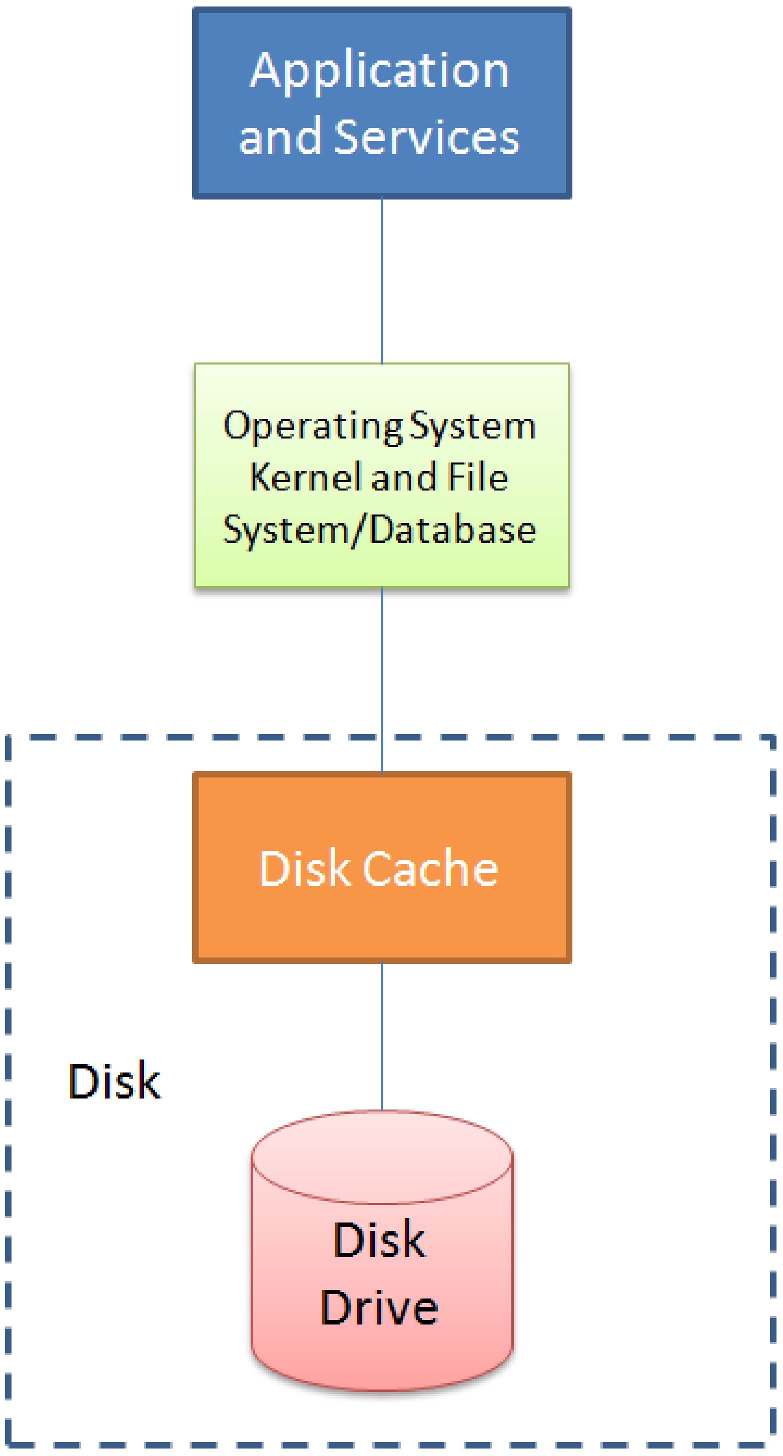

Figure 1.1: Memory Hierarchy

the physical cache. When the processor makes a request to the disk drive, the OS first checks the logical cache. A miss at the logical level cache results in an I/O request to the physical level cache. If a miss occurs at the physical level cache then the physical drive is accessed. In this research we focus on the performance of the physical level cache.[1]

In this research, we address the disk cache of the disk drive system. Every hit to a disk cache results in an I/O time that is substantially less (e.g., 1-4 ms) than would otherwise be required (10-100ms) [16]. We evaluate the performance of a disk cache split into data and metadata regions. Metadata takes only a small portion of the memory, but is accessed very frequently [5][6][2].

1.1 Motivation

New I/O access techniques have improved the efficiency of I/O operations between the processor and the storage devices. Some of the techniques include caching, write buffering, prefetching, request scheduling, parallel I/O, various bus architecture optimizations etc. These techniques have reduced the performance gap between the processor and the hard disks, but hard disk speed still lags behind the processor speed.

Hsu and Smith [9][10] have analyzed the performance of these techniques by using the physical I/Os from several real servers and PC workloads. Their investigations have led to the conclusion that maximum improvement can be obtained by reducing the number of physical I/Os through read-caching, prefetching and write buffering. Also, it has been concluded that read caching can be more effective at the physical level, if the cache size is large enough, typically 1% of the storage used. But commercial hard disks are not equipped with caches of that size. For example a 320 GB hard disk has to have 1.6 GB cache when it is half full. Such large size caches are impractical as it is very expensive. Hence in this research we split the data region of the disk cache in to two regions namely, data and metadata regions. As explained previously metadata takes only a small portion of the memory, but is accessed very frequently [5][6][2]. Hence with the split, we try to bring the metadata region close to 1% of the space occupied by metadata in the hard disk.

1.2 Problem Statement

Hard disk operation is slower compared to the speed of the processor. As a result of this gap in operational speed, processor has to wait for a long time when accessing the hard disk for data. Hence it is important to reduce this gap. Hard disk cache plays an important role closing the gap between the processor and the hard disk. Optimizations at the disk cache level is important and provides various performance benefits.

Frequency of metadata access at the hard disk level plays an important role in improving the performance of the overall system. Although various techniques have been developed to optimize the metadata access, there is still some room for performance improvement.

1.3 Contribution

Secondary storage (Hard disk drives) is an essential component of a computer system when dealing with big data sets. With the storage density bursting every year and various break through continuing in processor technology, it is very important to optimize the performance of hard disks. Through this research, we have aimed at reducing the number of I/O operations going to physical hard disk. We try to improve the hit ratio to disk cache buffer and hence reduce the gap between the processor and the hard disk drives. This can be implemented in hard disk cache for an enhanced performance.

1.4 Report Organization

This report is organized as follows. Chapter 2 presents the literature review. Chapter 3 explains the split percentage design. Chapter 4 discusses about the implementation of the split disk cache design and the evaluation methodology. In Chapter 5 we present the graphs and results. In Chapter 6 we conclude the report and discuss about future work.

Chapter 2

Literature Survey and Review

In this chapter, we take a look at various research techniques done in the area of disk cache. Also, since metadata and its access pattern is important for our research, we would discuss a few optimizations done in the field of metadata access. We start with a literature survey about disk cache and its characteristics and later introduce a few optimization techniques in disk cache. We also emphasize the importance of metadata by going through a few literature.

2.1 Disk Cache Basics

Memory hierarchy in a computer system is shown in the Figure 2.1. In this memory hierarchy, large amounts of data are stored in secondary storage systems like tapes, disks etc and the data which is in active usage is stored in a more expensive fast storage namely main memory and CPU cache memory. When dealing with huge data sets, secondary storage systems are accessed. The problem with secondary storage systems is the large access time. This in turn may keep CPU idle for sometime and hence degrade the system performance. Various techniques have been introduced to close this time gap but the access time is still a bottle neck as far as secondary storage devices are concerned. In order to overcome this bottleneck and make the processor usage more efficient, cache system is also used for secondary storage systems. Such caches are referred to as disk cache.

Disk Cache or buffer is used to hold some data portions of actual disk. Disk cache operation is much faster compared to the external disk and hence if disk cache can respond to a significant fraction of I/O operations, the this would enhance the system performance

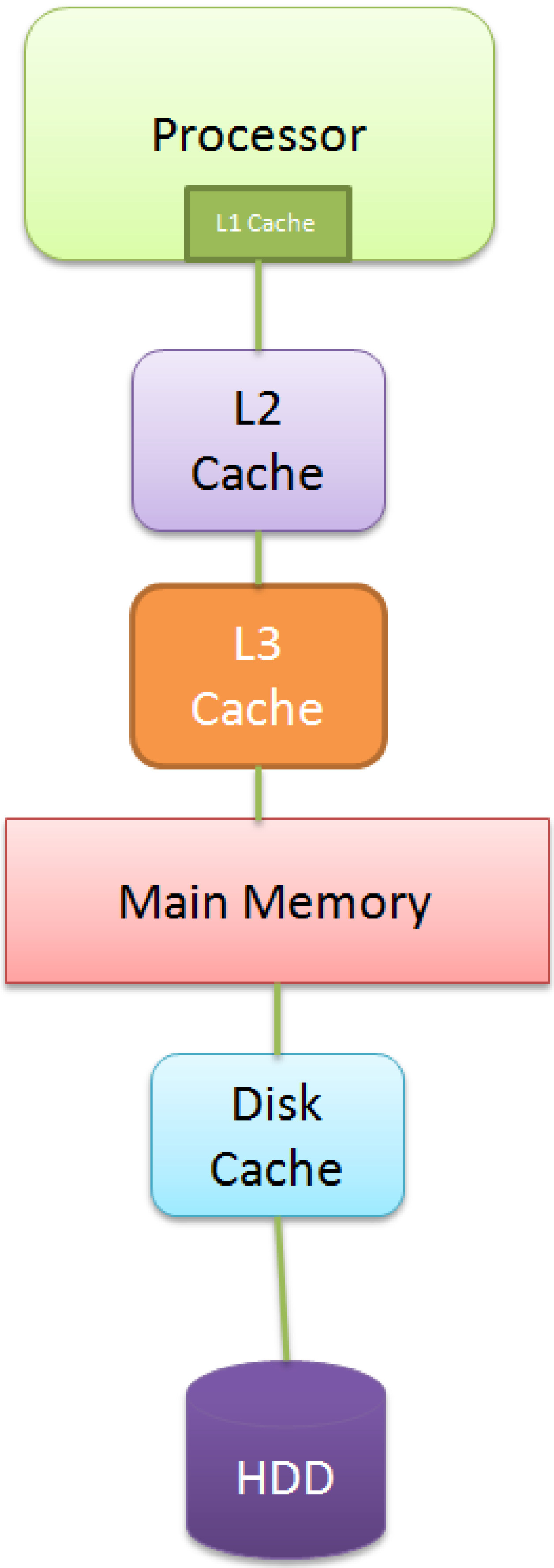

Figure 2.1: Memory Hierarchy

by a great margin. Some of the other characteristics of disk cache are that they are less expensive and have access times and transfer rates significantly lower than that of disk.

Disk Cache produces great performance improvements as the access pattern to disk cache is similar to on-chip cache. Disk cache also works on the principle of locality wherein the recently accessed data is more likely to be accessed again and also the next data sets to be accessed may be packed closer to the current data set accessed [13].

Disk Cache can be placed at any convenient location between the CPU and physical disk along the data path. If the cache is closer to the CPU then the data access is also faster. Along with this the same cache can be used by multiple disk as their caching system.

2.2 Disk Cache Optimization Techniques

In this section we discuss a few disk optimization techniques that are relevant to this research.

2.2.1 Optimization Techniques at Secondary Storage (Hard Disk cache)

Qing Yang and Yiming Hu [18] present a novel disk storage technique called disk caching disk in order to improve the I/O performance of disk cache. In this research, they use a small log disk called cache-disk as the secondary disk cache to optimize the write performance. This exploits the access speed difference between the normal data disk and cache disk with the latter being faster even though both have the same physical characteristics. This speed is because of the different data units used and also the difference in the way the data is accessed. Data transfer rate in units of tracks is almost eight times faster than in unit of blocks [18] and based on this observation, the log buffer is used as an extension to RAM buffer to cache file changes and then destage this to the data disk when the system is idle. All the small and random writes are first buffered in to the RAM cache. Later all this data is written in one data transfer to the cache disk when it is idle. As a result, RAM buffer is cleared for more data transfer. When the disk is idle, the destage operation between the cache to disk and data disk is performed. Experiments show a performance improvement of two orders of magnitude in response time for writes. Experimental results with three traces namely hplajw, cello and snake, show that the DCD technique improves the write performance at the secondary storage level by one to two order magnitude [18]. This technique involves the use of an additional hardware element to get the performance improvement.

As shown in Figure 2.2 [18], cache disk can be a separate physical disk drive or a logical partition that is residing on the disk drive or a group of disk drives [18] as shown in Figure 2.3 [18].

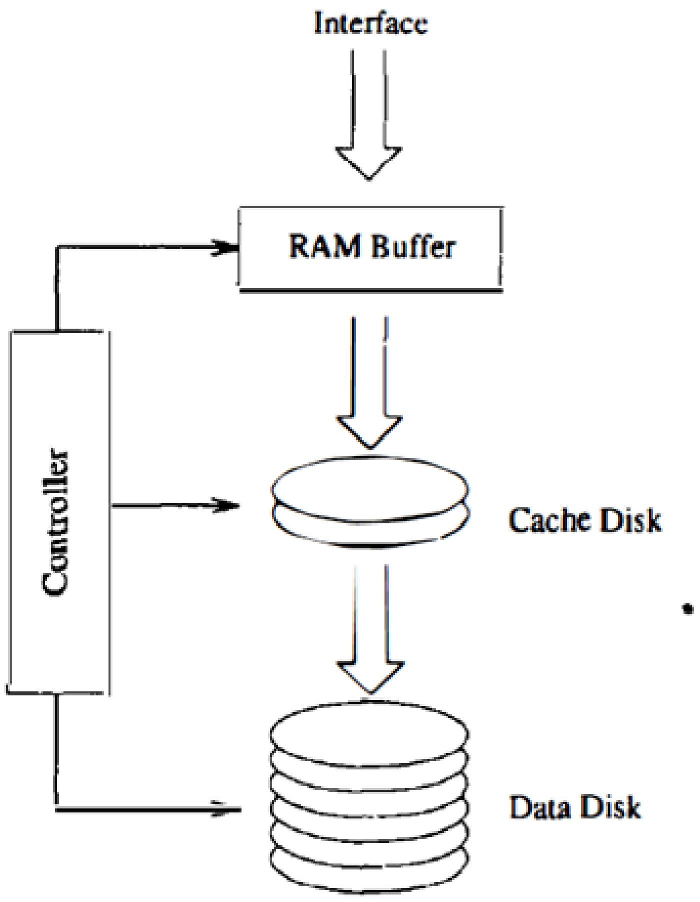

Figure 2.2: DCD with 2 Physical Disks

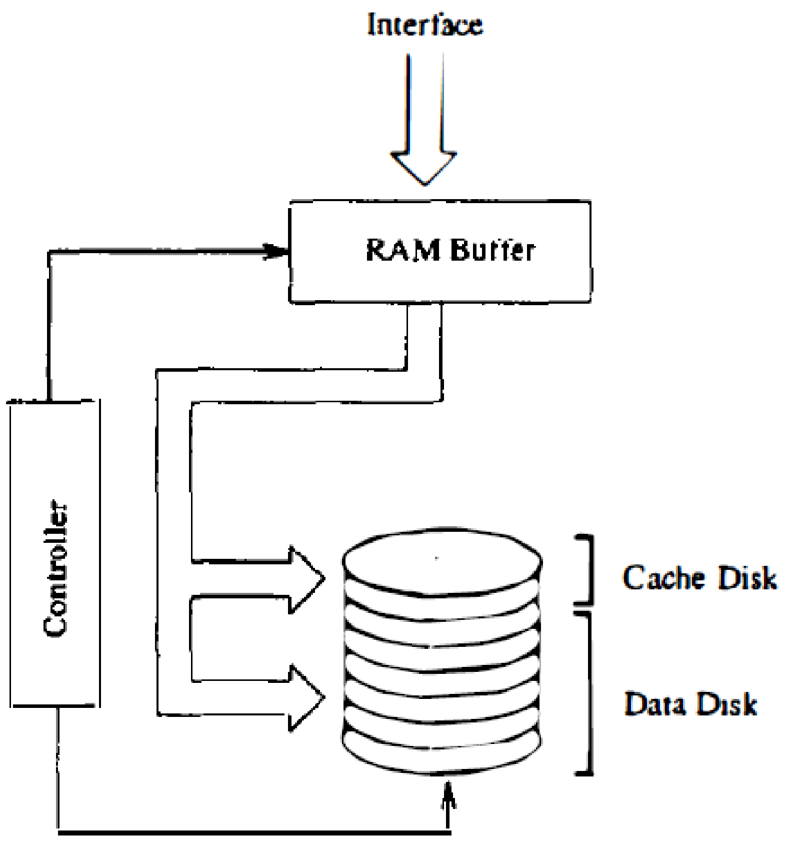

Figure 2.3: DCD with 2 Logical Disks

Yingwu Zhu and Yiming Hu [19] have investigated the performance of large built in caches on response time of the file system. The conclusions from this work are:

- With a file systems size of 16MB and more, there is little performance difference with a built in cache bigger than 512 KB.
- When disk built-in cache is used as a readahead buffer, there is considerable performance benefits for workloads with read sequentiality but fails to provide improvements when there are more concurrent sequential workloads than cache segments.
- It provides some improvements with some workloads when used as writing cache but reduces the reliability.

Hence with these findings, they claim that using bigger caches does not improve the performance and hence is a waste of power and money. With this, they conclude that disk manufacturers can plug in smaller built-in cache for better performance.

2.2.2 Optimization Techniques for Metadata Access

In this section we discuss a few optimization techniques introduced to enhance the handling of metadata in the secondary storage systems.

Pen Gu et al. [5] have proposed a prefetching algorithm for metadata access. Their study is based on the fact there there are many optimizations done for data prefetching but not for metadata access. According to this paper, the existing data prefetching algorithms do not take group prefetching in to account and have higher computational complexity. These techniques do not work with metadata access. Hence in this paper, they propose an accurate and distributed metadata-oriented prefetching algorithm [5]. They present a novel weighted-graph-based technique for prefetching. Experiments conducted with this algorithm have shown considerable improvements for metadata access on the client side. The average response time for metadata was reduced by 67% compared to the LRU caching algorithm and other prefetching algorithms available.

Bo Hong [6] has evaluated the performance impact by using micro electromechanical systems(MEMS) as metadata storage and disk cache. MEMS have seek times which is 10-20 times faster than hard drives, storage density 10 times higher and also lower power consumption. When MEMS is used as dedicated metadata storage in file system, simulations how that there is an improvement of 28-46% in system performance using a user workload depending on how much metadata traffic counts for the whole workload. Also they have discussed that by using MEMS as disk write buffer, the system performance can be improved by a factor from 3.3 to 8.2. They also show that this system provides better consistency on system performance than a disk system by a factor of 2.4 to 5.7.

Scott et al. [2] have discussed the importance of efficient metadata management in large distributed storage systems. In this paper, they claim that subtree partitioning and pure hashing are two common techniques used for managing metadata in such systems, but have a bottle neck of high concurrent access rates. They present a new approach called Lazy Hybrid(LH) technique for metadata management which combines the advantages of the two approaches leaving behind the disadvantages.

One common conclusion in these publications is the fact that metadata is smaller in size but is accessed very frequently. This is the basis for our design and more on the trace formats is discussed in the next section.

2.2.3 Split Disk Cache into Data and Metadata

Baskiyar et al. [1] have discussed the split cache architecture wherein they split the data region of the cache in to data and metadata. In this theoretical paper, analysis of split architecture has been carried out in order to improve the read miss ratio. They have claimed that it would reduce the interference between data and metadata. They claim that by splitting the disk cache the effective read miss ratio can be improved by 20% and this in turn would improve the response time by 16%.

Size of Metadata(block size 4KB)	**Size of data**
128+263=391B (Direct)	0-48K
391+4K=4.38K (1-indirect)	48K-4.02M
4.00M (2-indirect)	4.02M-4G
4G (3-indirect)	4G-4TB

Table 2.1: Size relation between data and metadata

We provide a practical implementation of this paper. In this paper, they have also made detailed analysis about data and metadata. Some of the highlights of this literature is listed below:

- Metadata accounts for only a small portion of the data but is accessed very frequently.
- In this paper, they have used Linux EXT2 file system to calculate the size of metadata. The size relation between data and metadata is provided in the table.
- It has been concluded that metadata account for 4.66% in an 8KB file and 3.156% in a 12 KB file.

From this paper, the overall read miss ratio for split disk cache scenario is given by the below equation:

$$\begin{aligned} f(x) &= 2.10k(\tfrac{tx}{m} + 2.14)^{-1.07} \\ &+ 2.10(1-k)[\tfrac{(1-t)x}{1-m} + 2.14]^{-1.07} \end{aligned} \tag{2.1}$$

Where

x - percentage of storage used in the cache including metadata cache and data cache

k - fraction of requests to metadata

m - fraction of file occupied by metadata

2.2.4 Cache Partitioning based on processes

Thiebaut et al. [16] have introduced an on-line algorithm to partition the cache storage in to disjoint blocks whose sizes are determined by the locality of the processes access the

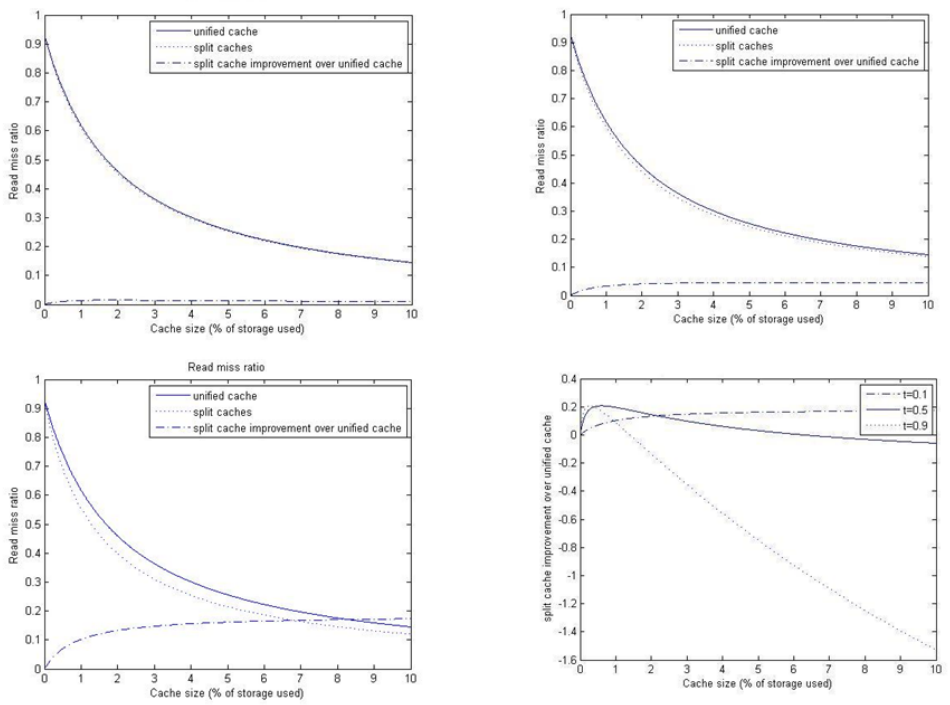

Figure 2.4: Split Cache Result for different values of k and t

cache. This models is a direct extension of the model presented by Stone, Wolf, and Turek [14]. In this work [14], they have deduced that, when two processes A and B share a cache of size C, by partitioning the cache by allocating C_A lines to process A, and C_B lines to process B ($C_A + C_B = C$), maximum hit ratio can be achieved when miss rate derivative of Process A as a function of cache size, in a cache of size C_A is same as the miss-rate derivative of process B in a cache of size C_B[16]. This rule holds for both, set associative caches and for fully associative caches. But in [16], it is being used for only fully associative cache. In [16], they provide a practical implementation of the replacement policy of Stone, Wolf and Turek [14] and also provide new ways to derive miss-rate derivative on-line. This model is adaptive and it uses shadow directory for each class. The shadow directory contains only the block addresses and no data and hence the additional space required for this is very minimal.

Figure 2.5[14][16] illustrates the partitioning model of stone, Wolf and Turek (SWT).There are two processes A and B and a cache of size 1024 lines. The miss rate for process A is shown above for increasing cache sizes. The miss rate of Process B is also shown in the graph

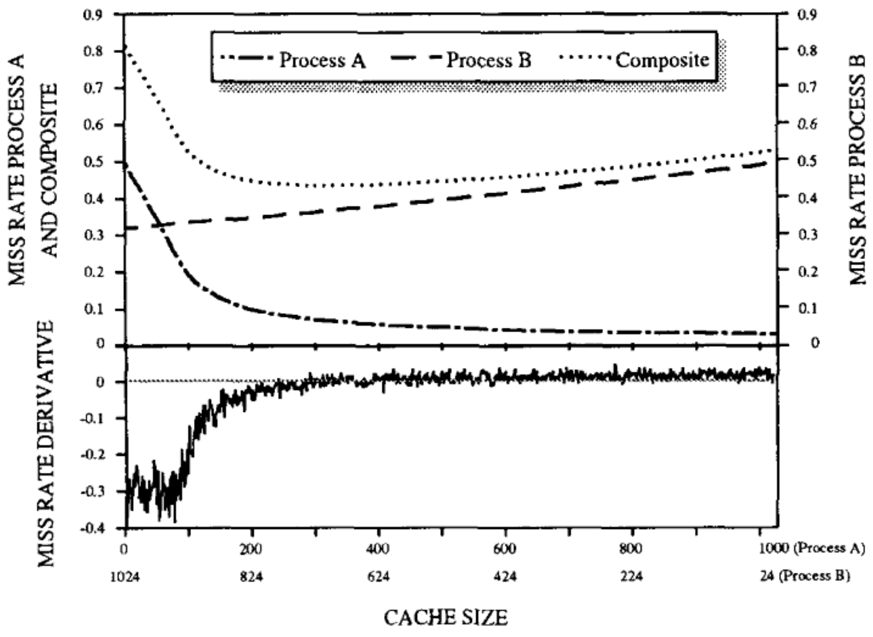

Figure 2.5: Miss Rates and Composite Derivatives for Two Processes

but for decreasing cache sizes. Assuming that all the lines accesses by process A is stored on the left side of the cache and all the lines accessed by process B is stored on the right side, the composite miss rate for these two processes can be computed very easily. A vertical line dividing the cache in to two partitions, one for process A and the other for process B can be drawn and the point at which they intersect the miss rate of A and B, the miss rates can be added to get the miss rate for the composite cache. The curve for miss rate in a composite curve is shown above the curves for A and B. Theoretically, it is always efficient to partition the cache at a point where the composite curve hits a global minimum. The SWT model shows that when the optimum partition is reached, the derivatives of the miss rates of Process A and Process B with respect to cache are identical [16].

The calculation of approximate miss rate derivative to implement in the algorithm is shown below. With a cache of size i, the miss rate of a process can be represented as the percentage of lines that flow through the ith cell of an infinite LRU stack. Also, when there is a miss, the new block is brought in to the most recently used position of the stack and all the blocks are pushed to the right by one position. Hence in a cache of size i, the line at position i moves to the right whenever there is a miss. The miss rate derivative represents the incremental change in this flow as the boundary of the ith cell is moved by a small amount δ. In this implementation δ is assumed to be equal to 1. Hence the approximate equation [16] for the miss rate derivative for a process can be given by

$$\begin{aligned} Miss-rate\ derivative &= \frac{Number\ of\ A\ misses\ in\ C_i}{Total\ number\ of\ A\ and\ B\ references} \\ &- \frac{Number\ of\ A\ misses\ in\ C_{i+1}}{Total\ number\ of\ A\ and\ B\ references} \end{aligned} \tag{2.2}$$

$$Miss-rate\ derivative = \frac{Number\ of\ A\ misses\ in\ C_i - Number\ of\ A\ misses\ in\ C_{i+1}}{Total\ number\ of\ A\ and\ B\ references} \tag{2.3}$$

Similarly, miss rate derivative is calculated for all the processes and the split is dynamically varied based on the result and the access patterns. From the above equation, it is clear that both the processes have the same denominator. Hence to obtain a partition region, we have to identify the point at which the numerators are equal. The numerator indicated the number of hits at index i+1 in the LRU stack. Similar principle is extended to all the processes running and a split point is identified accordingly.

This technique partitions the cache based on processes. If there are many processes running, then this technique creates a number of partitions in cache. This can be a overhead as the partition keeps changing as the frequency of the processes accessing the cache changes.

Input request format is an important factor in deciding the impact of optimizations at the storage level. Extensive work has been done to analyze the access patterns at the file system level. The findings in [9][10][15][8][12] have studied the file system access patterns in detail and have found optimization techniques based on the results. One of the conclusion in these studies is that access to metadata in the form of requests to inodes, directory entry etc. is frequent. These results were the driving force behind this research.

Ruemmler and Wilkes [12] conducted extensive studies on three benchmarks namely Hplajw and Cello. They concluded that 40% of reads in Cello are for metadata and the same is 10.7% in hplajw. Based on these studies, we evaluated our split cache implementation using standard and synthetic benchmarks.

Chapter 3
Optimum Split Percentage Design

The split cache design used in this research is similar to [1]. The rationale behind this development is explained below. From the experiment of Hsu and Smith [9][10], the cache system can be effective when the cache size reaches 1% of the external storage space used. For example, for a 320 GB hard disk, the disk cache has to be 1.6 GB when the hard disk is half full. It is impractical to provide a cache which is euqal to 1% of the used external storage. Hence we split the cache in to data and metadata regions, so that the metadata region at least reaches 1% of the space occupied by metadata in the hard disk drive. Consider an example where we have a cache of size 2KB as shown in Figure 3.1(a) [1]. The disk is accessed twice to retrieve a file of data size 2KB and metadata file of size 1KB. Metadata is first retrieved from the disk and stored in the cache as in Figure 3.1(b), and then the data is retrieved from the disk and brought into the cache. Since data is of size 2KB, it replaces the metadata in the cache as shown in Figure 3.1(c). When the same file is requested again, the metadata for the file is not present in the cache and hence there is a cache miss. Metadata is now retrieved from the hard disk drive and stored in the cache replacing 1KB data as in Figure 3.1(c). Hence the cache now contains 1KB data and 1KB metadata resulting in a hit for half data. But if the cache is split into data and metadata regions as shown in Fig 3.2(a), each of size 1KB, then the metadata does not get a miss in the second request. This reduces the interference between data and metadata and increases the overall hit ratio. It is important to identify the optimum split point at which, metadata hits exceed the loss in data hits and then provide additional performance gain.

But in the actual scenario, it is important to determine the split region. As there is some space taken away from the data region, the number of hits to data might go down. But

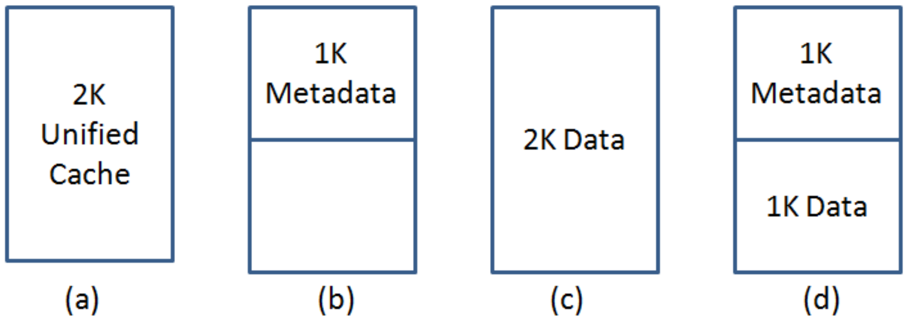

Figure 3.1: Miss Rates and Composite Derivatives for Two Processes

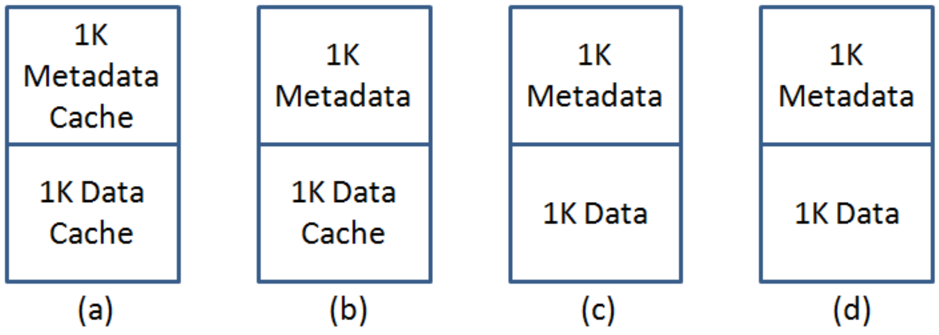

Figure 3.2: Miss Rates and Composite Derivatives for Two Processes

metadata is accessed more frequently [5][6][2] and hence the number of metadata accesses results in a very high number of hits compensating for the hits lost due to data loss. But on the other hand, metadata is small in size and hence the space for metadata has to be allocated carefully. Allocating more space than required for metadata would reduce the performance considerably as this space is at the expense of data region. In this thesis, we vary the percentage of metadata and data region and evaluate the outcome.

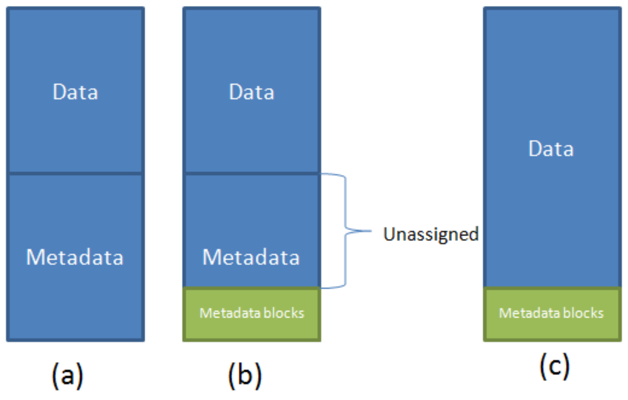

Figure 3.3: Data and Metadata Regions in Cache

As discussed in [1], to direct the requests to appropriate regions (data request to data region and metadata request to metadata region), the request commands are modified at the OS level, which can send one additional bit of information. This bit can be read by the disk controller and the request can be directed appropriately.

The motivation behind varying the split percentage is explained using the figure 3.3. In Figure 3.3(a), the cache is split at the middle of the cache. That is 50% for data region and 50% for metadata region. But metadata is smaller in size compared to data. Hence in 3.3(b), metadata blocks occupy only a small portion of the space allocated to it. As a result of this, there is lot of space unassigned in the metadata region. Since this space is taken from the data region and as a result data region also has less space for data in the cache. This increases the hit misses for data and as a result degrade the system performance. Hence as

Shown in Figure 3.3(c), we vary the data and metadata regions to find the optimum point at which the hit ratios are the maximum compared to unified cache.

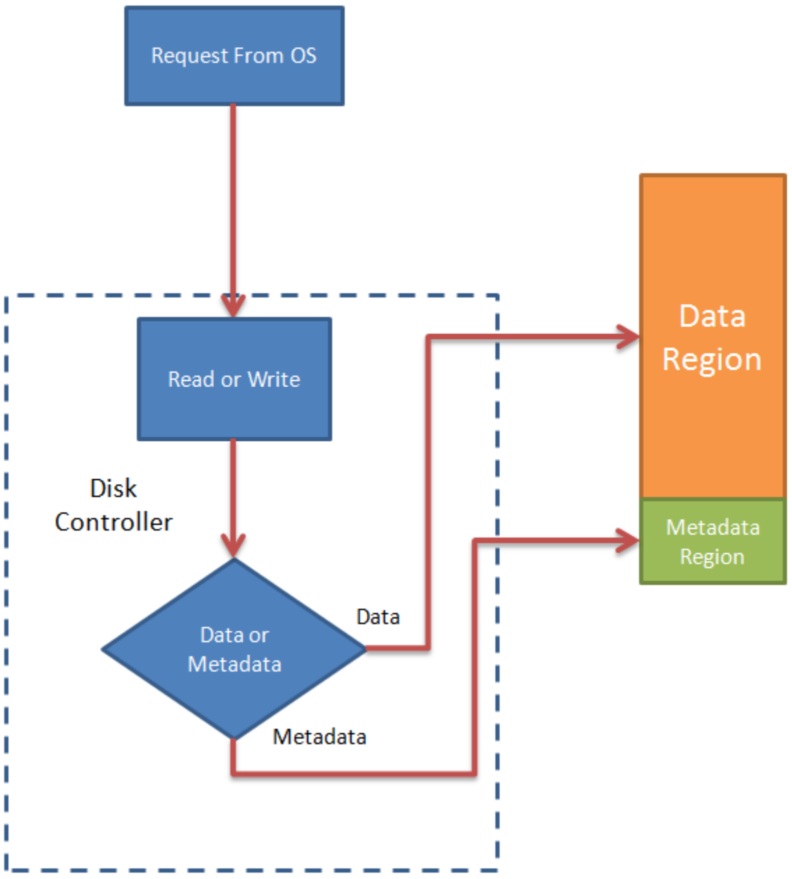

Figure 3.4: Data and Metadata Regions in Cache

In order to achieve the above mentioned design, there are a few tasks to be done at various stages along the data flow from processor to the disk. These are listed below.

- The nature of the I/O request. This involves finding out whether the request is a read or a write request.
- Once the I/O type is identified, it is important to differentiate between a data read and a metadata read.

- When the request type is completely identified, the request has to be satisfied by searching for the block numbers in their respective regions. This means that, if it is a data read request then we have to search for the block number in the data region of the cache. If not, then we have to start the search at the metadata region of the cache.

The nature of the I/O request can be identified at the operating system level. When the I/O request type is identified, the operating system can attach an additional bit along with the request to indicate that it is a metadata request. This bit can be read at the device controller level and then the search can be directed to the specific regions. This operation is shown in the block diagram Figure 3.4.

Chapter 4

Evaluation Methodology

The split disk cache scenario explained in the previous chapters was implemented in a well known simulator called DiskSim 4.0. This simulator is written in C language and is a very widely accepted for storage system simulation. We implemented our idea in to this simulator and carried out extensive experiments. We used many benchmark traces to arrive at the result.

In this chapter, Section gives a brief introduction to Disksim 4.0. In Section we discuss the system setup. In Section we talk about the traces files used. Metrics are presented in section.

4.1 Disksim Simulator

The proposed idea was implemented and evaluated using Disksim 4.0. Disksim is a well known simulator developed to support research in storage subsystems. Disksim includes modules which simulate disks, intermediate controllers, buses, device drivers, request schedulers, disk block caches and disk array data organization. DiskSim has been successfully validated against various commercial disk drives and has produced exceptional results. [3].

The system architecture of a DiskSim simulator is shown in Figure 4.1. The device driver has the implementation to direct the working of the underlying hardware. The Controller is a software piece which manages the overall application of the cache system. Typically, a controller decodes the request from the processor, checks the cache for the data, and if not present, retrieve the data from the hard disk drive. This data is then placed in the cache based on the replacement algorithm and is also sent to the processor. The device driver and the controller are connected to the system bus. The controller is connected to the cache through the I/O bus. The disk-cache is the component which has been modified to incorporate a split in to data and metadata regions. Basically, disk cache is a buffer system that is used in the simulator. The data region of the buffer is subdivided in to data and metadata and the split region is modified to get the maximum overall hit ratio.

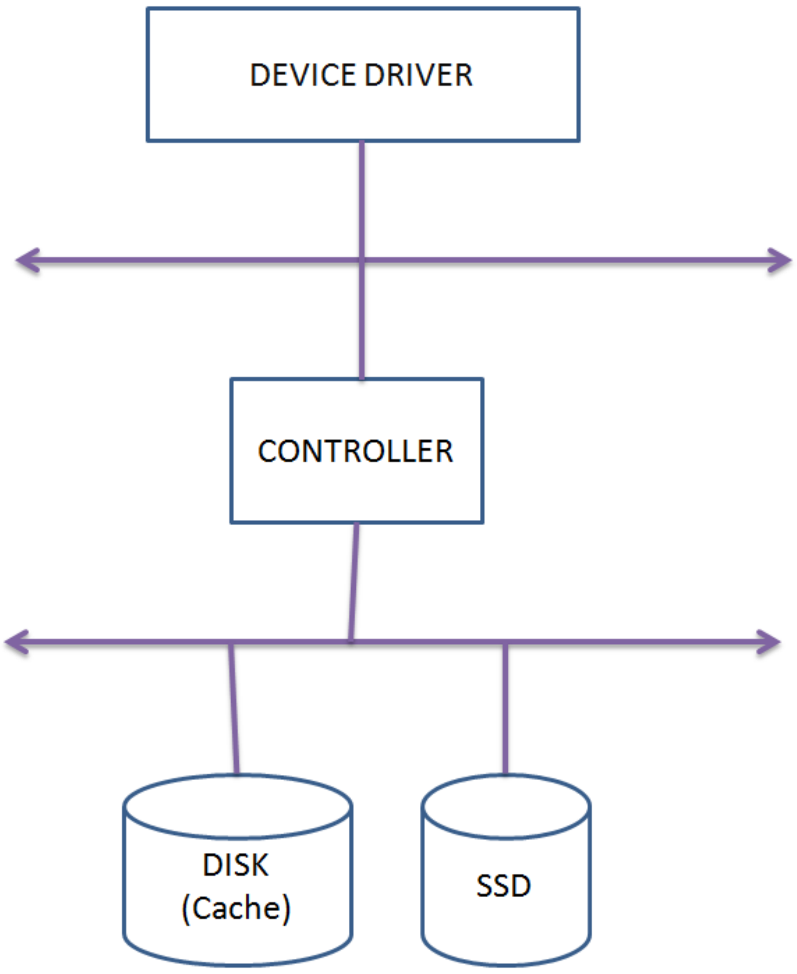

Figure 4.1: Disksim Architecture

4.2 System Characteristics

The system environment in which the split cache research was implemented is shown in Table

	Parameters	**Description**
Hardware	**CPU**	Intel (R) Core (TM) 2 Quad Q6600 2.4 GHz
	L2 Cache	4096KB
	Memory	2 GB
	Storage	Intel SATA X25-M 80GB SSD
Software	**OS**	Ubuntu 10.10
	gcc/g++	v4.4.5
	bisson	v2.4.1
	flex	v2.5.35
	GNU Bash	v2.1.5
	perl	v5.10.1
	python	v2.6.6
	gnuplot	v4.4
	pdfcrop	v1.2
	emacs	v23.1.1

Table 4.1: System Environment

4.3 Evaluation Traces and Workloads

Experiments were conducted with two benchmarks from HP-UX namely, hplajw and cello 1999. These traces have I/O events which consider both data and metadata access [5]. In hplajw, write requests dominate for the traces selected with 67% being write requests. On the other hand, read requests dominate in cello trace files accounting for approximately 63% reads in both the traces [18]. Out of overall read requests, metadata read requests account for 40% requests in cello and 10.7% in hplajw [12]. We used two full day traces of cello and a single day trace from hplajw. We evaluated the performance of the system with split disk cache and compared it to the results obtained through unified cache. Results for hplajw is shown in Figure 5.1 and Figure 5.2. Results for cello is shown in Figure 5.3 - 5.6.

Trace	**Processor**	**MIPS**	**HP-UX Version**	**Physical Memory**	**Fixed Storage**	**Read/ Write Ratio**
cello	HP 9000/877	76	8.02	96 MB	10.4 GB	0.79
hplajw	HP 9000/845	23	8.00	32MB	0.3 GB	0.72

Table 4.2: Hplajw and Cello-1999 trace Charateristics

Along with the two practical and real life benchmarks, there were various synthetic traces generated to evaluate the behavior of split disk cache. To determine the effectiveness of the split, there are two important factors to be considered about metadata. One is the percentage of metadata requests and the other is the access pattern. In order to account for these two important factors, we generated the synthetic traces of validate trace format and assigned a few read and write events as metadata events. Since metadata is smaller in size [5][6][2], we assign those read events which retrieve small number of blocks from the disk as metadata read. Based on the size calculation of metadata in 2.1, the block size in the cache and a detailed analysis in [17] about data and metadata block sizes, we randomly mark records where access size is less than or equal to 2 blocks as metadata. For a given cache size, the experiment is conducted with three different access patterns and the average value is computed. Figure 5.7 show the results with varying read percentage and metadata percentage. For example, if there are 10000 records in the trace file, then we randomly select 4000 records as metadata reads to account for 40% of metadata reads. We created trace files with metadata content as 20%, 40% and 60%.

Experiments were conducted with nine different cache sizes from 10KB to 16MB, different read percentages in the trace files ranging from 30-90%, different metadata percentages in the trace files ranging from 20-60% and different metadata:data split percentages of the disk caches ranging from 10:90 to 90:10. All the experiments were conducted thrice and the average value was evaluated. In total, there were 9*7*3*5*3 = 2835 data points with these different combinations. We vary the split regions between data and metadata to find an optimum point at which we get maximum number of hits.

To account for different access patterns for metadata, three different methodologies were used to generate the random numbers namely:

- Standard random number generator using API
- Random number generation using Gaussian Distribution

- Random number generation using Poisson Distribution

Initially for a particular trace file, all the block numbers with minimum number of blocks request (two or less than two) is collected. For a particular random number, the request at that line in the trace file is verified to see if the block number is in the list of block numbers collected before, and if it exists then this is considered to be a metadata access. This process is conducted thrice with the same trace files but different random numbers and different seeds. Finally, the average hit ratio is taken. In this way, we account for both metadata percentage in the trace file and also the different access patterns. This experiment is to mimic the metadata access patterns similar to the various test benchmarks. The entire trace file is spanned to obtain a particular percentage of metadata access.

4.4 Disksim Changes

A few modifications were done in disksim source code to implement the split cache design in data region of the cache. Major code changes were done in disksim_iotrace.c, disksim_disk.c, globalṁodspec, disksim_cachemem.c. There were a few modifications included in the parameter files of all the traces in order to define the split percentage.

4.5 Validation

Disksim 4.0 includes a set of validation tests used to evaluate and validate the simulator with the physical disks. The command for this is runvalid. Runvalid used a series of validation data to validate the simulator. The validation data have been extracted from a logic analyzer that is attached to an SCSI bus. By comparing the response time distributions of Disksim and various hard disk drives, the simulator was validated. After introducing the split cache design in to Disksim, it still produces same values as Disksim 4.0 for all validation tests shown in the Table.

Tests	**Disksim 4.0**	**Disksim 4.0 with Split Cache**
QUANTUM QM39100TD-SW	0.377952	0.377952
SEAGATE ST32171W	0.347570	0.347570
SEAGATE ST34501N	0.317972	0.317972
SEAGATE ST39102LW	0.106906	0.106906
IBM DNES-309170W	0.135884	0.135884
QUANTUM TORNADO	0.267721	0.267721
HP C2247 validate	0.089931	0.089931
HP C3323 validate	0.305653	0.305653
HP C2490 validate	0.253762	0.253762

Table 4.3: Root Mean Square Error of Differences(rms)

Tests	**Disksim 4.0**	**Disksim 4.0 with Split Cache**
Open synthetic workload	10.937386ms	10.937386ms
Closed synthetic workload	87.819135ms	87.819135ms
Mixed synthetic workload	22.086628ms	22.086628ms
HP srt trace input	48.786646ms	48.786646ms
ASCII input	13.766948ms	13.766948ms

Table 4.4: Response Time in milliseconds(ms)

Chapter 5

Results and Graphs

The split disk cache setup was implemented and tested with various traces from HP-UX (hplajw and cell) and synthetic trace files. As discussed, hplajw is a write intensive trace with 67% write requests. Cello is a read intensive trace and hence we take the traces from two days, 02/01/1999 and 03/01/1999 and check for the performance. We then generate synthetic traces of validate format and vary the percentage of data read and metadata read in the trace file. These trace files are then used with both split and unified cache to analyze the performance at different metadata and read percentage. Split cache evaluation is done with different metadata and data regions. Performance ia also evaluated with different cache size. The cache size if varied from 10KB to 16MB. The evaluation for split cache starts with the ratio 70:30 for data region and metadata region. Metadata region is increased gradually for a constant cache size, metadata and data percentage from the input. As the metadata region is increased, hit ratio is reduced. This is because the percentage of data region is reduced and hence the hit miss for data increases. Also, metadata occupies only a small portion of the memory and if allocated more space than required; the additional space is not sufficiently utilized thereby reducing the performance. The percentage of read also plays a critical role in optimizing the hit ratio. This technique provides improvement only when the percentage of reads in the trace file is above 70%. For all the other cases, there is either very small or no optimization. Hence split cache implementation provides improvements in hit ratio for read intensive applications and, when the data region is above 50%.

5.1 Evaluation Graphs

This section provides the graphical representation of the split disk cache evaluation.

5.1.1 Graphs for Standard Benchmarks (hplajw and Cello)

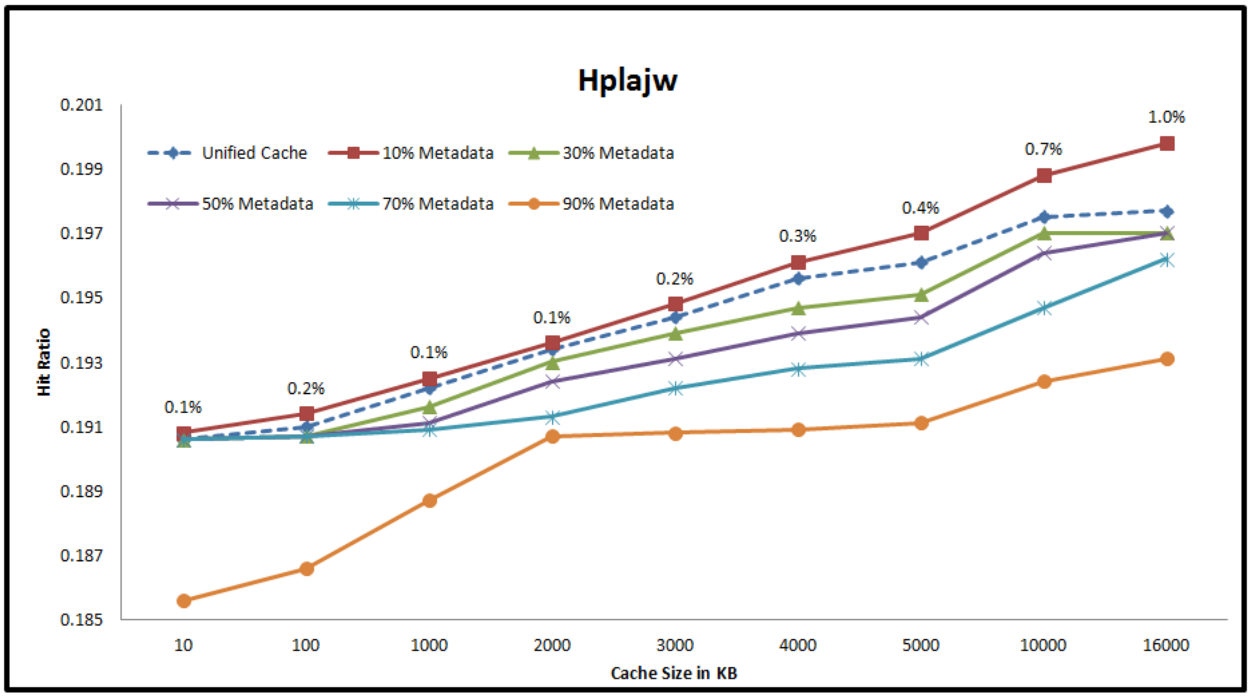

Figure 5.1: Hit Ratio with varying cache size for hplajw

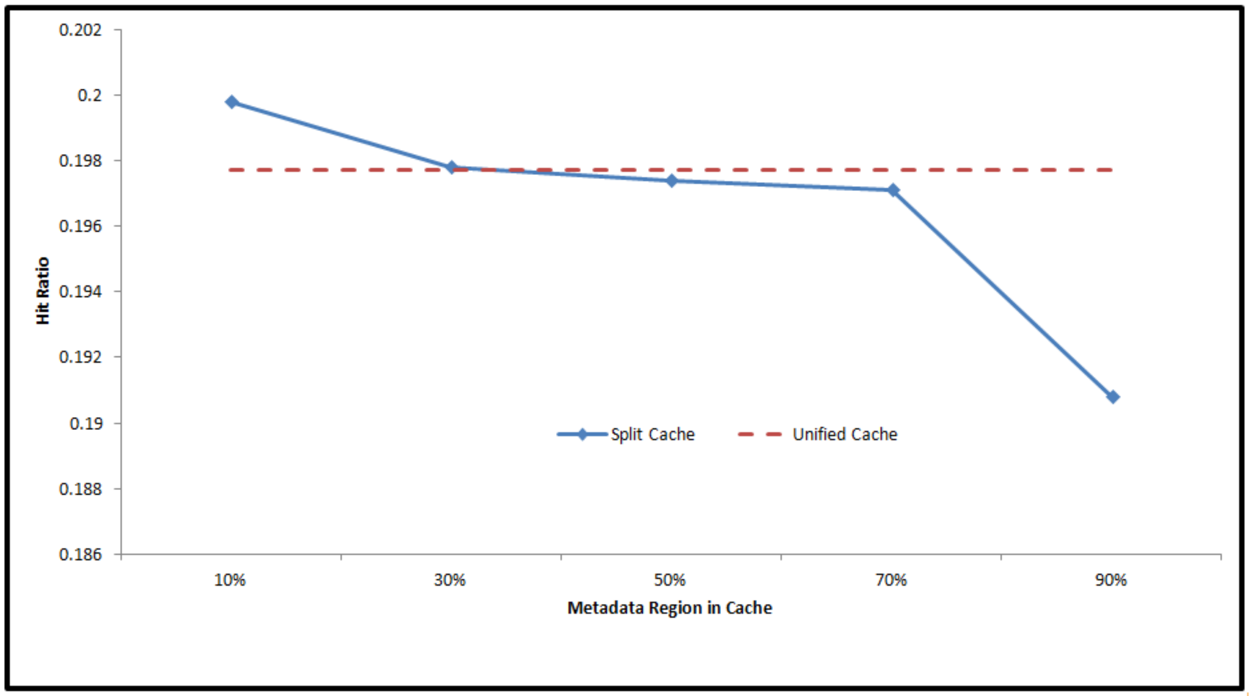

Figure 5.2: Hit Ratio with Varying Metadata region in Cache for hplajw

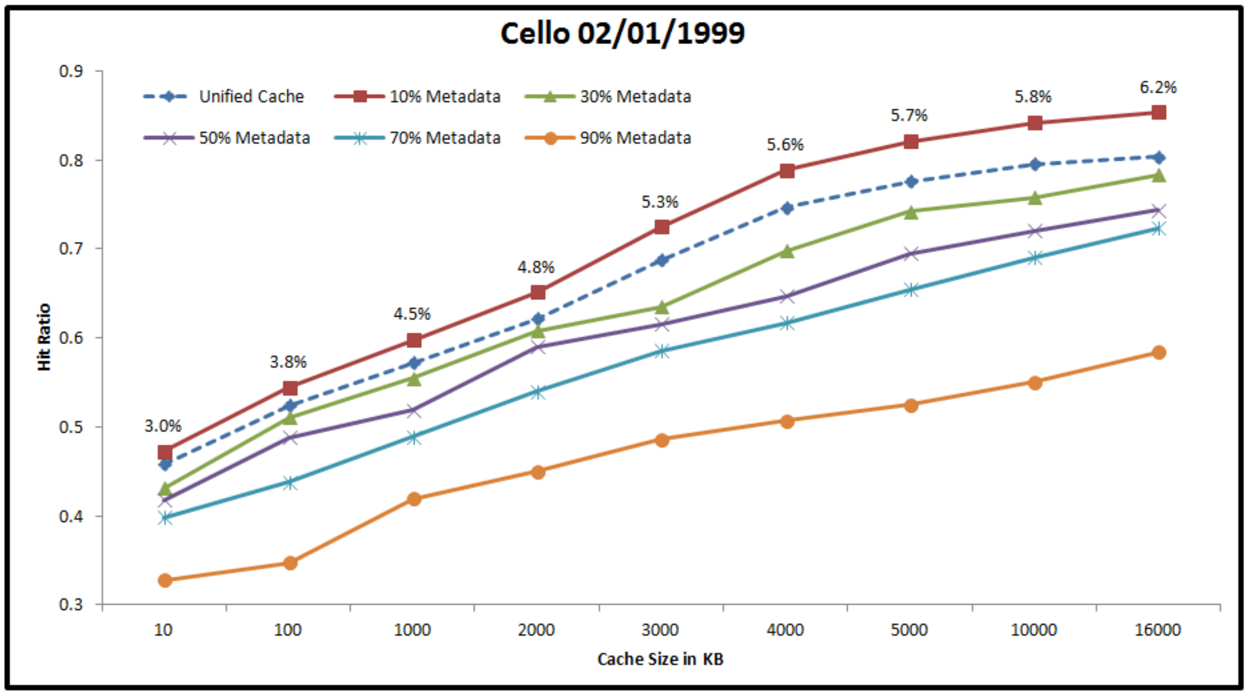

Figure 5.3: Hit Ratio with varying cache size for cello trace taken on 02/01/1999

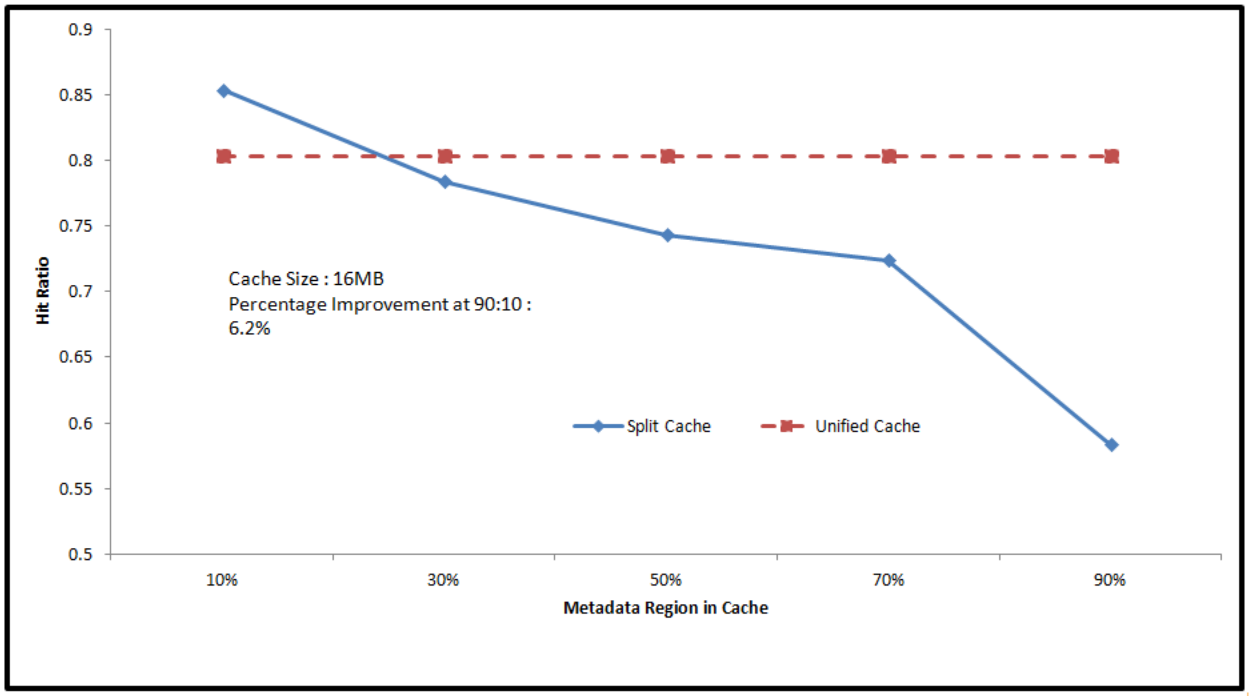

Figure 5.4: Hit Ratio with Varying Metadata region in Cache for cello trace taken on 02/01/1999

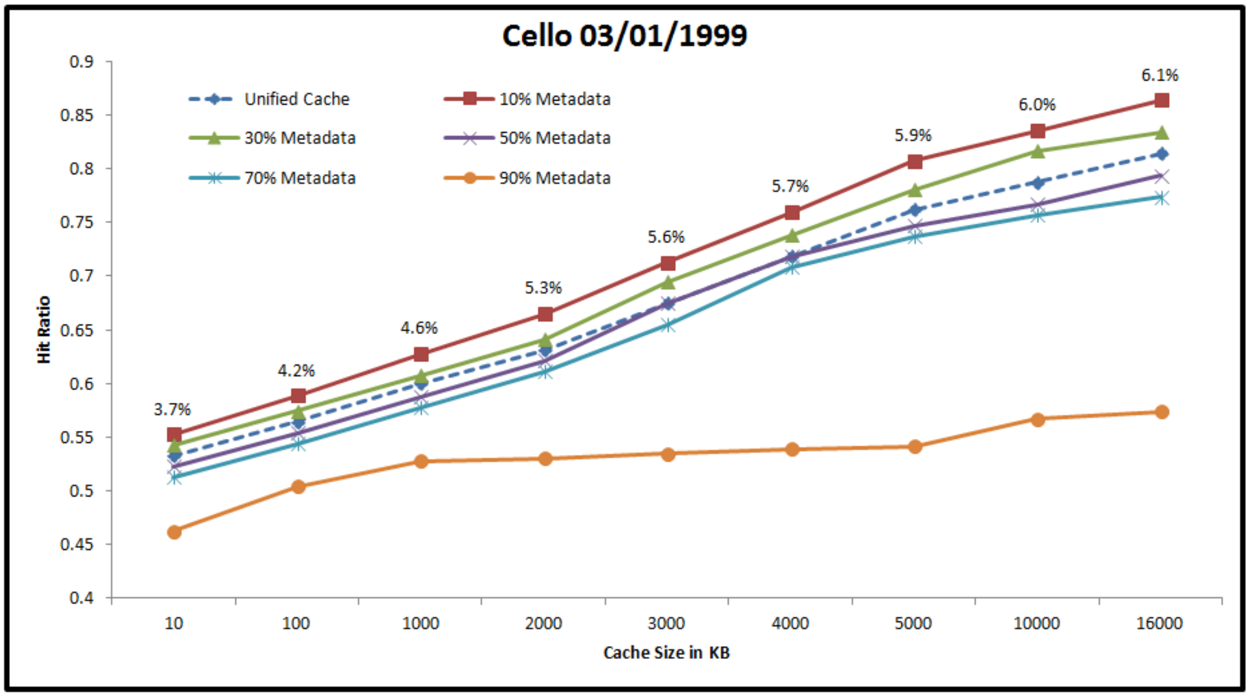

Figure 5.5: Hit Ratio with varying cache size for cello trace taken on 03/01/1999

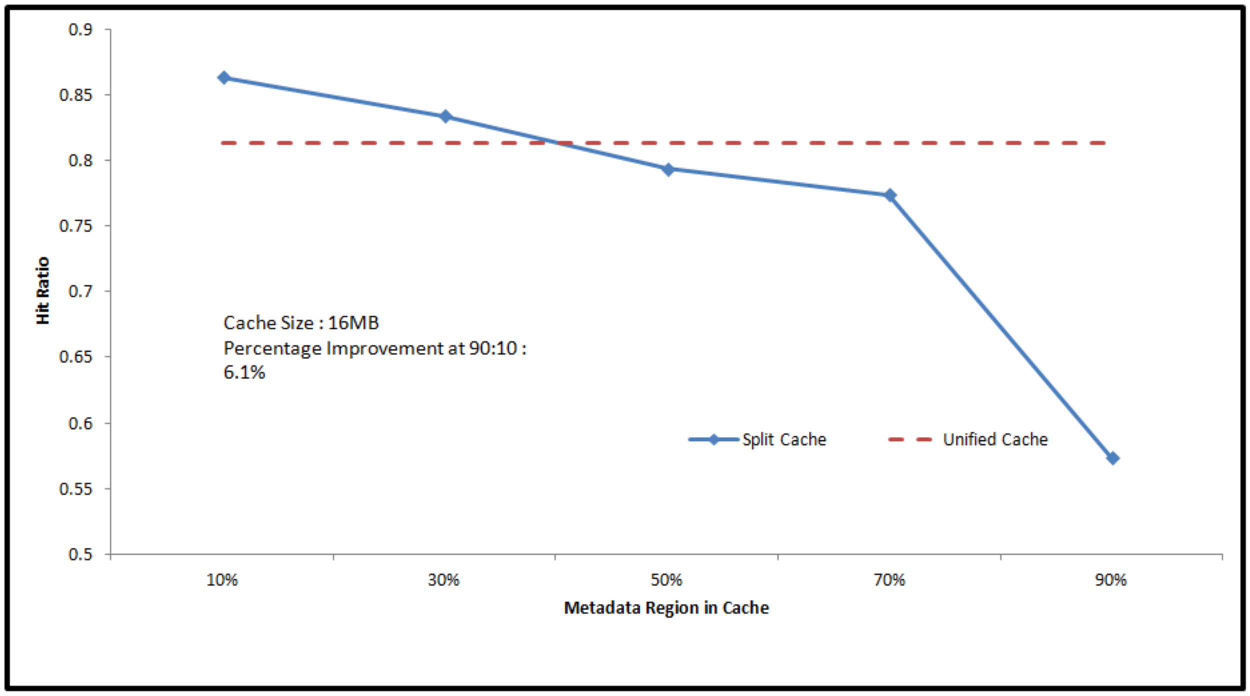

Figure 5.6: Hit Ratio with Varying Metadata region in Cache for cello trace taken on 03/01/1999

5.1.2 Graphs for Synthetic Trace files and Random Access Methods

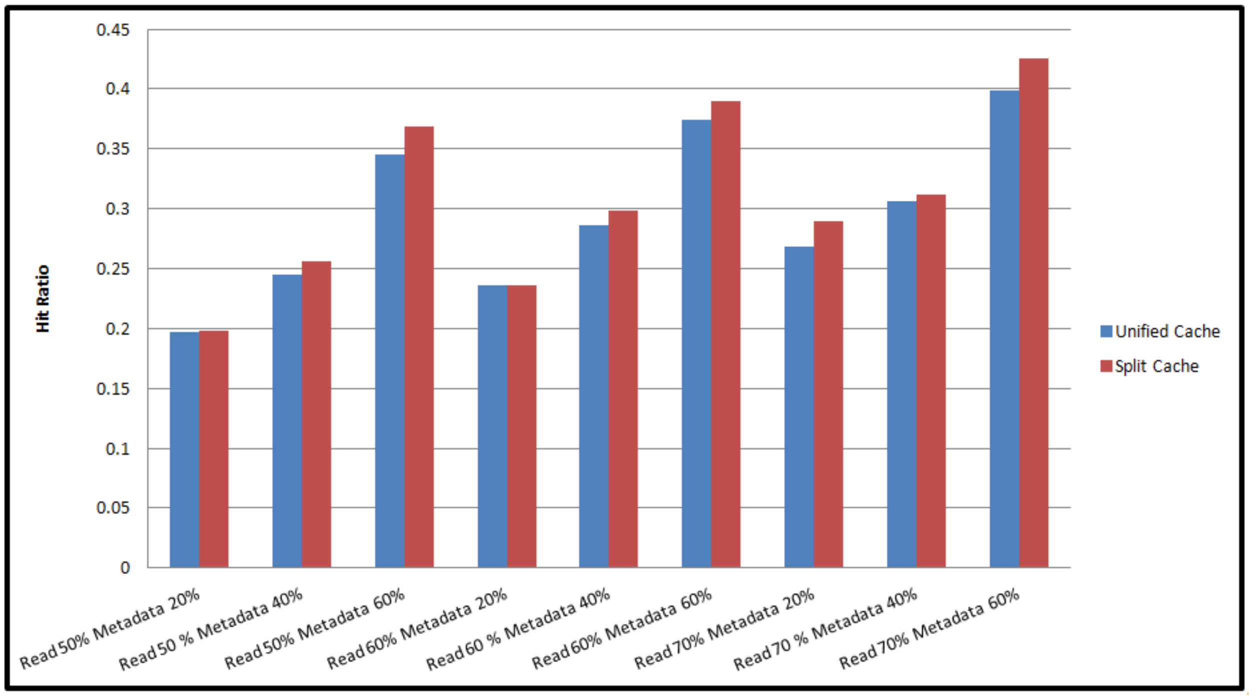

Figure 5.7: Hit Ratio with Varying percentage of Read and Metadata Read

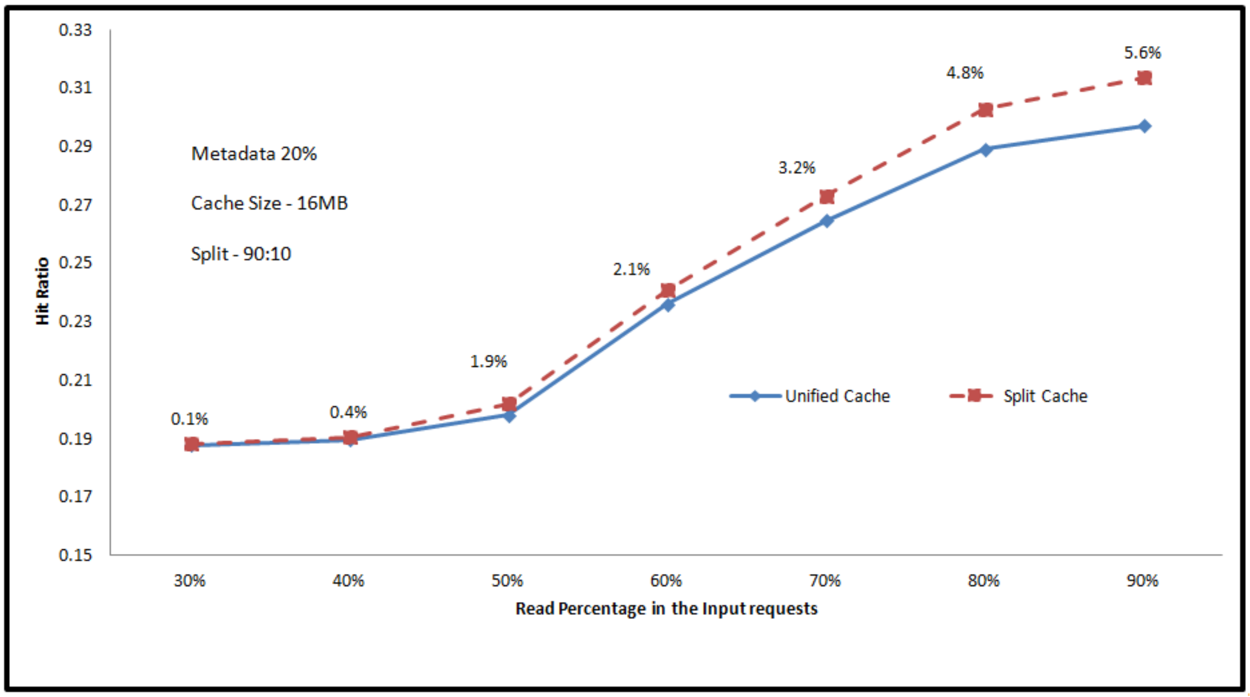

Figure 5.8: Hit Ratio with varying read percentage for Synthetic trace with Validate trace format and 20% Metadata

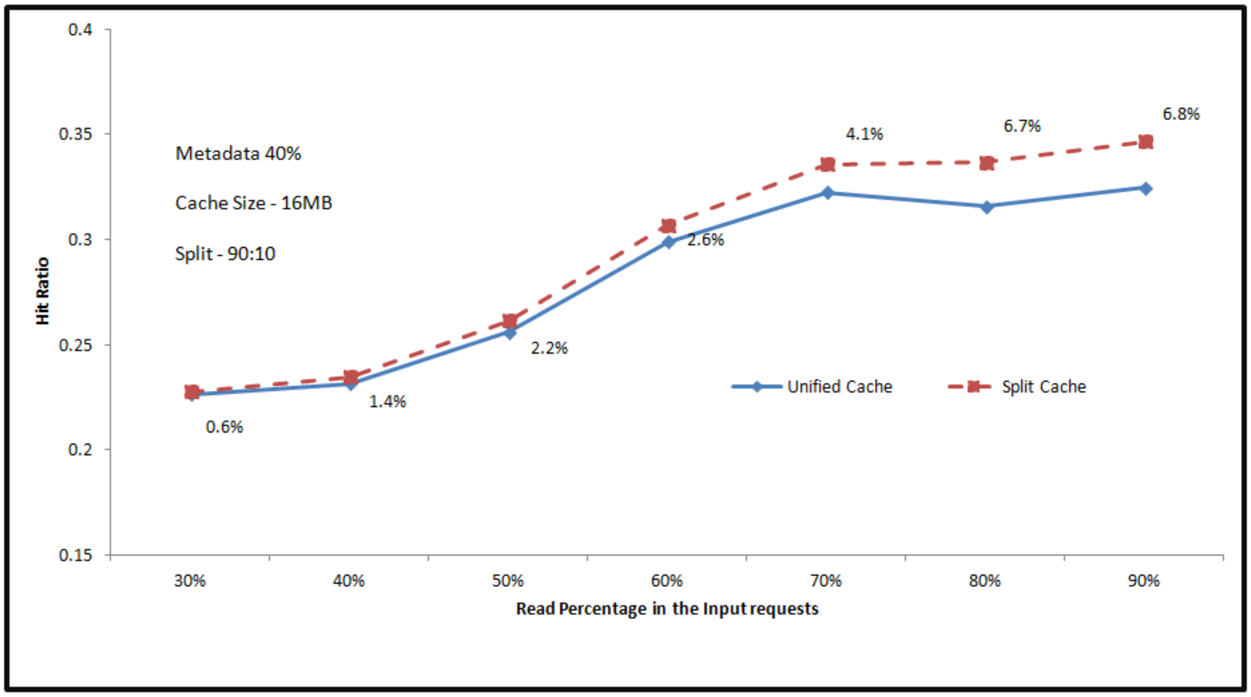

Figure 5.9: Hit Ratio with varying read percentage for Synthetic trace with Validate trace format and 40% Metadata

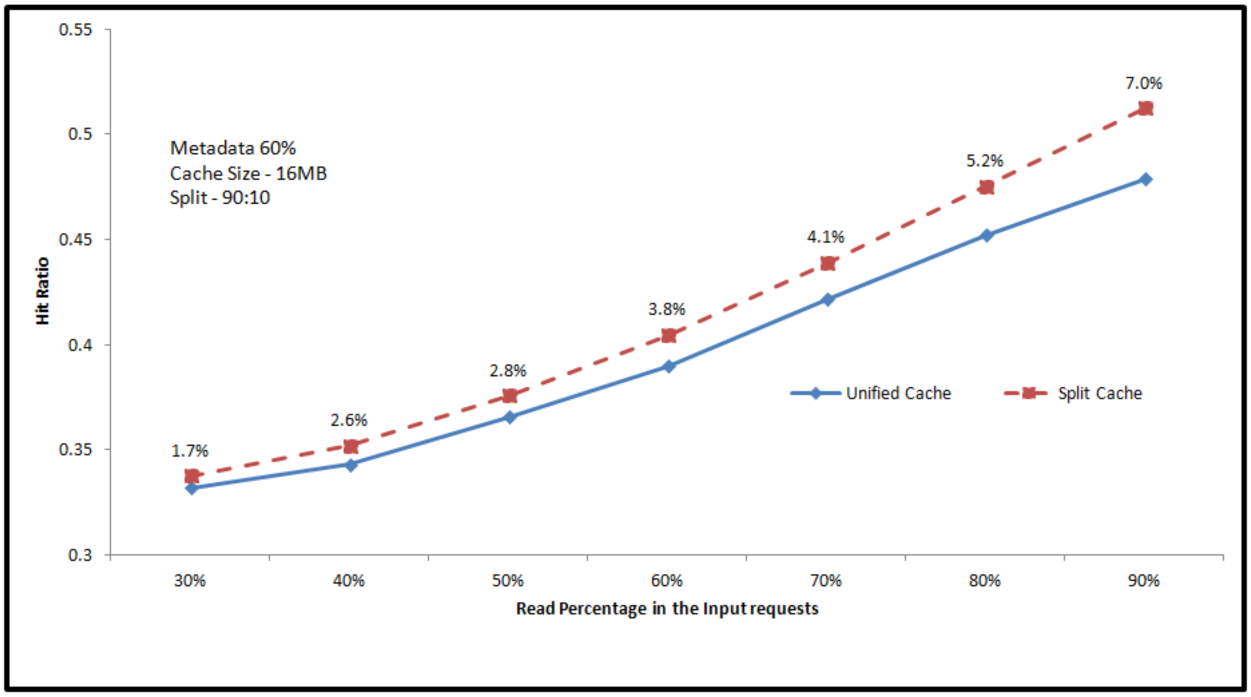

Figure 5.10: Hit Ratio with varying read percentage for Synthetic trace with Validate trace format and 60% Metadata

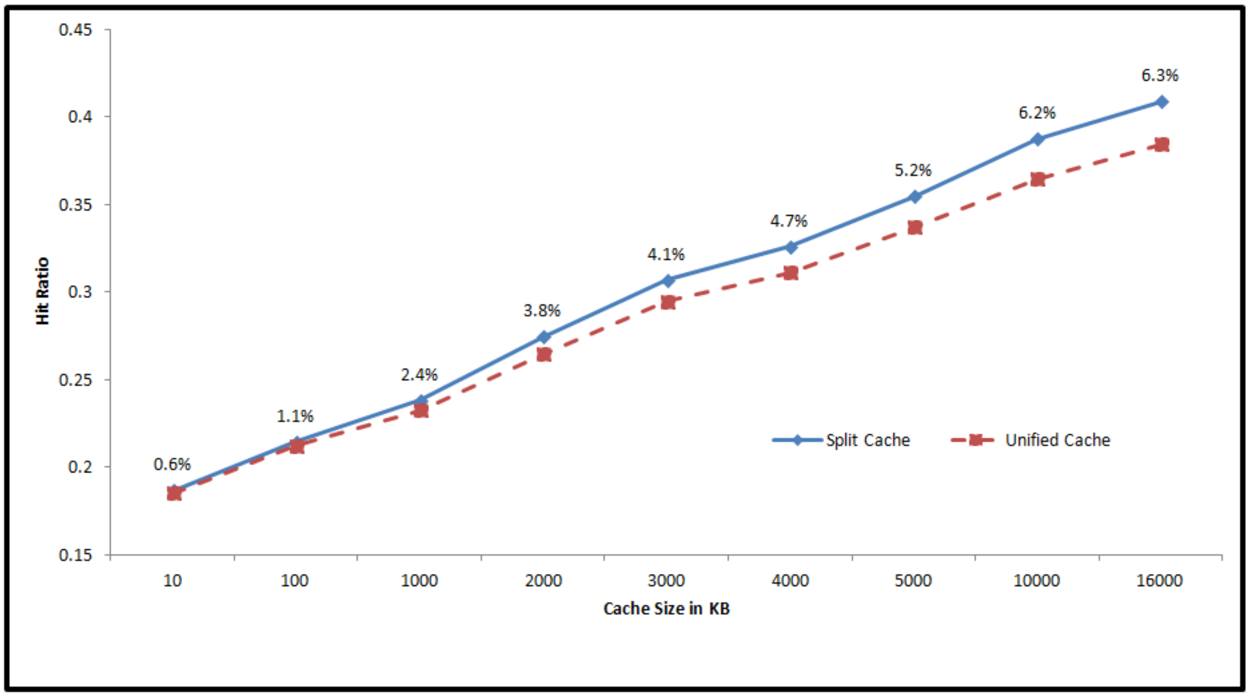

Figure 5.11: Average Hit Ratio with varying cache size for Synthetic trace with Validate trace format and random access

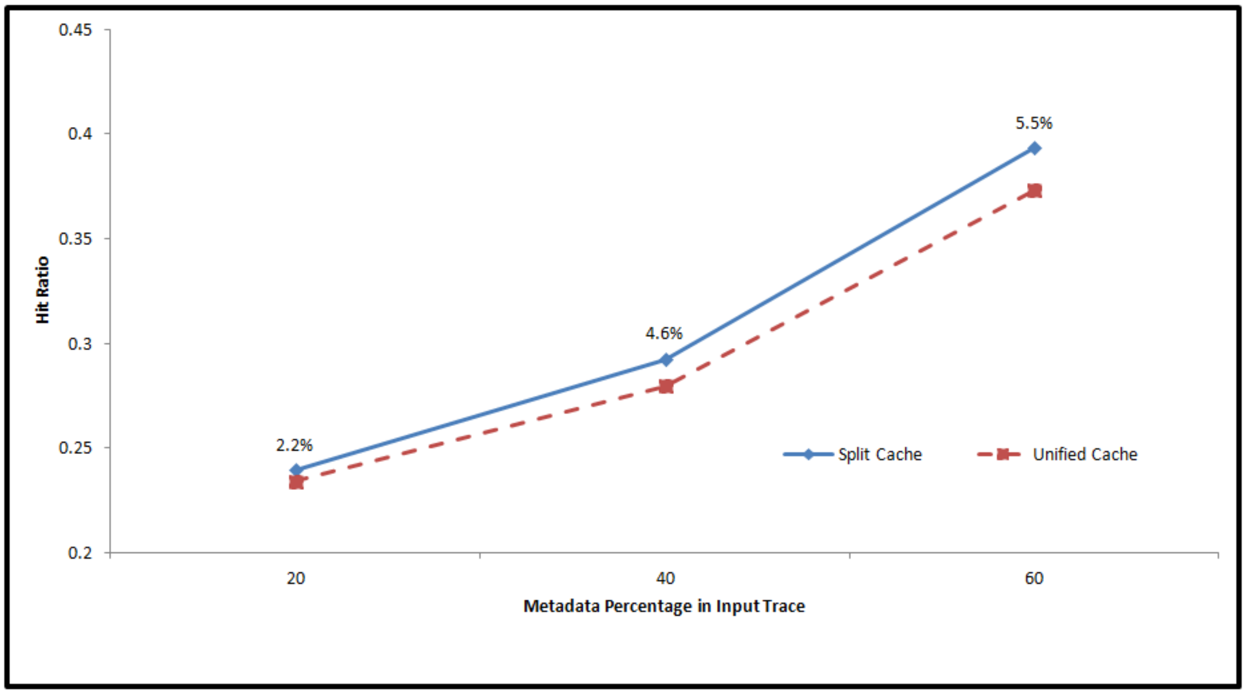

Figure 5.12: Average Hit Ratio with varying metadata percentage in input for Synthetic trace with Validate trace format and random access

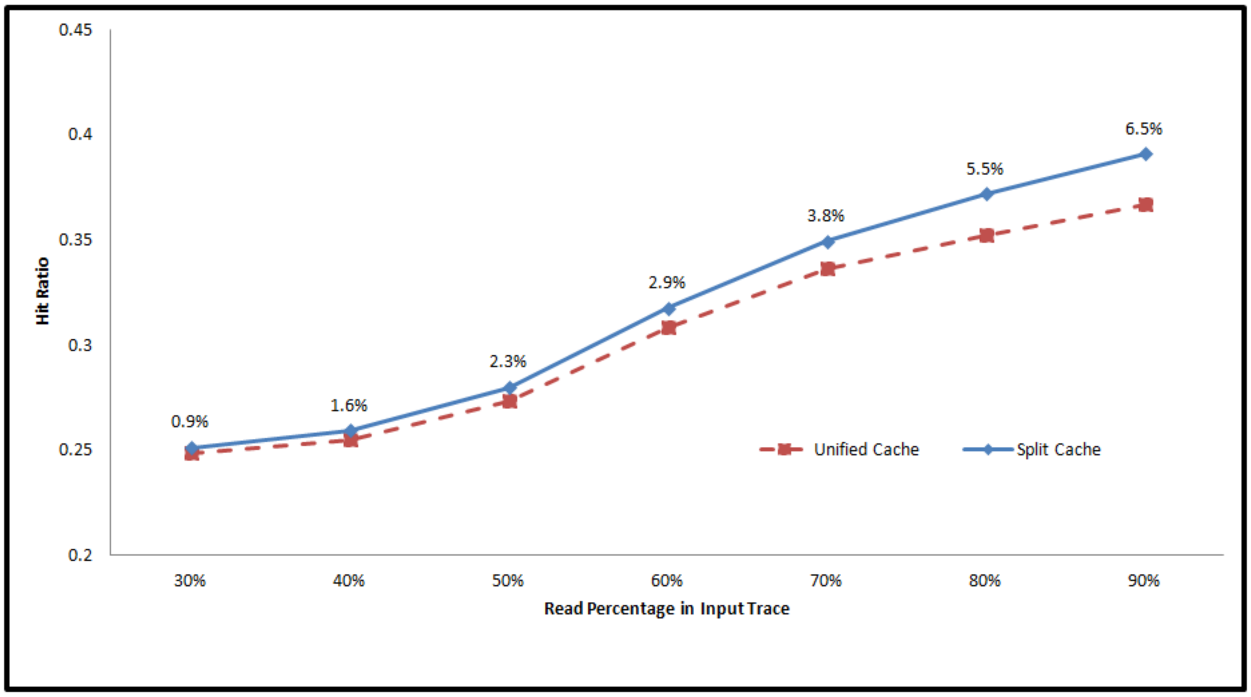

Figure 5.13: Average Hit Ratio with varying read percentage in input for Synthetic trace with Validate trace format and random access

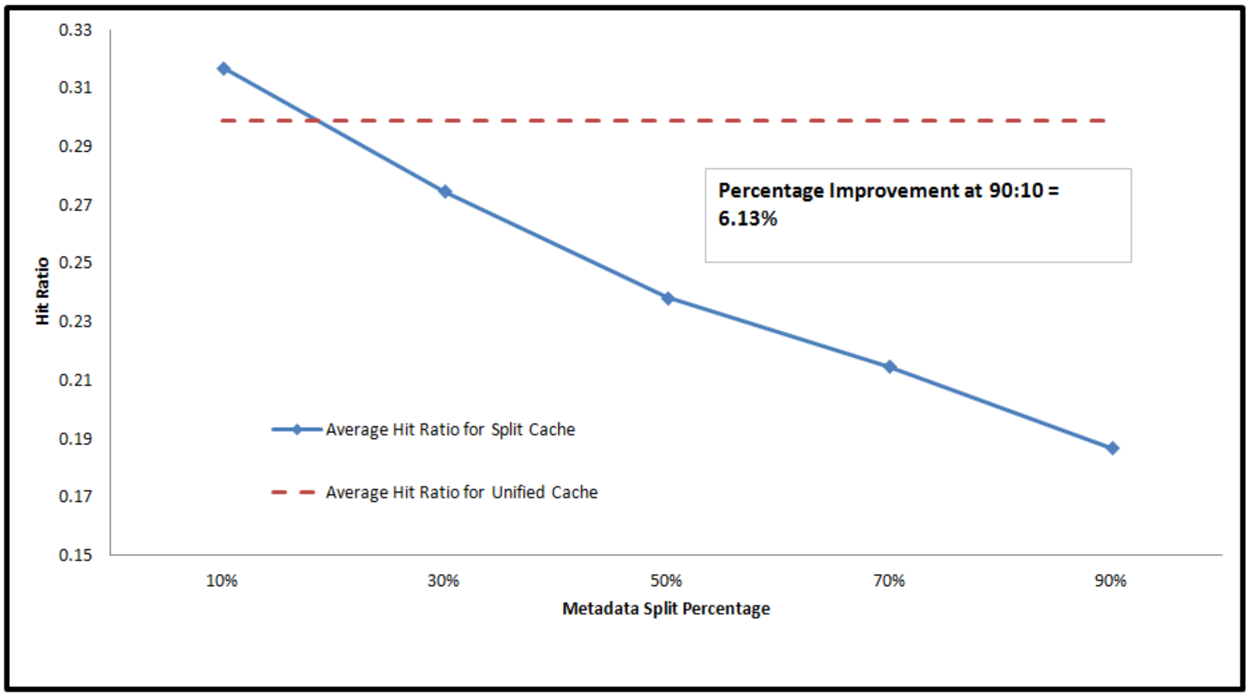

Figure 5.14: Average Hit Ratio with varying data:metadata split in disk-cache for Synthetic trace with Validate trace format and random access

5.1.3 Graphs for Synthetic Trace files and Random Access Methods using Gaussian Distribution Method and Poisson Distribution Function

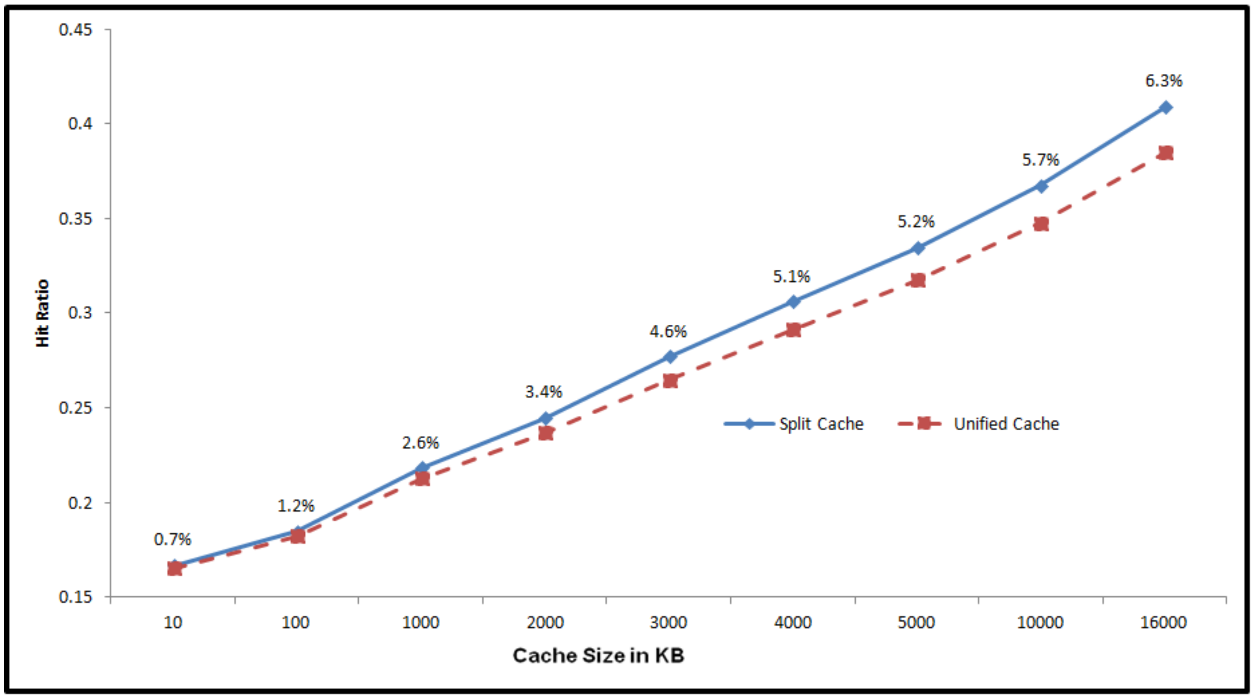

Figure 5.15: Average Hit Ratio with varying Cache Size for Synthetic trace with Validate trace format and Gaussian Distribution

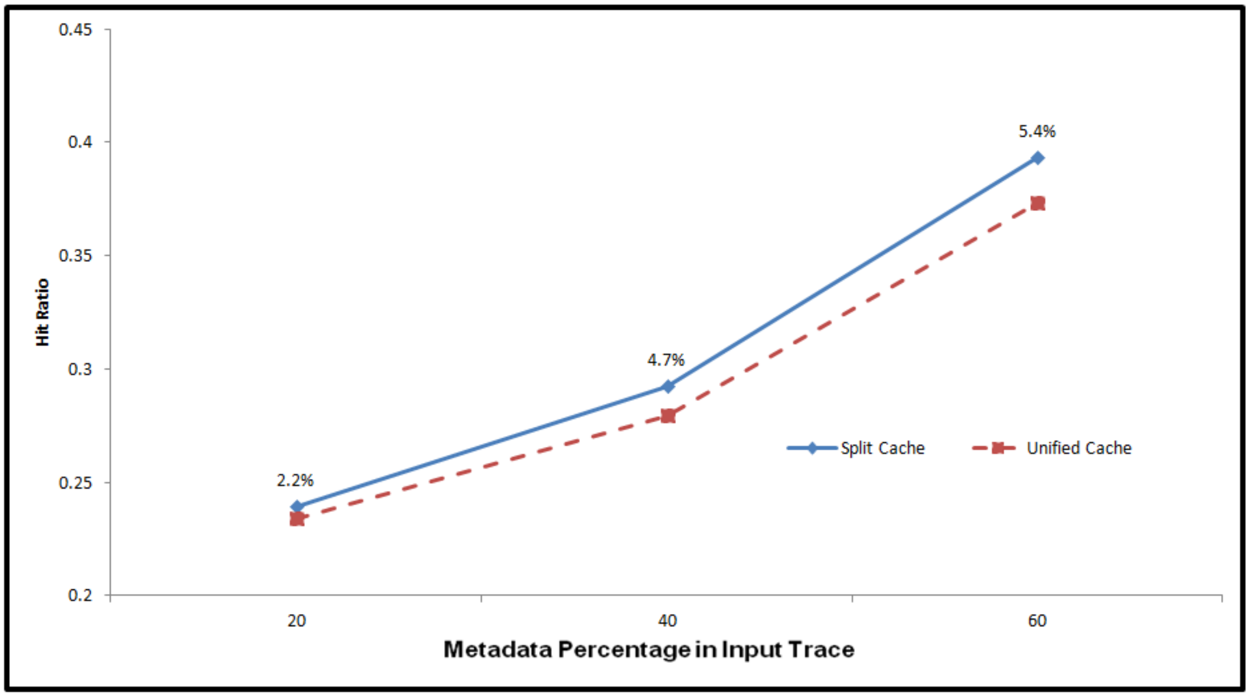

Figure 5.16: Average Hit Ratio with varying Metadata Percentage for Synthetic trace with Validate trace format and Gaussian Distribution

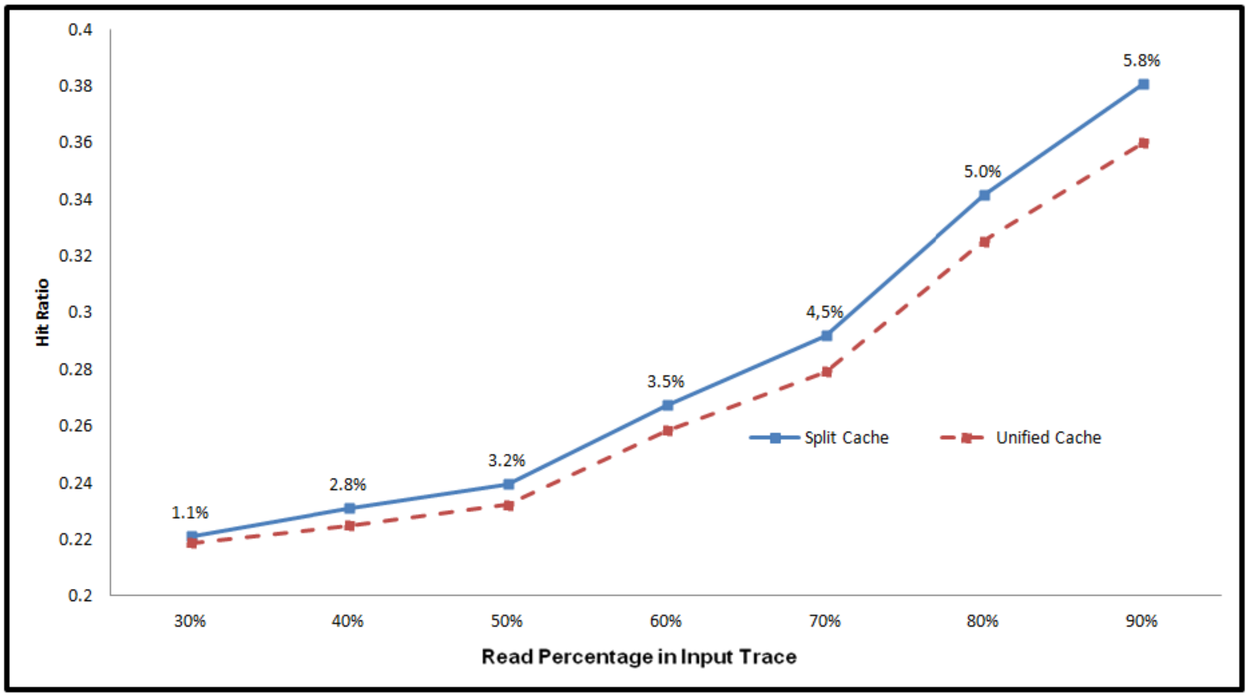

Figure 5.17: Average Hit Ratio with varying Read Percentage for Synthetic trace with Validate trace format and Gaussian Distribution

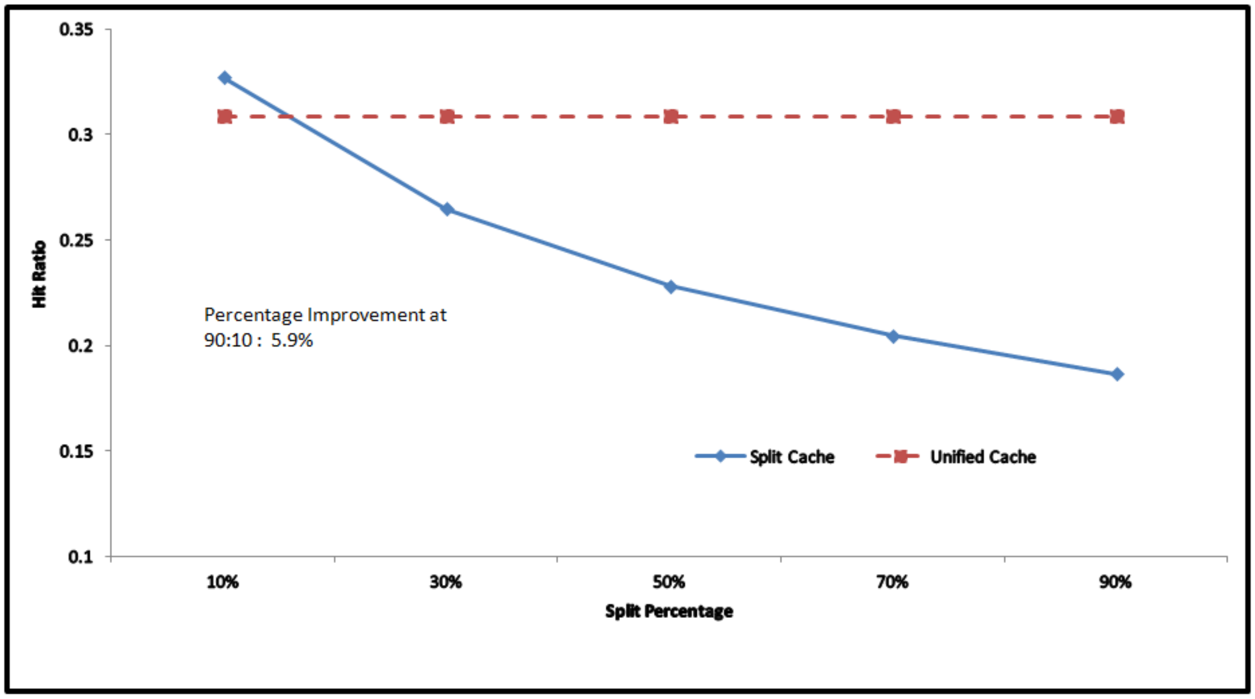

Figure 5.18: Average Hit Ratio with varying metadata split region for Synthetic trace with Validate trace format and Gaussian Distribution

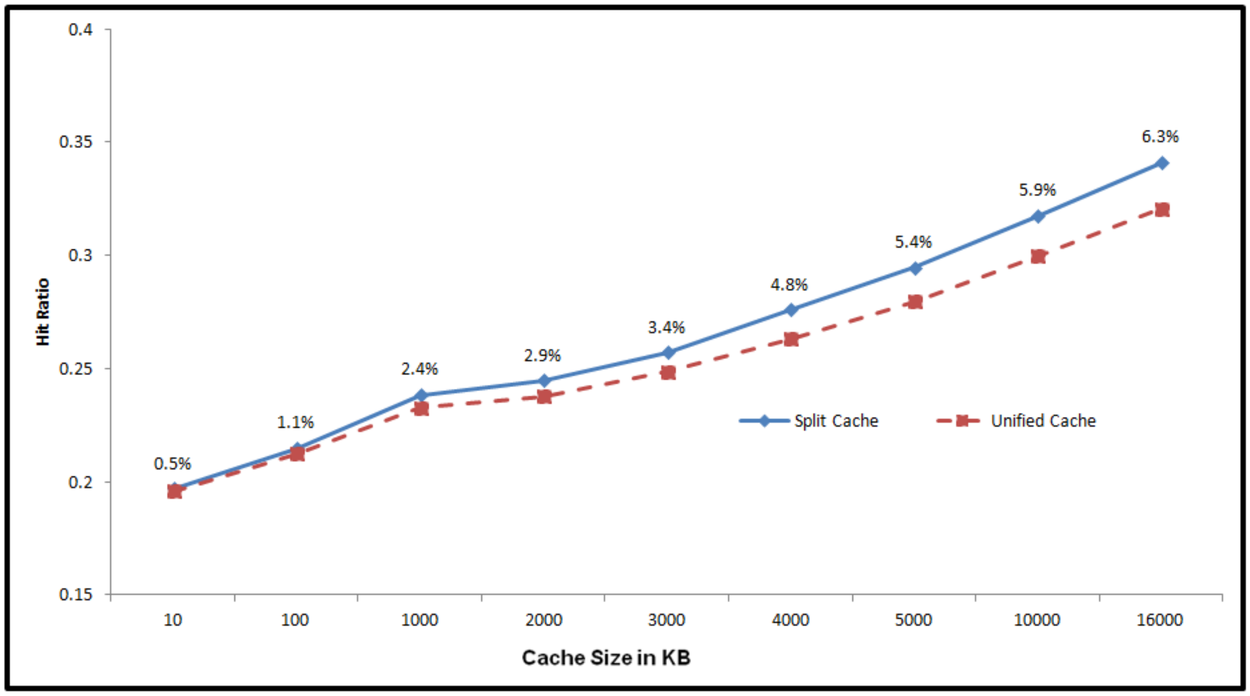

Figure 5.19: Average Hit Ratio with varying Cache Size for Synthetic trace with Validate trace format and Poisson Random Access

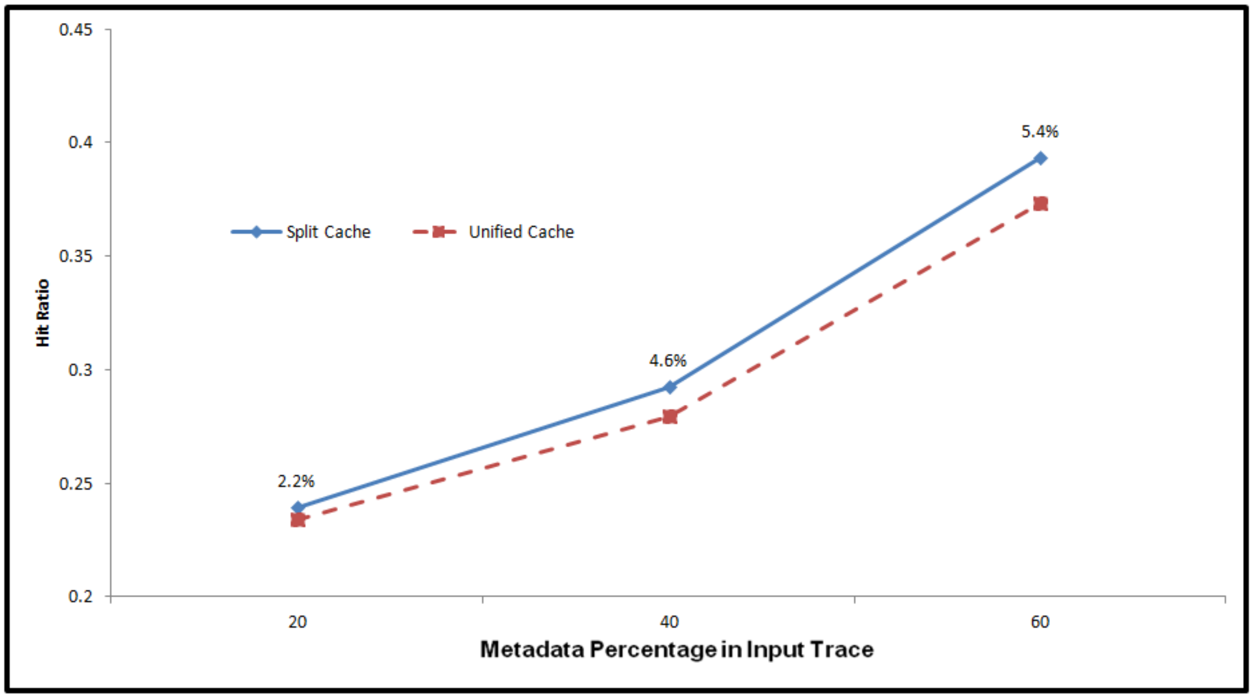

Figure 5.20: Average Hit Ratio with varying Metadata percentage in input trace for Synthetic trace with Validate trace format and Poisson Random Access

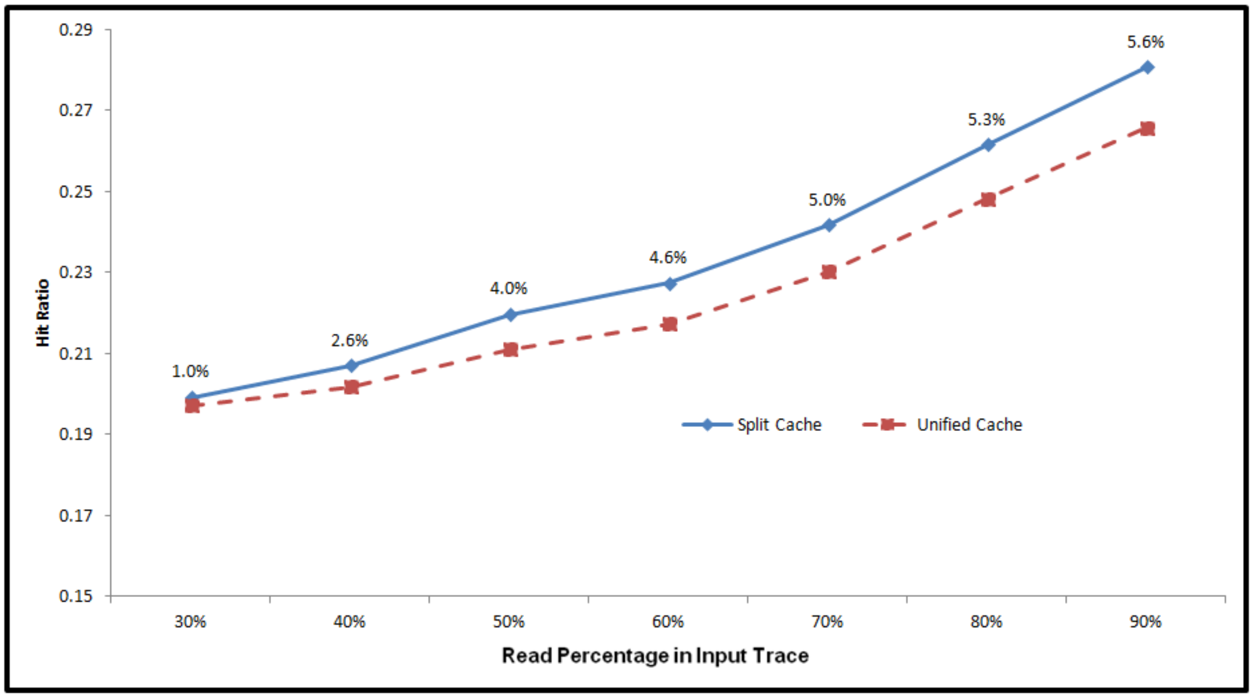

Figure 5.21: Average Hit Ratio with varying read percentage for Synthetic trace with Validate trace format and Poisson Random Access

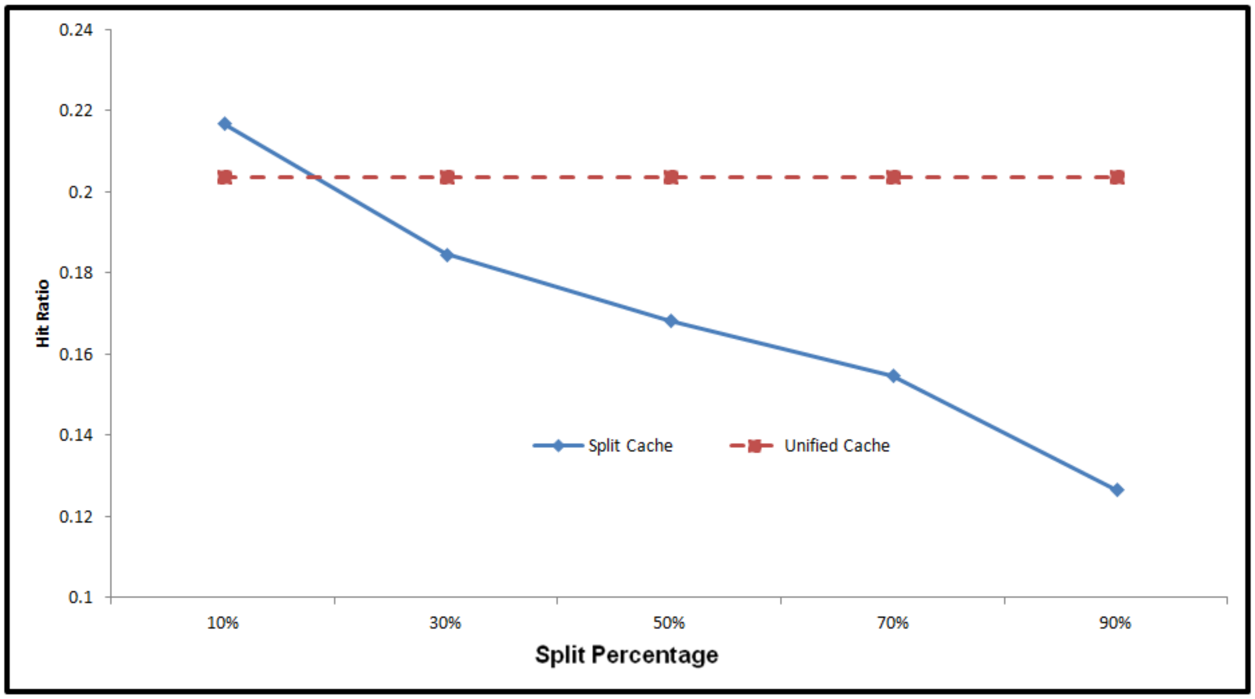

Figure 5.22: Average Hit Ratio with varying Split percentage for Synthetic trace with Validate trace format and Poisson Random Access

5.2 Hit ratio improvement in Split Disk Cache

Figure 5.1 and 5.2 shows result for hplajw trace which has 67% write requests. We observe that compared to unified cache the split cache design does not give a considerable performance improvement with increasing cache size. A maximum improvement of 1% is obtained at 16 MB cache size. But there is no performance degradation because of the split for a write intensive application.

Figure 5.3 - Figure 5.6 depict results for Cello-1990 traces (which have predominantly read accesses), which are taken over two days. We observe that compared to a unified cache the hit ratio improvement rises to 6% with increasing cache size. We note that the 6% improvement over unified cache is observed at 16000 KB or .0153 GB, which is .17% or 1/6th of the disk size of 9.1GB. If the disk were considered half full, it is only 1/3rd of 1% of the used space of disk. Thus, with moderate size caches the split cache performance is beneficial and is cost-effective than a unified cache of 1%.

It is also evident that the hit ratio increases as the data region is increased and metadata region is decreased. with the most benefit coming at a data: metadata region ration of 90:10.

Figure 5.7 through Figure 5.22 represent the results for synthetic traces with varying read, metadata percentages, different split percentages and different access patterns. Figure 5.7 through Figure 5.14 represents the results obtained by using the standard random number generator API for access pattern. Figure 5.15 through Figure 5.18 shows average hit ratios for various combinations of read percentage, metadata percentage in the input trace, split percentage in the disk cache and with different cache sizes. Figure 5.19 through Figure 5.22 shows results with the random numbers generated using Poisson distribution. From the figures we observe that when the total input read requests in the trace is less than 50%, the performance improvement is not much, whereas, when the read requests are 80% of all accesses, the performance gain is between 4.8-6.7%. We note that many applications do have a read to write access ratio of 80:20.

From Figure 5.14, we see that the split cache produces almost same hit ratio as the unified cache when the data to metadata region ratio is 80:20. But when the metadata region is further reduced, there is an improvement in overall hit ratio over the unified cache. Similar results can be seen with Guassian distribution in Figure 5.18 and Poisson distribution in Figure 5.22. From these figures it can be concluded that split cache design can give positive result when the split is in the ratio 90:10 between data and metadata regions.

From Figure 5.13 we observe considerable benefits in the hit ratio of the split cache when the input trace has read requests above 60%. Furthermore, the split cache does not hinder performance for write intensive applications, having read percentages over 30%.

Hit ratio is also dependent on the size of the cache but up to a certain limit as the performance growth due to cache size reaches a limit after a certain size. From Figure 5.11, it is evident that split cache scenario is not beneficial with very small caches size but provides benefits at moderate cache sizes.

Figure 5.12 shows the results for varying metadata percentages in the input trace. We observe that the split cache design is effective when the number of metadata requests in the input trace is high. As the number of accesses to metadata increases, the hit ratio for metadata increases.

All the above results can be seen even with results obtained from Gaussian distribution and Poisson distribution for different access patterns.

As the metadata region is increased, hit ratio with a given cache size reduces. This is because the percentage of data region is reduced and hence the cache misses for data increases. Also, metadata occupies only a small portion of the memory and if allocated more space than required; the additional space is not sufficiently utilized thereby reducing the performance.

Hence split cache implementation, for moderate size disk caches, with a data metadata split region between (90:10..70:30) provides improvements in hit ratio for read intensive applications.

At the disk cache level, small gains provide considerable improvements as the penalty of a miss at this level is very expensive in terms of response time and system performance. For example, consider there are 1000 requests to the disk. If the hit ratio with unified cache is 30%, then 70% requests are sent to the disk. Hence the total time for 1000 I/O operations is equal to 7.03 s [see Table 1]. On the other hand, the split cache with a hit ratio of 36% would take 6.5s. Hence the access time is reduced by approximately 10%. Similarly when the hit ratio is 60% for unified cache and 66% for split cache, the access time is reduced by approximately 15%.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Through this research we have tried to increase the hit ratios for read intensive applications. This research shows that considerable improvements can be achieved when the percentage of reads in an application is more than 60% and metadata reads out of that is around 40%. With this condition, we have evaluated the performance at various split percentages between data and metadata in data region of the cache. From the results we can deduce that there is an optimum point at which the number of total hits in split disk cache is more than the unified cache. This would reduce traffic to physical disk and hence gives greater response time and higher processing speeds when external storage is involved in I/O operations. As a result of this, the processor can be kept more busy and the bottleneck between the processor and the secondary storage system is reduced to an extent.

Hence, with this reserahc we conclude that split in the data region of the disk cache in to data region and metadata region can yield positive results when the split ratio is between 70:30 to 90:10. Furthermore, we would emphasize to have a split ratio of 90:10 as this would not reduce the hit ratios to data read considerably and yet at the same time provide hihger overall hit ratio.

6.2 Future Work

Some of the future work on this research can be as follows:

- To have a dynamic split which can vary based on the access pattern.

- To investigate and provide an optimum split cache scenario considering both reads and writes.

Bibliography

[1] Sanjeev Baskiyar and Chengjun Wang. Split disk-cache architecture to reduce read miss ratio. In *Proceedings of the 9th IASTED International Conference*, volume 676, page 249, 2010.

[2] Scott A Brandt, Ethan L Miller, Darrell DE Long, and Lan Xue. Efficient metadata management in large distributed storage systems. In *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*, pages 290–298. IEEE, 2003.

[3] John S Bucy, Jiri Schindler, Steven W Schlosser, and Gregory R Ganger. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). *Parallel Data Laboratory*, page 26, 2008.

[4] Robert E Fontana Jr, Steven R Hetzler, and Gary Decad. Tape based magnetic recording: Technology landscape comparisons with hard disk drive and flash roadmaps. *dimension*, 1:2, 2011.

[5] Peng Gu, Jun Wang, Yifeng Zhu, Hong Jiang, and Pengju Shang. A novel weighted-graph-based grouping algorithm for metadata prefetching. *Computers, IEEE Transactions on*, 59(1):1–15, 2010.

[6] Bo Hong. Exploring the usage of mems-based storage as metadata storage and disk cache in storage hierarchy. *Storage Systems Research Center, Jack Baskin School of Engineering, University of California at Santa Cruz http://www. cse. ucsc. edu/hongbo/publications/mems-metadata. pdf*, 2003.

[7] Andy Hospodor. Hit ratio of caching disk buffers. In *Compcon Spring'92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers.*, pages 427–432. IEEE, 1992.

[8] Windsor W Hsu and Alan Jay Smith. Characteristics of i/o traffic in personal computer and server workloads. *IBM Systems Journal*, 42(2):347–372, 2003.

[9] Windsor W Hsu and Alan Jay Smith. The performance impact of i/o optimizations and disk improvements. *IBM Journal of Research and Development*, 48(2):255–289, 2004.

[10] Windsor Wee Sun Hsu and Alan Jay Smith. *The real effect of I/O optimizations and disk improvements*. Computer Science Division, University of California, 2003.

[11] Szymon Jankowski. Future of Disk Drives. [Online; created 2012].

[12] Chris Ruemmler and John Wilkes. Unix disk access patterns. In *Proceedings of the Winter 1993 USENIX Technical Conference*, pages 405–420, 1993.

[13] Alan J Smith. Disk cachemiss ratio analysis and design considerations. *ACM Transactions on Computer Systems (TOCS)*, 3(3):161–203, 1985.

[14] Harold S. Stone, J. Turek, and J.L. Wolf. Optimal partitioning of cache memory. *Computers, IEEE Transactions on*, 41(9):1054–1068, Sep.

[15] Andrew S Tanenbaum, Jorrit N Herder, and Herbert Bos. File size distribution on unix systems-then and now. *Operating systems review*, 40(1):100, 2006.

[16] Dominique Thiébaut, Harold S. Stone, and Joel L Wolf. Improving disk cache hit-ratios through cache partitioning. *Computers, IEEE Transactions on*, 41(6):665–676, 1992.

[17] IBM Developer Works. `http://www.ibm.com/developerworks/wikis/display/hpccentral/Data+and+Metadata+-+Separate+or+mixed`, 2012. [Online; created November 2012].

[18] Qing Yang and Yiming Hu. Dcd—disk caching disk: A new approach for boosting i/o performance. In *Computer Architecture, 1996 23rd Annual International Symposium on*, pages 169–169. IEEE, 1996.

[19] Yingwu Zhu and Yiming Hu. Disk built-in caches: evaluation on system performance. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 306–313, Oct.