The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

QUALITY OF SERVICE BENEFITS OF FLASH IN A STORAGE SYSTEM

A Thesis in

Computer Science and Engineering

by

Kanishk Jain

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2008

The thesis of Kanishk Jain was reviewed and approved* by the following:

*Signatures are on file in the Graduate School

ABSTRACT

Virtualized storage systems can be shared by applications (workloads) with substantially different resource requirements, workload patterns, and levels of importance. In such a system, fairness can be defined as a metric that estimates how well the application throughputs match their assigned level of importance (i.e. relative weights). Furthermore, clients paying for storage as a service may expect a certain minimum level of performance from the virtualized system. In general, for a storage system, there is a fundamental tradeoff between the goal of achieving fairness and the goal of maximizing overall performance. Thus in a virtualized storage system, an important quality of service (QoS) goal is to try to optimize fairness, while also ensuring that the overall performance stays above a minimum acceptable threshold.

Storage performance is notoriously difficult to estimate in a storage system and varies significantly with workload characteristics. Due to the mechanical nature of a disk, a random workload achieves a lower throughput, and experiences a higher response time, than a sequential workload. As a result of this behavior, an algorithm that aims to achieve the aforesaid QoS goal, results in asymmetric fairness between a random and a sequential stream.

Flash is a storage device that has the potential of making storage systems independent of the effects of randomness in workloads, and thus of making the performance of virtualized storage systems more predictable. The scope of this thesis is to investigate how flash can be used to augment virtualized storage systems, in order to improve fairness, while also achieving sufficient minimum performance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my adviser, Dr Bhuvan Urgaonkar, for his guidance and support right through my Masters. He was a mentor, a guide and a friend, and each meeting with him opened my mind to new possibilities, and helped me grow as a computer scientist. I would also like to thank Dr Trent Jaeger for introducing me to the process of research in my first semester and for helping me right through.

Special thanks to Youngjae Kim and Aayush Gupta for sharing their knowledge of working with flash, and for helping me along the way.

I would like to take this opportunity to thank my parents and my sister for their endless support for me over the years. Full credit to them for helping me come this far. A special mention for my friends, Dr Dhiraj Joshi and Nandini Bhardwaj, who inspired me to follow my dreams.

Chapter 1

Introduction

1.1 Storage Virtualization

In an enterprise environment, storage consolidation is being increasing used to create shared pools of storage resources [**3**]. Storage virtualization consists of making these shared pools appear as a single storage entity. Apart from the economies of sharing, increased scalability and increased resource utilization, a major advantage of the aforesaid virtualization is the ease of centralized management. It also allows companies to provide storage as a service to clients, wherein the clients pay per byte or per share of bandwidth and expect certain reliability and access guarantees, for example bounds on response time and throughput.

1.2 Performance Virtualization

In a virtualized storage system, providing guarantees on performance is a significant challenge. A particular request may be serviced by potentially different components of a heterogeneous consolidated storage system, and thus it is difficult to estimate performance parameters such as throughput (bandwidth) and response time (latency) for the overall virtualized system. Even for a homogeneous system, the performance of different components may vary due to differences in age and lack of standardization in production.

To complicate matters further, storage capacity (i.e. bandwidth) is notoriously difficult to estimate in a storage system and varies significantly with workload characteristics [**2**]. This is because of the mechanical nature of disks, which results in delays such as seek time and rotational latency. These delays are dependent on the characteristics of the workload, and increase with an increase in randomness in the workloads. (A "random" workload can be defined as one which incurs mechanical delays, such as seek time.)

Virtualized storage systems might be shared by applications (workloads) with substantially different resource requirements, workload patterns, and levels of importance (depending on the amount paid by clients). A performance virtualization scheme must provide isolation among different competing workloads. Isolation implies the providing of performance guarantees to an application irrespective of the behavior of other applications.

Clients paying for storage as a service may expect a certain minimum level of service (performance) from the virtualized system. Thus algorithms such as reservation based fair queuing (RBFQ) [**6**] aim to control the allocation of resources in a way that meets the minimum resource requirements for each client application, while allocating the remaining bandwidth based on the importance levels (i.e. relative weights) of each application, all in the face of dynamically varying workload patterns.

Most performance virtualization schemes [**1**][**2**][**3**][**4**][**5**][**6**][**7**][**8**][**9**][**10**] are easily implementable as interposed request scheduling schemes, that treat the storage server as a black box, and control resources externally. This organization is illustrated in Figure **1.**

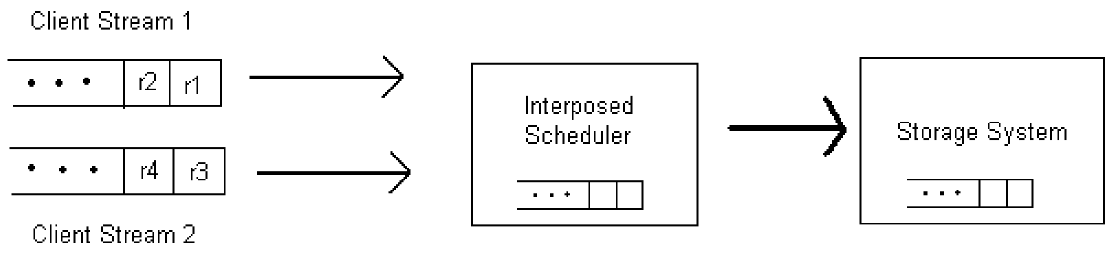


Figure **1**: Interposed Storage Request Scheduling

1.3 Fairness

In a virtualized storage environment, fairness can be defined as a metric that estimates how well the application throughputs match their assigned weights [**5**]. In general, for a storage system, there is a fundamental tradeoff between the goal of achieving fairness and the goal of maximizing overall performance (throughput) [**8**]. Almost all performance virtualization schemes attempt to optimize this tradeoff [**1**][**2**][**4**][**5**][**7**][**8**][**9**][**10**].

An important quality of service goal for a storage system can be stated as follows:

Maximize *fairness F*,

such that *Throughput(F)* > *Throughput_threshold*

where

F is a measure of the fairness achieved

Throughput(F) if the throughput achieved while achieving fairness F

Throughput_threshold is the minimum acceptable throughput

The above construct simply states that one must try to optimize fairness, while also ensuring that the overall performance stays above a minimum acceptable threshold.

Reservation based fair queuing (RBFQ) [**6**] is an algorithm that addresses the challenges of performance virtualization under varying capacity. It aims to solve the quality of service problem in the form stated above. However, such an algorithm can exhibit asymmetric fairness for different workload patterns, as explained below.

To demonstrate the aforesaid problem, consider the simulation results in Table **1**.

Table **1**: Disk under varying weight ratios, using RBFQ

Desired Weight Ratio (Sequential:Random)	Throughput (Requests/sec)	**Achieved** Weight Ratio (Sequential:Random)
1:1	448.31	**1.0:1**
2:1	635.90	**1.9:1**
5:1	847.18	**4.3:1**
10:1	1461.91	**8.3:1**
25:1	2379.10	**18.9:1**
100:1	3698.37	**54.5:1**
1:2	360.70	**1:1.8**
1:5	313.12	**1:3.3**
1:10	280.04	**1:4.6**
1:25	263.92	**1:6.0**
1:100	258.99	**1:7.1**

The simulation consists of directing a synthetic sequential workload and a synthetic random workload to a simple storage system consisting of a single disk, using RBFQ as an interposed scheduler (in an organization similar to Figure **1)**. In general, due to the mechanical nature of a disk, a random workload achieves a lower throughput than a sequential workload, because it incurs higher seek time and rotational latency. RBFQ [**6**] works as follows: when the observed bandwidth (throughput), split according to the weight of each workload, is enough to satisfy the minimum requirement of each workload, the algorithm splits the bandwidth in the exact ratio of the desired weights. However, when the observed bandwidth, split in accordance with the weights, is not enough to satisfy the minimum requirement of each workload, the algorithm first tries to ensure that the minimum requirements are met (by not splitting the total bandwidth in the exact ratio of the desired weights). Hence, the achieved weight ratios can deviate from the desired weight ratios each time the observed bandwidth falls below the total minimum requirement of all workloads (i.e. the throughput threshold). The deviation of the achieved weight ratios from the desired weight ratios can be termed as unfairness. The working of RBFQ has been shown pictorially in Figure **2**.

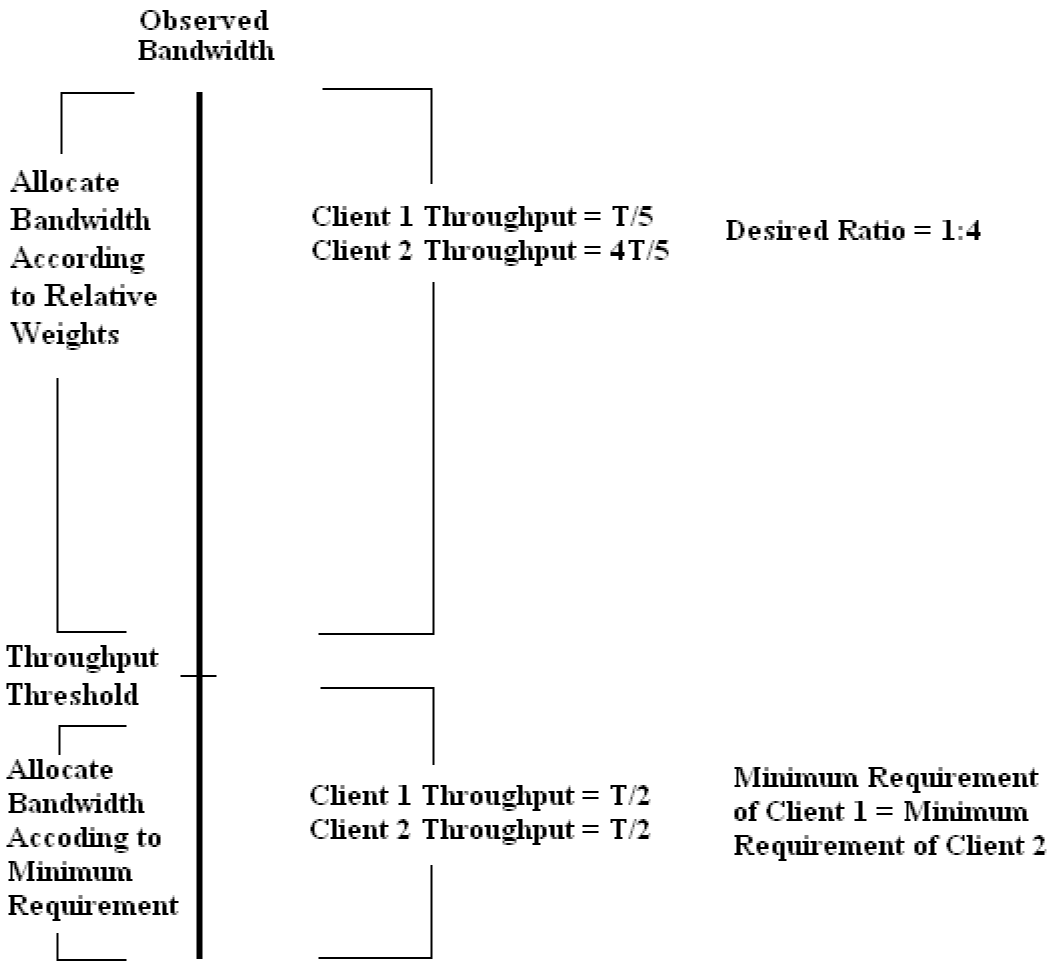

Observed Bandwidth
Allocate Bandwidth According to Relative Weights
Client 1 Throughput = T/5
Client 2 Throughput = 4T/5
Desired Ratio = 1:4
Throughput Threshold
Allocate Bandwidth Accoding to Minimum Requirement
Client 1 Throughput = T/2
Client 2 Throughput = T/2
Minimum Requirement of Client 1 = Minimum Requirement of Client 2

Figure **2**: Bandwidth Allocation using RBFQ

For the simulation in Table **1** , the minimum requirement of each application has been fixed at 30 Requests/sec. At a 1:1 desired weight ratio, the overall bandwidth, after being split equally between the two workloads, is enough to satisfy the minimum requirement of both workloads and thus this weight ratio is achieved exactly (i.e. perfect fairness is achieved). As the relative weight of the sequential workload increases, the overall throughput increases (because the percentage of sequential requests becomes higher in the mixed workload). Thus, the desired weight ratio can still be achieved while guaranteeing minimum service to both workloads. However, when the desired weight ratio becomes very high, then, though the overall throughput is relatively higher, the very small throughput share of the random workload falls below its minimum requirement. Hence the achieved weight ratio cannot keep up with the desired weight ratio for very high values, leading to unfairness at very high ratios.

Now consider the case when the random workload has a higher weight that the sequential workload. As the weight of the random workload increases, the overall throughput decreases, and thus soon the bandwidth allocated to the workloads (especially the sequential workload, which gets a smaller share) start falling below their minimum requirement. Hence, in such as case, the achieved weight ratio cannot match the desired weight ratio, even for small ratios. *Thus, randomness in the workloads exacerbates the unfairness problem.*

While the unfairness problem due to very high weight ratios is caused by the basic nature of an algorithm such as RBFQ, the exacerbation of the unfairness due to randomness in the workloads is a creation of the storage system, and is thus avoidable by changes to the storage system.

The problem has been illustrated in Figure **3**. The figure has been drawn to a log scale (base 2) (i.e. the values plotted are the log of the (sequential weight)/(random weight) ratio). Thus, the origin represents a 1:1 weight ratio between the sequential and the random workload, the positive x-axis represents the cases where the sequential workload has a higher weight, and the negative x-axis represents the cases where the random workload has a higher weight. Note that in the random dominant parts, the achieved weight ratio is not able to match the desired weight ratio. Thus the fairness graph between a sequential and a random workload is asymmetric for a disk.

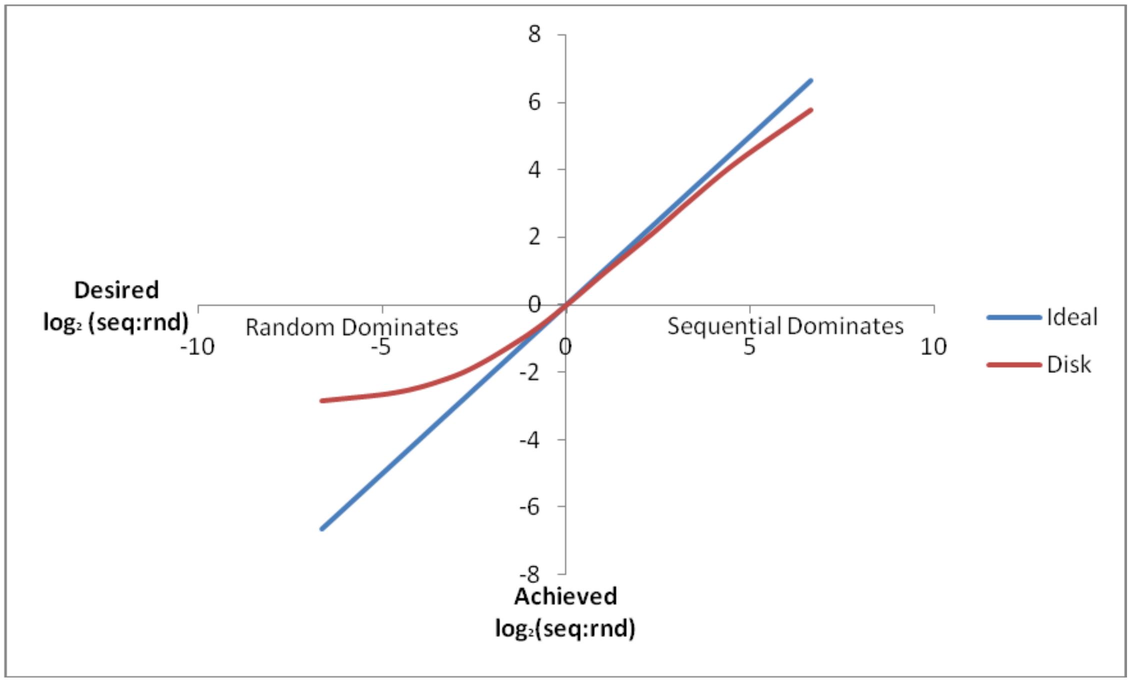


Figure **3**: (Log scale) Disk under varying weight ratios, using RBFQ

1.4 Response Time

Response time is a performance metric which is defined as the time taken for a request to complete, after it has been issued to the storage system. It is a dual of the throughput of a storage system (and is inversely related to it). Figure **4** shows a time series of the average response times of the sequential and the random workloads from the simulation of Table **1**. Note that the sequential workload experiences a much lower average response time than the random workload.

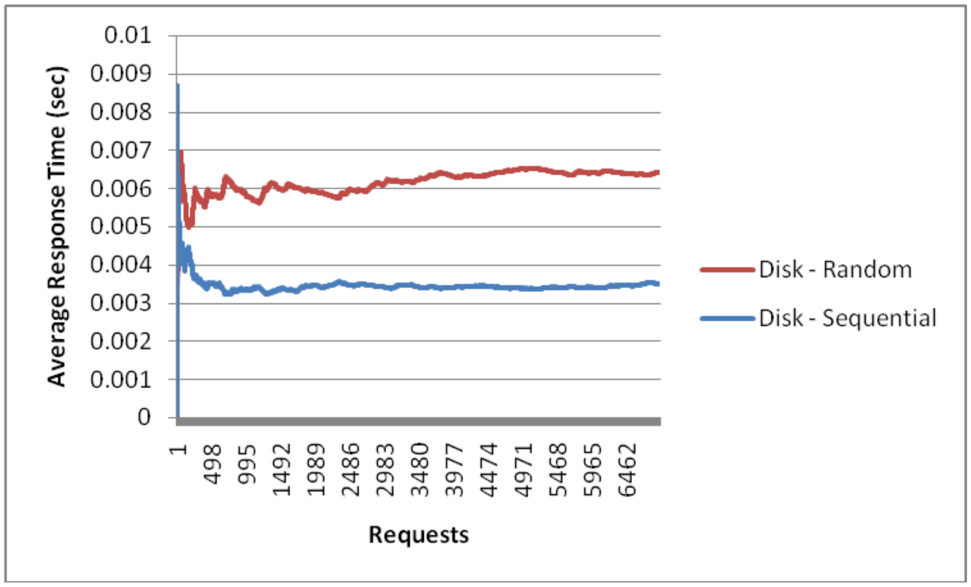


Figure **4**: Time Series of the Average Response Times for synthetic random and sequential workloads, at a disk, under RBFQ

1.5 Flash in a Virtualized Storage System

Flash [**15**][**16**] is a solid state storage device that has the potential of making storage systems independent of the effects of randomness in the workloads, thus making the performance of virtualized storage systems more predictable. The scope of this thesis is to investigate how flash can be used to augment virtualized storage systems in order to improve fairness and predictability, while also achieving sufficient minimum performance.

1.6 Organization

The rest of this thesis is organized as follows. Chapter 2 documents the relevant related work. Chapter 3 compares a disk and a flash, and illustrates the idea of a hybrid store which uses both, a disk and a flash. It also presents an offline scheme (for read requests only) which can be used to organize data in the hybrid store. Chapter 4 extends the aforesaid offline scheme to include write requests, by analyzing a basic flash model. Chapter 5 goes on to develop an online scheme, based on the aforesaid flash model. Finally, Chapter 6 concludes the thesis.

Chapter 2

Related Work

Storage consolidation has the advantages of the ease of centralized management, economies of sharing and increased resource utilization. It also allows storage to be provided as a service by companies, where clients can pay per byte of storage, and expect certain reliability and access guarantees. These guarantees are usually expressed in terms of throughput (or bandwidth) and/or latency (or response time) constraints.

One of the main challenges in the above consolidated scenario is to provide isolation among different competing workloads. Isolation implies the providing of performance guarantees to an application irrespective of the behavior of other applications.

Another consideration for most performance virtualization schemes is whether or not they are work conserving (i.e. a system implementing a work conserving scheme will not idle if requests are pending)

This chapter reviews some existing performance virtualization schemes.

2.1 Proportional Sharing in Disks

Traditional disk access is best-effort, and there are no timing guarantees [**10**]. Acceptable performance is achieved when the disk is not overloaded. However, when the demand for disk bandwidth exceeds the supply, *all* applications may experience performance degradation. The aim of isolation is thus to provide a mechanism that provides performance guarantees to an application irrespective of the behavior of other applications. One way of achieving this, is to proportionally share the instantaneous disk bandwidth, i.e. to split the available disk bandwidth between applications in proportion to their relative importance (weights).

Specifying a proportional share of disk bandwidth may seem intuitive. However, disk bandwidth is not constant: service times vary depending upon the initial position of the read-write head, the position of the requested data on the disk, the low-level disk scheduling algorithm, etc. Translating a bandwidth requirement into low-level disk operations is a complicated task. Alternatively, specifying a proportional share by a fixed time-slice at the storage (instead of bandwidth) may result in different amounts of data being retrieved per time-slice. Thus the specification of time at the disk is not intuitive from the user’s perspective. The Cello [**1**] scheduler presents two methods of accounting, either by size or time. Some schedulers allow specification of shares in terms of number of requests. However, requests may also vary in size and service time.

There is a *fundamental conflict* between the goal of achieving proportional-share allocations and the goal of maximizing overall disk throughput [**8**]. Thus, most proportional sharing algorithms for disks, *trade-off* overall throughput in order to achieve proportional sharing of bandwidth.

2.2 Cello [1]

- Cello is a QoS-aware scheduler designed specifically to address *mixed workloads* (for example, a workload mix can include real-time, best-effort, and interactive workloads).
- It is a two-level disk scheduling architecture: one level is *application independent* and the other is *application specific*. The two levels schedule at a *different granularities* (the class-independent scheduler does coarse-grain allocation of bandwidth to application classes, while the class-specific schedulers control the fine-grain interleaving of requests from the application classes).
- It isolates application classes.
- It is work conserving. It reassigns *idle bandwidth*.
- It is a storage specific scheduler and allows *seek optimization* at certain points while scheduling.
- It allows proportionate bandwidth allocation to be configured as either proportionate time allocation (which may not be intuitive from the users' perspective) or proportionate byte allocation.
- It trades-off overall throughput to achieve fairness.

2.3 pClock [3]

- pClock is based on arrival curves (i.e. the leaky bucket model of computer networks). This can be seen to be closely related to service curve based methods which *decouple* *bandwidth (throughput) allocation and latency (response time) requirements* (in fair queuing schedulers such as Cello, latency cannot be controlled independently from bandwidth, since there is only one parameter to adjust both throughput and latency. Thus for those schedulers, latency incurred by a request is inversely related to its bandwidth).

- pClock requires certain capacity constraints to be satisfied for admission control into the storage system. It uses a real time tagging mechanism (which is the key for latency control).
- It isolates different streams of requests. Handles bursts too (by using the leaky bucket model).
- It is work conversing and uses spare capacity to service active flows without penalizing them for using spare capacity.
- The system capacity (IOs/second) is estimated at the time of admission control. This capacity can vary in a storage server (especially with the variation in workloads) and there are *no experiments to show how the variation of this capacity affects throughput and response time of pClock*.
- The scheduler does not have any storage specific characteristics, and thus could be used for any service. Also, since it does not consider the variation of storage capacity, it may not perform very well with variations in workload characteristics. It is thus not optimized for storage.

2.4 AVATAR [2]

- AVATAR is part of a two level scheduling framework: the higher level (SARC) does *rate control*, while the lower level (AVATAR) uses EDF scheduling to meet *latency* requirements. Thus the framework decouples bandwidth and latency requirements.
- AVATAR uses feedback based control (the Monitor, Controller, Dispatcher architecture) to avoid the need of an accurate performance model of a storage system (a performance model for a storage system is very difficult to achieve). A drawback of proportional bandwidth sharing schemes is that they require such a performance model to estimate the service time of an individual I/O request. AVATAR uses feedback control to dynamically regulate the number of requests dispatched from the EDF queue to the storage utility, where they may get reordered for better efficiency.
- The higher level (SARC) of the two level scheduling framework isolates the workloads.
- The two level scheduling framework is work conserving. A “spareness status” of the storage utility is sent from the lower level (AVATAR) to the upper level (SARC) to allow the rate to be controlled accordingly.
- Storage specific characteristics are accounted for using the feedback based control: after the low level AVATAR scheduler dispatches requests to the storage utility, the requests may be reordered. However as far as AVATAR is concerned the storage server is a black box.
- The Monitor of AVATAR continuously collects IO statistics from the underlying system.

- The Controller determines the queue threshold at the storage utility for the current time window.
- According to the authors, the most critical parameter set by AVATAR is the threshold of the storage utility queue length (L0). This can be interpreted as a concurrency parameter which represents the number of outstanding requests at the storage device.

2.5 SFQ(D) [4]

- SFQ(D) is a proportionate bandwidth allocation method: it is a “depth-controlled” extension of Start-time Fair Queuing (SFQ).
- A depth parameter (D) has been incorporated, which controls the number of requests outstanding at the server. When a request completes, the scheduler selects the next queued request according to its scheduling policy, and dispatches it to maintain a concurrency level of D within the server.
- The value of D represents a tradeoff between server resource utilization and scheduler fairness. For instance, a larger D may allow better multiplexing of server resources, but it may impose a higher waiting time on an incoming client request.
- Different policies can be defined to configure D or adapt it (a similar adaptation is done in [5])
- The decrease in fairness has been shown to be proportional to D.
- SFQ(D) scheduling is not specific to storage and is thus not optimized for a storage system. However the policy configuring/adapting D can be storage specific.

2.6 Adaptive Scheduling [5]

- Proportional share disk schedulers tradeoff overall throughput (performance) for fairness. An analysis of this tradeoff has been provided by the authors.
- The metrics defined to analyze the aforesaid tradeoff are:
    - *Efficiency* denotes the ratio of the actual system throughput to the throughput attained when the applications are run without interference
    - *Fairness* refers to how well the application throughputs match their assigned weights
- Two factors have been identified which impact both I/O efficiency (performance) and fairness guarantees:
    - Concurrency Bound (D) : the number of outstanding scheduled requests at the storage server
    - Batch Size (Gi) : Number of I/O requests scheduled from one application in one round
- An adaptive algorithm adapts the parameters D and Gi to optimize the tradeoff between performance and fairness.
- The main conclusions of the analysis are:
    - A random workload mix (RRR) is unaffected by batching parameters and its efficiency is solely dependent on the bounded concurrency (D)
    - Batching helps workloads with locality and their performance improves as we increase the batch size

  - It is possible to get high efficiency with small values of D. This is important, since setting D to a large value causes fairness to deteriorate significantly.
  - However it is not clear whether high efficiency with small values of D is achievable for random workloads too.

2.7 A note on Concurrency and the use of Flash

The number of outstanding requests at a storage system is represented by the concurrency parameter (D in the case of SFQ(D) [**4**] and adaptive scheduling [**5**], and L0 in the case of AVATAR [**2**]). Increasing the concurrency is shown to benefit throughput [**2**][**4**], especially in the presence of random workloads [**5**], but results in increased latency [**2**][**4**] and decreased fairness [**4**][**5**]. Since flash benefits random read-dominated workloads (as explained in later chapters), the use of a flash device in a storage system can be seen as an alternative to increasing the concurrency, without decreasing fairness.

2.8 Proportional Share Scheduling for Distributed Storage Systems [7]

- This scheduler provides differentiated quality of service to clients in a distributed environment.
- It adapts a fair queuing algorithm (SFQ(D) [4]) to a distributed server environment. Providing performance guarantees in distributed storage systems is more complex because clients may have different data layouts and may access their data through different access nodes, yet the performance guarantees required may be global.

- It allows the specification of different performance goals e.g., per storage node proportional sharing (sharing the bandwidth of each node proportionally), total service proportional sharing (sharing the total bandwidth of the distributed storage system proportionally) or hybrid proportional sharing.
- The fair queuing algorithm (SFQ(D)[4]) runs at each node (i.e. each distributed server). However, it has been modified to incorporate a delay function. The delay function corresponds to the amount of service the client gets at other nodes.
    - The choice of the delay function allows the specification of different performance goals
    - The delay function can be zero, in which case it reduces to the scenario where normal SFQ(D) runs at each node, and results in per-node proportional sharing. This allows each client to get certain minimum service at each node, but may not share the overall bandwidth of the distributed system proportionally.
- The scheduler is work-conserving: no resource is left idle if there is any request waiting for it. However, it is also shown that a work-conserving scheduling algorithm for multiple resources (i.e. nodes in the distributed system) cannot achieve proportional sharing in all cases. Thus a hybrid mechanism, between system-wide proportional sharing and per-node proportional sharing, has been suggested which allows total proportional sharing when possible while ensuring a minimum level of service on each node for all clients.

2.9 Reservation Based Fair Queuing [6]

- Capacity (i.e. bandwidth) of a storage system varies significantly with workload characteristics.
- Reservation based fair queuing (RBFQ) aims to control the allocation of resources in a way that meets minimum resource requirements for each client application (when possible), while allocating remaining bandwidth based on the importance levels (i.e. relative weights) of each application, all in the face of dynamically varying workload patterns.
- Apart from the weight of each stream, the algorithm also takes the minimum service (performance/throughput) requirement of each stream as an input. *It does not need to know the capacity of the storage device for its functioning* (capacity is very difficult to estimate in a storage system).
- The algorithm works as follows: When the observed bandwidth (throughput), split according to the weight of each workload, is enough to satisfy the minimum requirement of each workload, the algorithm splits the bandwidth in the exact ratio of desired weights. However, when the observed bandwidth, split in accordance with the weights, is not enough to satisfy the minimum requirement of each workload, the algorithm first tries to ensure that the minimum requirements are met (by not splitting the total bandwidth in the exact ratio of the desired weights). Hence the achieved weight ratios can deviate from the desired weight ratios, each time the observed bandwidth falls below the total minimum requirement of all workloads. The working of RBFQ had been shown pictorially in Figure **2.**

- It keeps switching between two phases, a local phase which ensures the meeting of minimum resource requirements (if possible) and a global phase which allocates the remaining bandwidth according to the weight of each application.
- It is work conserving.

Chapter 3

Hybrid Store

Flash [**15**][**16**] is a solid state storage device that has the potential of making storage systems independent of the effects of randomness in workloads, thus making the performance of virtualized storage systems more predictable. This chapter develops the intuition on how flash can be used to augment virtualized storage systems in order to improve fairness, while also achieving sufficient minimum performance.

3.1 Flash Properties

For now, all one needs to know about a flash device [**15**][**16**] is that it has no moving mechanical parts (and thus no mechanical delays such as seek time and rotational latency), and that it performs similarly for random and sequential read requests [**16**]. It also has the same block based interface as a disk.

3.2 Fairness Comparison between a Disk and a Flash

Consider the problem described in Chapter 1. An algorithm such as reservation based fair queuing (RBFQ) [**6**] which aims to address the challenges of performance virtualization under varying capacity, can exhibit asymmetric fairness under different workload patterns. The simulation results in Table **1** illustrated this problem. Now consider the same simulation setup, with a flash device being used instead of the disk.

Table **2** is an extension of Table **1**. It compares the fairness and the performance (throughput and response time), of a flash with that of a disk, under RBFQ [**6**].

Table **2**: Disk and Flash under varying weight ratios, using RBFQ

	Disk			**Flash**		
Desired Weight Ratio (Seq:Rnd)	Throughput (Requests per sec)	Average Response Time (sec)	**Achieved** Weight Ratio (Seq:Rnd)	Throughput (Requests per sec)	Average Response Time (sec)	**Achieved** Weight Ratio (Seq:Rnd)
1:1	448.31	0.002231	**1.0:1**	5841.12	0.000171	**1.0:1**
2:1	635.90	0.001573	**1.9:1**	5841.12	0.000171	**2.0:1**
5:1	847.18	0.001180	**4.3:1**	5841.12	0.000171	**4.9:1**
10:1	1461.91	0.000625	**8.3:1**	5841.12	0.000171	**9.5:1**
25:1	2379.10	0.000360	**18.9:1**	5841.12	0.000171	**22.0:1**
100:1	3698.37	0.000220	**54.5:1**	5841.12	0.000171	**64.9:1**
1:2	360.70	0.002772	**1:1.8**	5841.12	0.000171	**1:2.0**
1:5	313.12	0.003225	**1:3.3**	5841.12	0.000171	**1:4.9**
1:10	280.04	0.004034	**1:4.6**	5841.12	0.000171	**1:9.5**
1:25	263.92	0.004070	**1:6.0**	5841.12	0.000171	**1:22.0**
1:100	258.99	0.004089	**1:7.1**	5841.12	0.000171	**1:64.9**

The simulation setup and assumptions of Table **2** are as follows:

1. The simulation uses the DiskSim 3.0 simulation environment [**11**]. This disk model used is IBM36Z15_15K, which is part of the standard DiskSim 3.0 distribution. The flash device has been simulated using FlashSim [**12**], a flash simulator, which has been plugged into the DiskSim 3.0 simulation environment. The organization is similar to Figure **1**.
2. The algorithm used is Reservation Based Fair Queuing (RBFQ) [**6**]. Its working was summarized in Chapter 1 (see Figure **2**) and explained in Chapter 2.

3. Two synthetically generated streams (workloads) are directed to the storage system. One of them is a purely sequential stream (i.e. probability of sequential requests is 1.0) and the other is a purely random stream (i.e. probability of sequential requests is 0.0). The simulation aims to observe the variation in achieved weight ratios and achieved throughput (and response time) as the relative weights of the two streams change. The weight ratios, which have been evaluated, range from 100:1 (very high priority to the sequential workload) to 1:100 (very high priority to the random workload).
4. All requests are read requests.
5. All requests are of same size (512 B).
6. All streams are constantly backlogged (i.e. the next request from a stream is always available).
7. The flash device used is large enough to service both streams completely (i.e. working set size of both streams is equal to the size of the flash). Also, the flash device is extremely fast. However, the throughput of the flash for a sequential workload is lower than the sequential throughput of the disk.

Under the aforesaid simulation assumptions, flash shows constant performance, irrespective of sequentiality.

For the simulation in Table **2**, the minimum requirement of each application has been fixed at 30 Requests/sec. At a 1:1 desired weight ratio, the overall bandwidth (throughput), after being split equally between the two workloads, is enough to satisfy the minimum requirement of both workloads and thus this weight ratio is achieved exactly (i.e. perfect fairness is achieved). As the relative weight of the sequential workload increases, the overall throughput for a disk increases (because the percentage of sequential requests becomes higher in the mixed workload). Thus, the required ratio can still be achieved while guaranteeing minimum service to both workloads. However when the ratio becomes very high, then, though the overall bandwidth is relatively higher, the very small bandwidth share of the random workload falls below its minimum requirement. Hence the achieved weight ratio cannot keep up with the desired weight ratio for very high values, leading to unfairness at very high ratios. This unfairness is caused by the basic nature of the algorithm, *and it is the same for both the disk and the flash.*

Now consider the case when the random workload has a higher weight that the sequential workload. As the weight of the random workload increases, the overall throughput for a disk decreases, and thus soon the bandwidth allocated to the workloads (especially the sequential workload, which gets a smaller share) start falling below their minimum requirement. Hence the achieved weight ratio cannot keep up with the desired weight ratio even for small weight values. Thus randomness in the workloads exacerbates the unfairness problem for a disk. Since flash is seemingly independent of randomness in workloads, it does not exhibit this exacerbation of the unfairness problem. The achieved weight ratios, while using flash are symmetric for the sequential and the random workload. *Hence flash can be used to solve this aspect of the fairness problem*.

The above behavior has been illustrated in Figure **5** . The figure has been drawn to a log scale (base 2). Thus, the origin represents a 1:1 weight ratio between the sequential and the random workload, the positive x-axis represents the cases where the sequential workload has a higher weight, and the negative x-axis represents the cases where the random workload has a higher weight. *Note that flash exhibits symmetric performance for random and sequential dominant parts.*

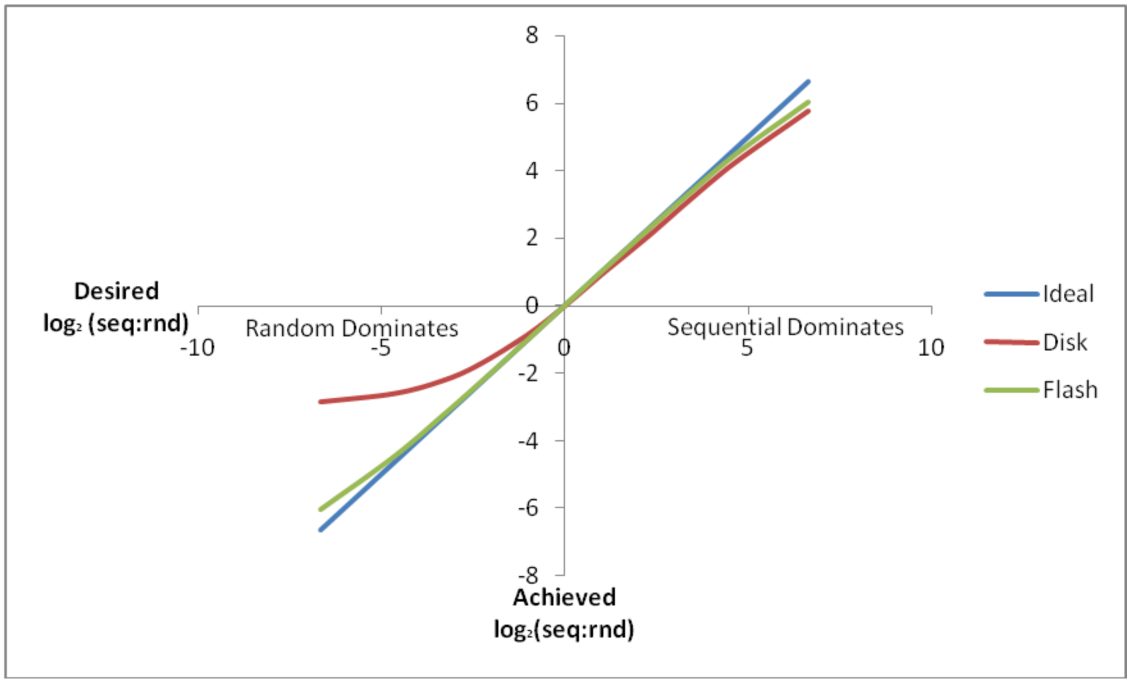


Figure **5**: (Log scale) Flash under varying weight ratios, using RBFQ

Also note that, while the throughput and response time of the disk vary with workload characteristics, the flash throughput and response times are constant under the simulation assumptions. Hence flash could also be used to make the storage system more *predictable* by giving better bounds on throughput and response time, irrespective of workload characteristics.

However, *the aforesaid simulation assumptions represent an ideal scenario* because:

- Flash is expensive and thus the size of the flash device in a real system may not be large enough to be able to completely service all workloads.

- The above simulation considers only read requests.

Hence a *combination of a flash and a disk* could be used to exploit these properties.

3.3 Hybrid Store (Combination of a Disk and a Flash)

Hybrid Store [**13**] (as suggested by Dr Bhuvan Urgaonkar and Youngjae Kim) consists of a combination of a hard disk and a flash device. The two devices have complementary properties and can be intelligently used together to improve overall performance and fairness. The logical organization of the hybrid store used in the simulations in this thesis is shown in Figure **6**. The logical organization can be implemented either by using the flash as separate block device, or as an on-disk flash. The schemes described in this thesis can be implemented in a separate scheduler interposed before the I/O driver, or within the I/O driver itself. In the following simulation evaluations, to ensure that the improvements due to the hybrid store are not due to parallelism (resulting from the use of two devices instead of one), a particular request is issued to the I/O driver only when the previous request completes.

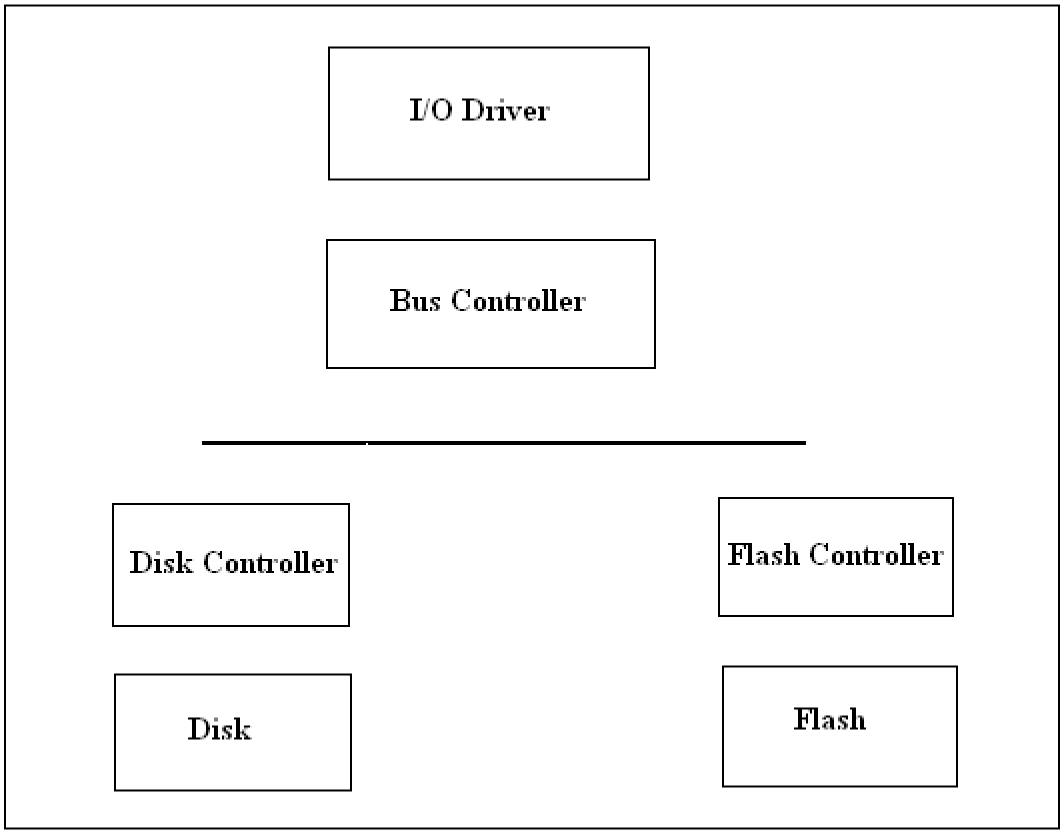


Figure **6**: Logical Organization of the Hybrid Store

3.3.1 Offline Classification in the Hybrid Store – Read Case Analysis

The challenge, central to the hybrid store architecture, is to decide the organization of the data on the two devices, the flash and the disk. The data on each device could be exclusive or could be replicated. Furthermore, there may be limitations arising out of the finite size of the devices (for example, due to the high cost of a flash device, one expects its size to be much smaller than the disk).

To develop the ideas of a hybrid store, consider an offline scheme which makes two passes over an input stream. An initial “preprocessing” pass allows the scheme to study the properties of the stream, allowing it to decide a way to statically map parts of the stream to either device. The second pass is made while actually running the simulation.

Since our knowledge of flash so far has been limited to the fact that it performs equivalently for sequential and random read requests, assume that the input streams consist entirely of read requests (write requests have been handled Chapter 4).

3.3.1.1 Preprocessing

Consider a scheme that partitions contiguous sectors of the logical address space into fixed size, non-overlapping clusters of sectors, and exclusively maps each cluster onto the flash or the disk. Once a cluster has been mapped to a particular device during the preprocessing phase, all read requests directed to the cluster are directed to that device during the simulation phase. If a request goes beyond the end of a cluster, and the next cluster has been mapped to a different device, the request must be split and sent to the two devices respectively.

During the preprocessing phase, certain statistics are computed for the stream. These include the following:

- Sequentiality of requests to a cluster: A request is defined as sequential when it is made to a cluster, which is the same as, or adjoining, the cluster to which the previous request was made.

- Number of requests to a cluster

The offline scheme is based on the fact that flash is better than a disk at handling random read requests (as illustrated in the previous section) and thus clusters with sequentiality below a particular threshold should be mapped to the flash. Since the flash has a finite size, there may be a need to select among multiple possible clusters. One criterion could be the number of requests made to the cluster. Thus, *random clusters with more accesses should be given preference while mapping to flash.*

3.3.1.2 Simulation for the Offline Scheme for reads only

The aforesaid offline scheme was implemented in a simulation setup similar to the simulation environment of Table **2**. However the major difference is the following:

- Each stream is too large to be serviced entirely from the flash (Size of the flash used is 256 MB, while the range of logical addresses of each synthetic stream is now 2GB)

As before, the simulation consists of read requests only. The results of the simulation are shown in Table **3**. Note that as the weight of the random stream increases, more percentage of requests are serviced from the flash, which shows that mostly the clusters where more random requests were directed have been mapped to the flash by the offline scheme.

Table **3**: Offline Classification in the Hybrid Store (for read oriented workloads only)

	Only Disk		**Disk + 256 MB Flash**		
Desired Weight Ratio (Seq:Rnd)	Throughput (Requests/sec)	**Achieved** Weight Ratio (Seq:Rnd)	Throughput (Requests/sec)	% of requests serviced from flash	**Achieved** Weight Ratio (Seq:Rnd)
1:1	265.17	1.0:1	1156.43	**39.37**	1.0:1
2:1	340.25	1.8:1	1520.75	**27.09**	1.9:1
5:1	601.97	4.0:1	2245.22	**14.46**	4.7:1
10:1	915.46	7.5:1	2914.55	**9.24**	9.0:1
25:1	1601.71	16.9:1	3886.81	**4.42**	20.8:1
100:1	2936.24	48.7:1	4453.16	**2.04**	59.1:1
1:2	226.38	1:1.6	958.23	**51.39**	1:1.9
1:5	203.34	1:2.8	804.09	**63.17**	1:4.2
1:10	191.73	1:3.6	748.43	**68.39**	1:7.1
1:25	184.60	1:4.3	718.40	**72.04**	1:12.0
1:100	180.28	1:4.8	682.72	**73.88**	1:17.8

The results are illustrated in Figure **7**. The fairness of the hybrid store in the random dominant region deviates more from the desired fairness (as compared to the sequential dominant region) because of the finite size of the flash.

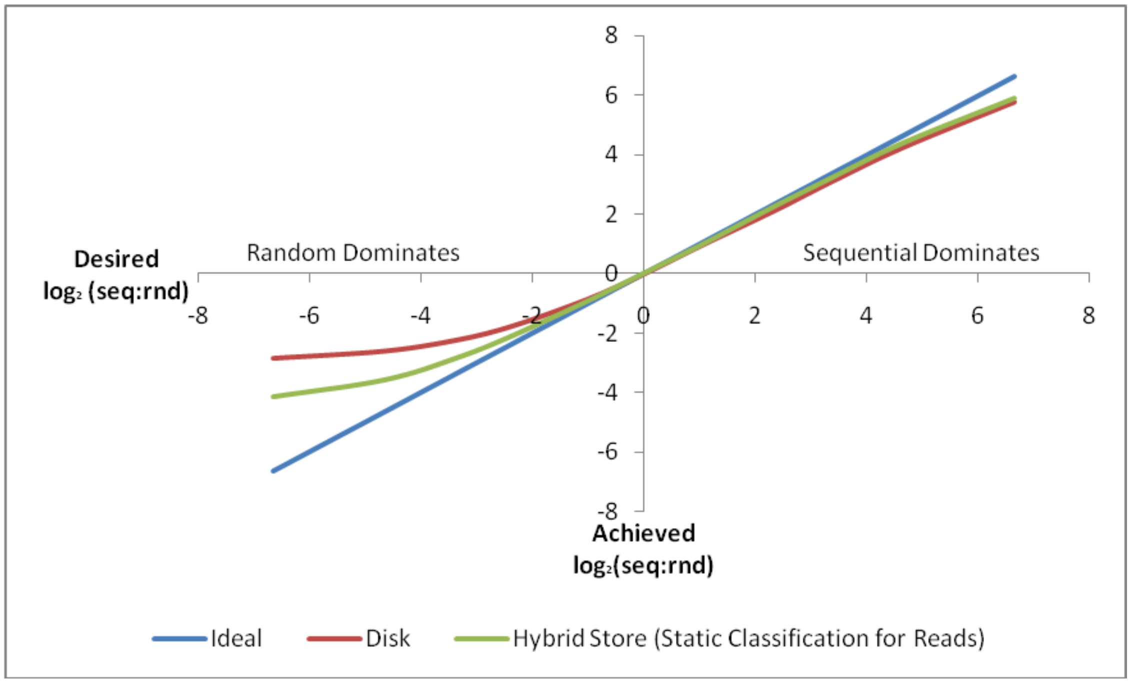


Figure 7: (Log scale) Offline Classification in the Hybrid Store (for read oriented workloads only) under varying weight ratios, using RBFQ

Chapter 4

Flash Model

This chapter considers a basic flash model, and uses it to extend the offline scheme developed in the previous chapter to include write requests. It goes on to show the performance of the extended offline scheme for a real workload (TPC-C benchmark [**14**]).

4.1 Flash Model (with parameters based on average estimates)

Let sequential reads on the disk be β_r times faster than random reads on the disk, and sequential writes on the disk be β_w times faster than random writes on the disk.

Let any writes on the flash be α_w times faster than random writes on the disk and any reads on the flash be α_r times faster than random reads on the disk. Let reads on the flash be D_{rw} times faster than writes on the flash.

Let t_r be the time for a read from the flash and t_w be the time for a write to the flash. Thus $D_{rw} = t_w / t_r$

It has been observed that once there have been a few random writes on the flash, the flash write performance degrades with each future write [**13**]. Assume that this degradation imposes a linear overhead (with slope m). Thus if the number of writes to the flash (following some random writes in the past) is x, the time to write to flash increases to $t_w + m\,x$

The above model parameters have been summarized in Table **4** .

Table **4**: Access times for a request equal to one flash page size

Type of access time	**Disk**	**Flash**
Time for sequential read	$\alpha_r\, t_r$	t_r
Time for random read	$\alpha_r\, \beta_r\, t_r$	t_r
Time for sequential write	$\alpha_w\, t_w$	$t_w + m\,x$
Time for random write	$\alpha_w\, \beta_w\, t_w$	$t_w + m\,x$

4.2 Extension of the Offline Scheme

Using the aforesaid model, the offline scheme developed in the previous chapter can now be extended to include write requests. In general, in the hybrid store, the decision to send a particular write request, either to the disk or to the flash, has the following impact:

1. It affects the performance of future read requests which correspond to that particular write request.

2. It may have a performance impact on future write requests, since the write performance of the flash degrades with each successive write (once there have been a few random writes on the flash in the past).

The restriction on read requests is that they must be sent to the same device where the corresponding write request was sent previously.

Consider a stream that has an average read to write ratio A_{rw}. Using the parameters from the model in the previous section, the average saving in access time obtained by sending a write request from this stream to the flash is expected to be:

Gain = (Speed up due to anticipated future read requests) –

(Slow down due to current write request)

i.e.:

$$\text{Gain} = A_{rw} * (\alpha_r \beta_r - 1) * t_r - (t_w + m\, x - \alpha_w \beta_w t_w)$$

This equation can be interpreted as follows: The difference between the access time for a single read request from the flash and a read request from the disk is $(\alpha_r \beta_r - 1) * t_r$. Since the average read to write ratio is A_{rw}, it is expected that the data written by this particular write request will be read, on the average, by A_{rw} read requests in future. The slow down due to the write request going to the flash is $(t_w + m\, x - \alpha_w \beta_w t_w)$. Note that, in general, this component increases with the number of writes to the flash (i.e. x). Initially this slow-down component is negative and thus it adds to the gain. As the number of writes on the flash increases, the slow-down component increases and correspondingly decreases the gain. *In practice it has been observed that the slow-down component eventually saturates* [**13**] (in future simulations, all measurements have been made at steady state, i.e. after the access time for a write request to the flash saturates).

It can be seen from the above equation that a higher value of A_{rw}, the read-write ratio, results in a greater saving of access time and thus better performance. This implies that, in a hybrid store, read oriented workloads (or read oriented parts of a workload) must be given preference while allocating their writes to the flash. Since the offline scheme (developed in the previous chapter) is based on mapping clusters (of sectors) to the flash or the disk, it can thus be extended to give preference to clusters with random requests and a *higher read to write ratio* while mapping clusters to flash.

4.3 Simulation for the Offline Scheme for reads and writes

4.3.1 Simulation on TPC-C

The above extension to the offline scheme was implemented and a simulation was run on the TPC-C benchmark [**14**]. TPC-C, is an on-line transaction processing (OLTP) benchmark with approximately 84% read requests. The simulation setup is similar to the simulation environment of Table **3** . However the two streams used are TPC-C streams (streams of TPC-C originally meant for disk 0 and disk 1 in a RAID, have been directed to the hybrid store as “tpcc1” and “tpcc2” streams respectively). The simulation considers only the first 10000 requests of “tpcc1” and first 10000 requests of “tpcc2”. Under RBFQ, the minimum service rate of both streams has been assumed to be 30 Requests/sec. The simulation results with a flash size of 256 MB are shown in Table **5**. All measurements have been made at steady state (i.e. after the access time for a write request to the flash saturates. This saturation has been enforced by “warming” the flash with dummy random write requests before the start of the simulation). Note that both tpcc1 and tpcc2 are in effect random workloads. By using flash we are being able to achieve better fairness with higher ratios, than is possible with only a disk.

Table **5**: Offline classification in the Hybrid Store for read and write requests of TPC-C (mapping random and read oriented clusters to the flash)

	Only Disk		**Disk + 256 MB Flash**		
Desired Weight Ratio (tpcc1:tpcc2)	Throughput (Requests/sec)	**Achieved** Fairness (tpcc1:tpcc2)	Throughput (Requests/sec)	% of requests serviced from flash	**Achieved** Fairness tpcc1:tpcc2
1:1	235.26	1.0:1	934.37	**84.96**	1.0:1
5:1	241.68	3.3:1	927.79	**83.73**	4.5:1
10:1	250.68	4.6:1	932.39	**84.11**	8.0:1
1:5	254.52	1:3.6	962.15	**84.41**	1:4.6
1:10	259.70	1:4.8	989.92	**84.89**	1:8.1

The results have been graphically shown in Figure **8** .

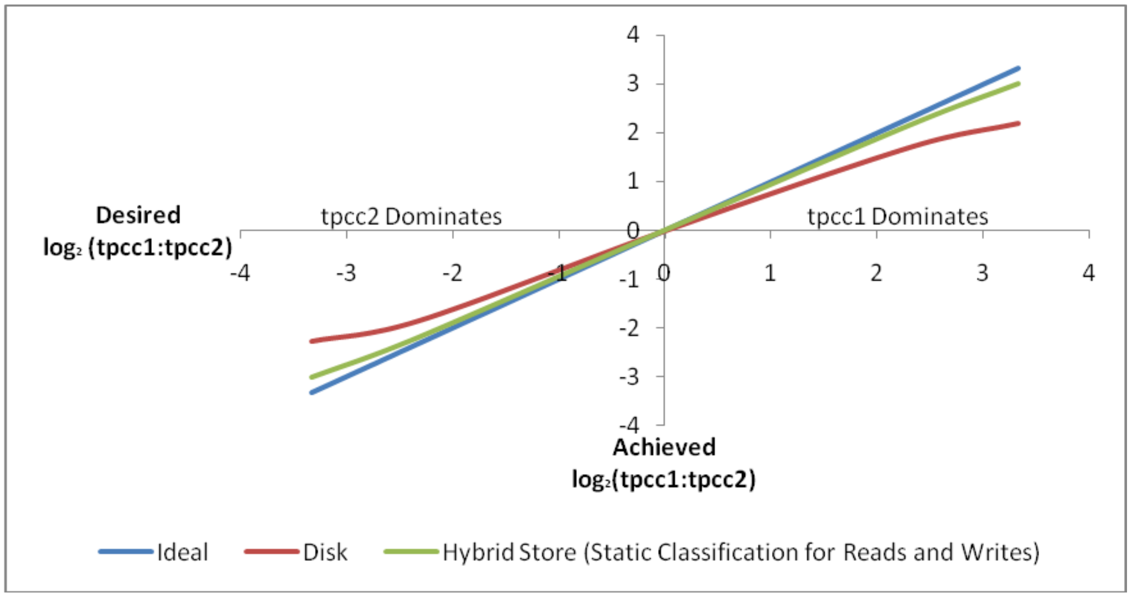


Figure **8**: (Log scale) Offline Classification in the Hybrid Store for reads and writes of TPC-C, under varying weight ratios, using RBFQ

4.3.2 Selecting the Flash Size

Note that a 256 MB flash (used in Table **5**) is able to service 85% of the requests of TPC-C. Experiments were also conducted with different flash sizes, as shown in Table **6** . Increasing the flash size from 128 MB to 256 MB shows significant improvement in throughput for this workload. However, increasing the flash size from 256 MB to 1 GB shows only limited improvement in throughput, and keeping in mind the high cost of a flash device, it may be seen as cost ineffective. Thus flash size can be selected based on the performance-cost tradeoff.

Table **6**: Offline classification for TPC-C: Different flash sizes

Flash Size	Throughput (Requests/sec)	% of requests serviced from flash	**Achieved** Fairness (tpcc1:tpcc2)
0 MB (Only Disk)	235.26	**0**	1:1
128 MB	352.02	**28.61**	1:1
256 MB	934.37	**84.96**	1:1
1 GB	1584.20	**97.58**	1:1

4.3.3 A Note on Read Requests in the Workloads

Since the logical organization of the hybrid store (shown in Figure **6**) is below the level of the file system, it is possible that many read requests may not reach the hybrid store, as they could be absorbed by the buffer cache in the file system. However, the above TPC-C traces are real traces captured below the file system. As can be seen in these traces, read requests can form a significant part of requests below the file system (this could be attributed to the limited locality of the workload).

Chapter 5

Online Scheme

This chapter uses the concepts of the offline scheme developed in the previous chapters, to develop an online scheme that does not preprocess the workloads and makes decisions (i.e. allocation of a write request to the disk or to the flash in the hybrid store) on the fly. It uses the recent history of the workload as a basis for the decisions.

5.1 Overview

The online scheme for the hybrid store makes decisions on the fly, on whether to send a particular write request in the hybrid store to the disk or to the flash. Read requests have the restriction that they must be sent to the device where the corresponding write request had been sent.

The scheme is based on the flash model developed in the previous chapter. The model makes the basic assumption that writes to the flash are slow (as given by the slow down component of the gain equation of the flash model) and majority of the gain in performance is obtained from fast reads on the writes sent to the flash (the read to write ratio dependent component of the gain equation). However, the number of future read requests corresponding to a particular write request are not know in an online scenario, and must be estimated using the history of the workload.

Since the scheme tries to use the flash (in the hybrid store) to reduce the effects of randomness in workloads, it gives random workloads (or random parts of a workload) preference while allocating writes to the flash.

5.2 History of the Workload

The scheme in this chapter maintains statistics based on the history of the workload seen thus far. The logical address space of each workload is divided into clusters of sectors (like in the offline scheme in the previous chapters) and statistics are collected and updated per cluster. Two main statistics need to be maintained:

- Sequentiality of read requests to a cluster: A particular read request is classified as sequential when it is made to a cluster, which is the same as, or adjoining to, the cluster to which the previous request was made. Random clusters are given preference while sending write requests of a cluster to the flash.

- Read to write ratio of requests to a cluster: In general the read to write ratio for a cluster seen thus far, may be used as a rough estimate of the number of times data written by a particular write request to that cluster will be read. The accuracy of this estimate has been analyzed in a later section of this chapter.

5.3 Fitness for Flash

In order for a write request to be sent to the flash, the cluster in which it falls must meet the following fitness criteria:

- It must meet a randomness criterion: for example, according to the history, the number of random read requests to the cluster in the past must have exceeded the number of sequential read requests to the cluster by a threshold (say 1).
- It must have a sufficiently high read to write ratio: for example, according to the history of the clusters, the current cluster must have had a higher read to write ratio than say 50% of the clusters.
- It must not be a "victim" cluster (victim clusters are explained in the next section). However, if the flash is less than say 70% full, this criterion can be relaxed (since as explained in the next section, the only aim of identifying victim clusters is to keep space available on the flash).

If a cluster meets the above criteria, write requests directed to it can be sent to the flash, provided there is space available on the flash.

5.4 Victim Clusters

Victim clusters are clusters which had been declared fit for flash at some point in the past, but which then changed to being unfit for flash, possibly due to a decrease in the observed read to write ratio, or increase in the estimated read sequentiality. Victim clusters are updated on read requests. Whenever a cluster is marked as a victim, write requests to it are sent to the disk (or maybe even data written by write requests to that cluster in the past could be migrated from the flash to the disk). This makes space available on the flash for requests directed to possibly more deserving clusters. For example, among the clusters fit for flash, one can mark say 10% of the clusters with the lowest read to write ratio as victim clusters. Note that since victim clusters are updated on read requests, consequent reads (which have the potential to increase the read to write ratio or decrease the read sequentiality) can cause a victim cluster to be marked as a non-victim and thus write requests to that cluster can be continued to be directed to the flash.

5.5 Simulation for the Online Scheme

5.5.1 Workloads with different percentages of reads to writes

The above online scheme was implemented and simulation results were documented for workloads with different ratios of read requests to write requests. The simulation setup is similar to the simulation environment of Table **2** . Along with synthetic workloads with 0%, 25%, 50%, 75% and 100% read requests, the simulation was also run for TPC-C benchmark which has 84% reads, and the TPC-H benchmark [**17**] which has 99% reads (the simulation for TPC-H had to be executed using a different disk model since the previous disk model was not large enough to accommodate the entire TPC-H trace). The simulation results are shown in Table **7** . Note that the scheme only shows substantial improvements when the percentage of reads is high (for example, above 75%). This is consistent with the assumptions of the flash model that writes to flash are slow and performance improvements are due to fast reads on the data written to the flash.

Table **7**: Online Classification in the Hybrid Store for workloads with different percentages of reads to writes

	Only Disk		**Disk + 256 MB Flash**		
Percentage of read requests	Average Throughput (Requests/sec)	Average Response Time (sec)	Average Throughput (Requests/sec)	Average Response Time (sec)	% of requests serviced from flash
0% (synthetic)	138.42	0.007225	138.42	0.007225	**0**
25% (synthetic)	151.43	0.006604	159.21	0.006281	**20.95**
50% (synthetic)	173.09	0.005777	205.88	0.004857	**45.58**
75% (synthetic)	198.56	0.005036	317.76	0.003147	**68.88**
100% (synthetic)	231.63	0.004317	848.21	0.001179	**90.63**
84% (TPC-C)	235.26	0.004251	790.34	0.001265	**75.56**
99% (TPC-H) [using a different disk model]	467.07	0.002141	1286.68	0.000777	**69.28**

5.5.2 Fairness for TPC-C with the Online Scheme

The simulation results for the fairness of the online scheme, with different desired weight ratios for TPC-C have been shown in Table **8** . The simulation environment is the same as the simulation environment of Table **7** .

Table **8**: Online Classification in Hybrid Store for read and write requests of TPC-C

	Only Disk		**Disk + 256 MB Flash**		
Desired Weight Ratio (tpcc1:tpcc2)	Throughput (Requests/sec)	**Achieved** Fairness (tpcc1:tpcc2)	Throughput (Requests/sec)	% of requests serviced from flash	**Achieved** Fairness (tpcc1:tpcc2)
1:1	235.26	1.0:1	790.34	**75.56**	1.0:1
5:1	241.68	3.3:1	689.57	**71.79**	4.3:1
10:1	250.68	4.6:1	646.08	**70.16**	7.3:1
1:5	254.52	1:3.6	644.34	**69.93**	1:4.5
1:10	259.70	1:4.8	647.13	**69.58**	1:7.3

The results have been graphically shown in Figure **9** .

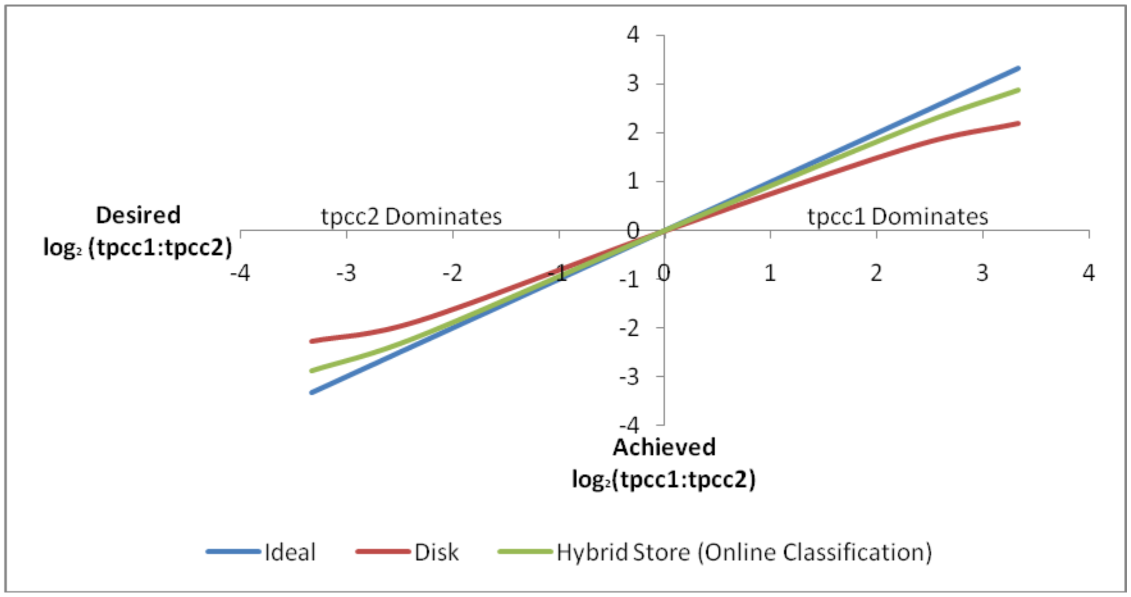


Figure **9**: (Log scale) Online Classification in the Hybrid Store for TPC-C, under varying weight ratios, using RBFQ

5.5.3 Response Time Behavior

In order to see the impact of the hybrid store online scheme on response times, consider the graph shown in Figure **10**. The figure shows a time series of the variation of the average response time of a synthetic workload consisting of 75% read requests (same as in Table **7**). The size of the flash used is 256 MB.

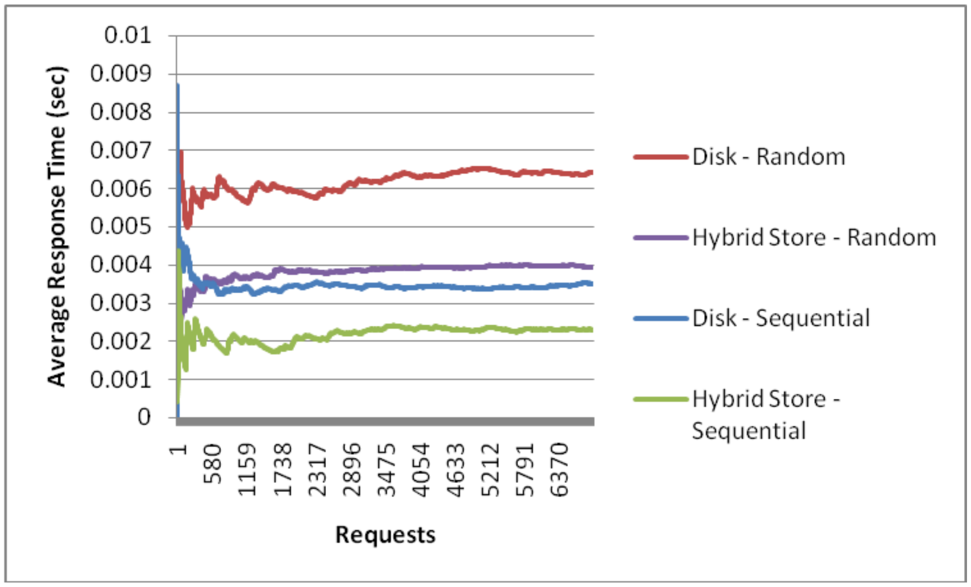


Figure **10**: Time Series of the Average Response Time for a synthetic workload with 75% read requests, using a 256 MB Flash in the Hybrid Store

As can be seen in the above figure, in case of a disk, the average response time of a random stream exceeds the average response time of a sequential stream. In the hybrid store, the average response time of both the random and the sequential streams has been decreased. Now consider the case when the size of the flash is increased to 1 GB. The corresponding graph is shown in Figure **11**.

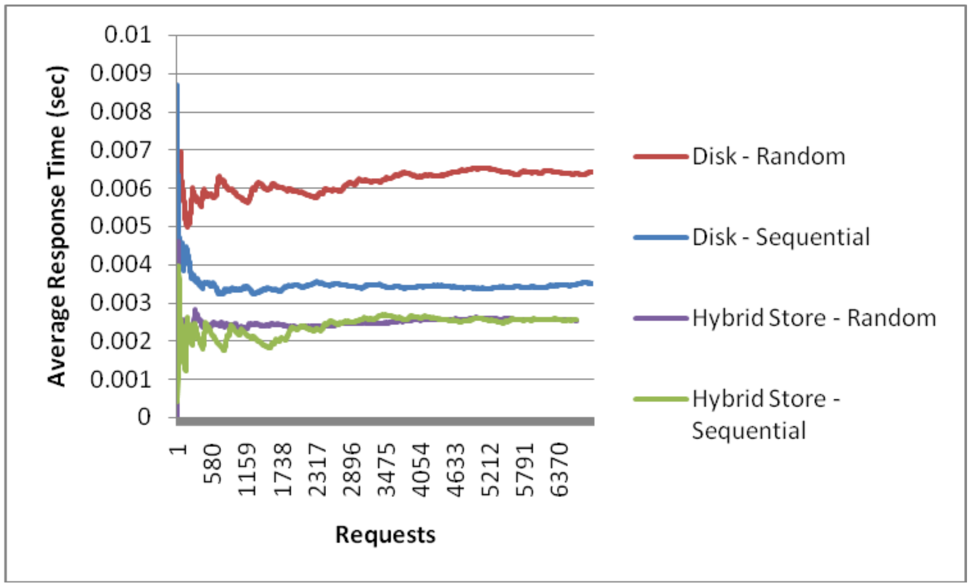


Figure **11:** Time Series of the Average Response Time for a synthetic workload with 75% read requests, using a 1 GB Flash in the Hybrid Store

Note that the only difference between Figure **10** and Figure **11** is that the average response time of the random stream in the hybrid store has been further decreased and is now very close to the average response time of the sequential stream. This is because there is now space available on the flash to service a greater percentage of the requests of the random stream (simulation results show that the increase in the size of flash causes the percentage of total requests getting serviced from the flash to increase from 68.88 % to 75.03%). *This shows that the online scheme for the hybrid store can preferentially decrease the average response time of a random stream*, thus leading to more symmetric average response times between a random and a sequential stream.

5.5.4 Comparison of Online and Offline Schemes

A comparison between the online and offline schemes for the hybrid store is shown in Table **9**. As expected, the offline scheme performs better than the online scheme because it computes exact statistics on the workloads in the first pass. On the other hand, the online scheme must predict parameters such as the read-write ratio and thus its performance is tied to the accuracy of the predictions.

Table **9**: Comparison between the online scheme and the offline scheme

	Disk	**Offline Scheme**		**Online Scheme**		**Comparison**
Workload	Throughput (Requests/sec)	Throughput (Requests/sec)	% Increase	Throughput (Requests/sec)	% Increase	**Offline:Online Ratio**
Synthetic (75% reads)	198.56	638.92	221.78	317.76	60.03	**3.69**
Synthetic (100% reads)	231.63	848.21	266.19	848.21	266.19	**1.00**
TPC-C	235.26	1152.51	389.89	790.34	235.94	**1.65**
TPC-H [using a different disk model]	467.07	1275.86	173.16	1286.68	175.48	**0.99**

Note that the performance improvement of the two schemes is workload dependent. The performance of the online scheme, as compared to the offline scheme, depends upon the predictability of workload characteristics. An analysis of the predictability of the read-write ratio, an important workload characteristic which affects the relative performance of the two schemes is provided here. The synthetic workload with 100% reads shows similar performance under the online and offline schemes since the read-write ratio is known and constant. Figure **12**, Figure **13** and Figure **14** and show the time series of the read write ratio for the synthetic (75% reads), TPC-C and TPC-H workloads respectively. In the figures, the read-write ratio of the overall workload, as well as that of five clusters is shown (the five clusters are chosen on basis of the range of their final read-write ratios). TPC-C and TPC-H show better predictability than the synthetic workload and thus for these workloads, the difference in performance between the online and offline schemes is lesser than in case of the synthetic workload. TPC-H exhibits much better predictability than TPC-C at the cluster level.

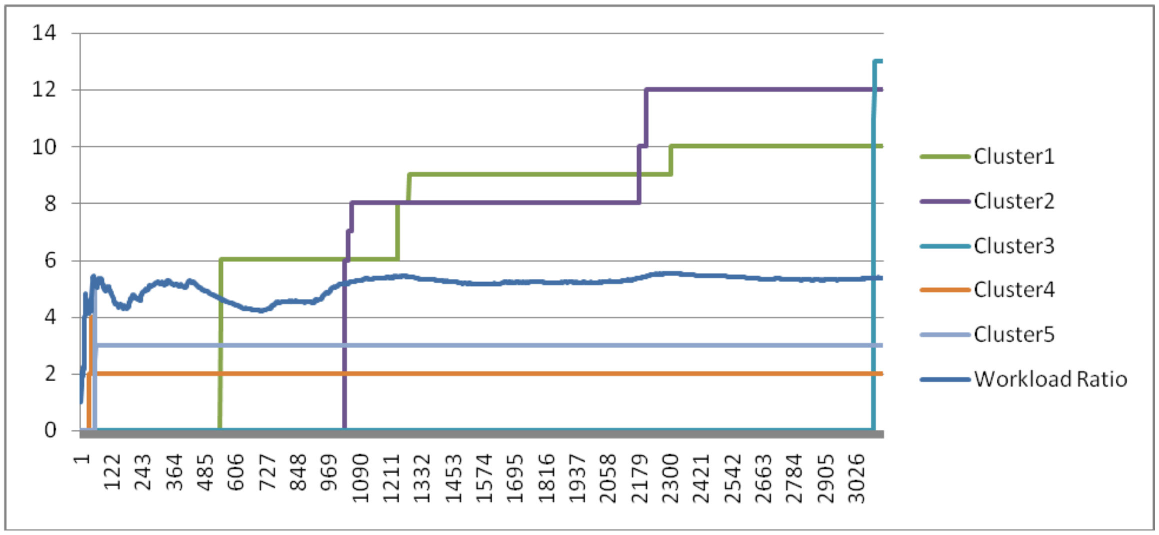


Figure **12**: Time Series of Read-write Ratio of TPC-C

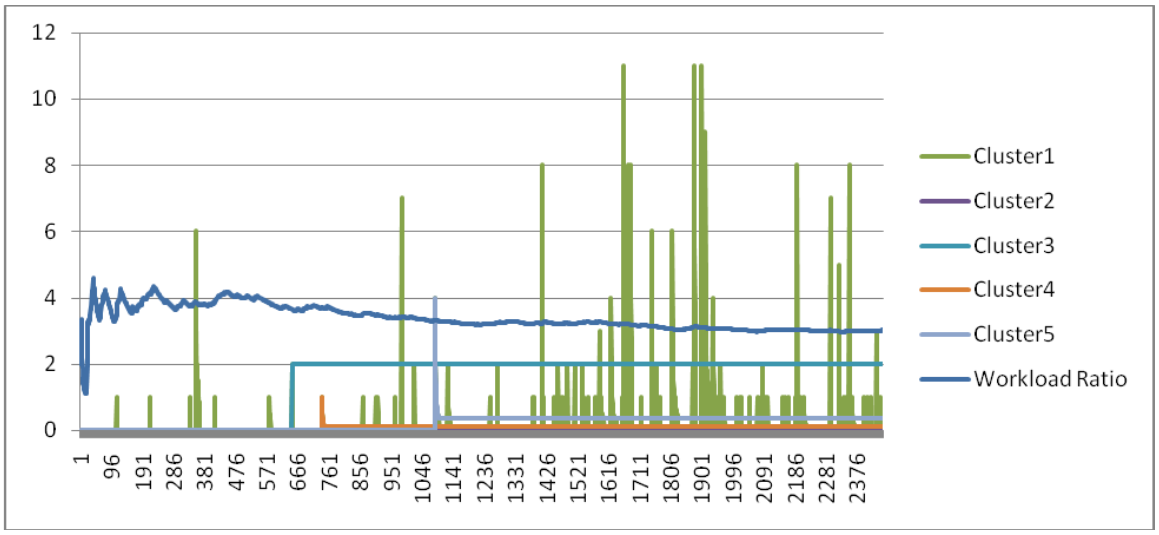
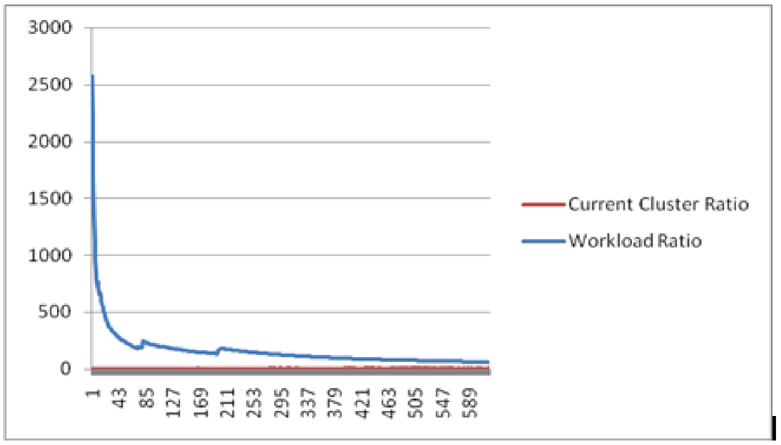


Figure **13**: Time Series of Read-write Ratio of synthetic workload with 75% read requests



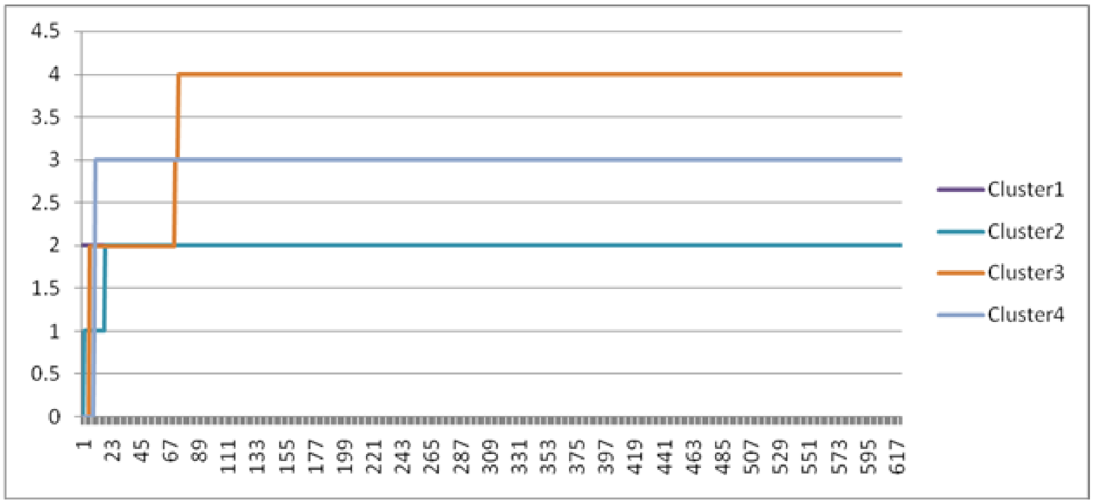


Figure **14**: Time Series of Read-write Ratio of TPC-H

5.5.5 Sequentiality of Workloads

Now consider sequentiality, another important workload characteristic. Note that since the two schemes define a random cluster as one whose sequentiality is below a particular threshold, an exact prediction of this parameter is not critical to the relative performance of the online scheme as compared to the offline scheme. However, the sequentiality of the workloads impacts the improvements obtained in the the hybrid store and the percentage of requests which are serviced from the flash. Figure **15** shows the sequentiality of the clusters to which instantaneous read requests are directed (i.e the current cluster at that point in the simulation), for the synthetic workload mix consisting of 75% reads, at a 1:1 ratio between the sequential and the random workloads. The workload mix is, in effect, random. Increasing the desired ratio to 100:1 results in a workload mix in which the sequential workload dominantes (as shown in Figure **16**), and thus this workload mix achieves a higher throughput at a disk. As was illustrated in Table **3**, the flash in the hybrid store services a lower percentage of requests of for such a workload mix.

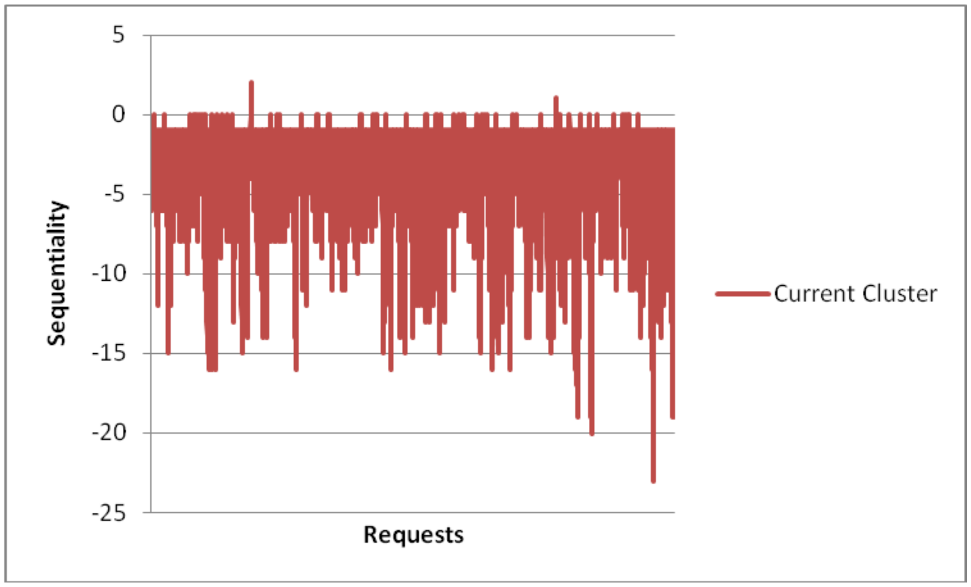


Figure **15**: Sequentiality of the clusters to which the instantaneous read requests are directed for the synthetic (75% reads) workload mix, at a 1:1 sequential to random ratio

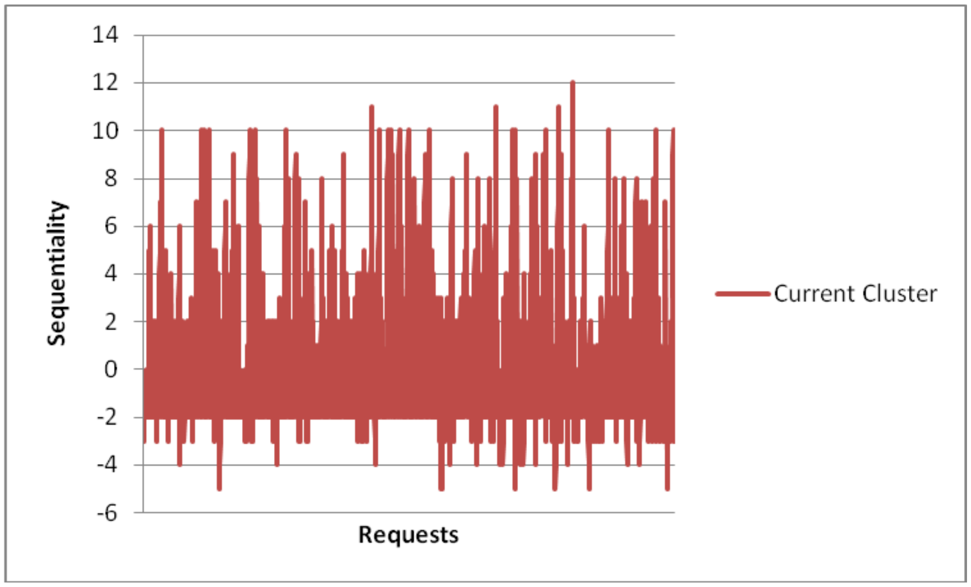


Figure **16**: Sequentiality of the clusters to which the instantaneous read requests are directed for the synthetic (75% reads) workload mix, at a 100:1 sequential to random ratio

Figure **17** and Figure **18** show the sequentiality of the clusters of TPC-C and TPC-H workloads respectively. TPC-C is predominantly random, while TPC-H exhibits more sequentiality than TPC-C. This explains why the percentage of requests serviced from the flash for TPC-H in Table **7** is lower than TPC-C, even though TPC-H has a much higher read to write ratio.

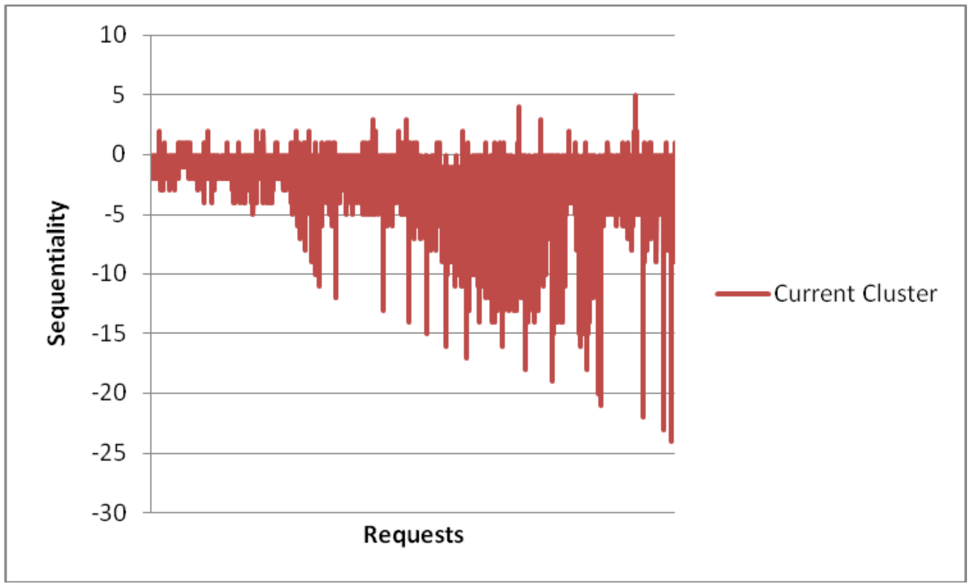


Figure **17**: Sequentiality of TPC-C

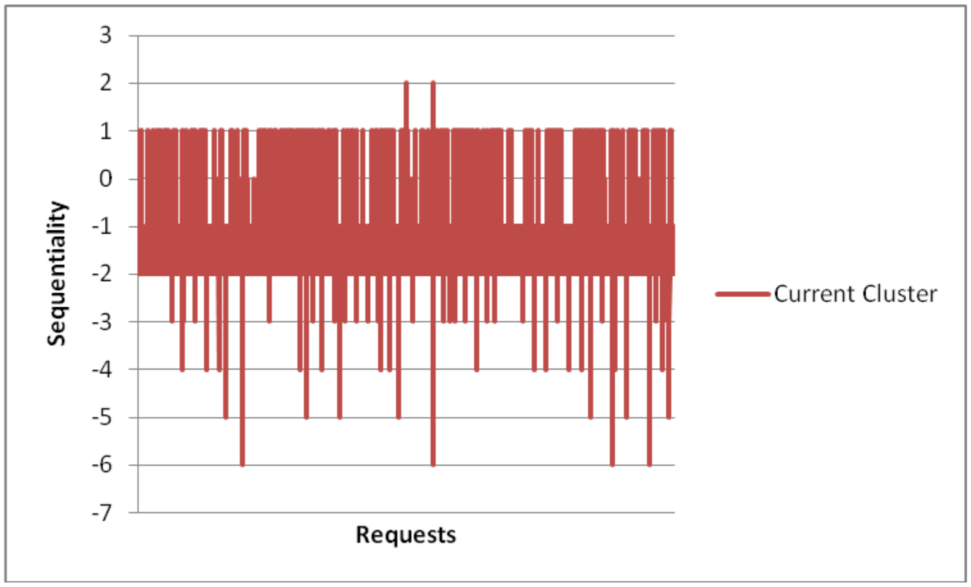


Figure **18**: Sequentiality of TPC-H

Chapter 6

Conclusions

In a virtualized storage system, an important quality of service goal is to try to optimize fairness, while also ensuring that the overall performance stays above a minimum acceptable threshold. Storage performance is notoriously difficult to estimate in a storage system and varies significantly with workload characteristics. In general, due to the mechanical nature of a disk, a random workload achieves a lower throughput, and experiences a higher response time than a sequential workload. As a result of this behavior, under an algorithm that aims to achieve the aforesaid QoS goal, achieved weight ratios can deviate from the desired weight ratios, causing asymmetric fairness between a random and a sequential workload.

Flash exhibits symmetric performance for a random and a sequential workload and thus has the potential of making storage systems independent of the effects of randomness in workloads. However, since flash is expensive and has a limited lifetime, the size of a flash device in a real system is expected to be small enough to service only certain parts of the workloads. Thus, flash must be used in combination with a disk, in an organization termed by its proposers as a “hybrid store”.

While organizing data in a hybrid store, the challenge is to decide the device to which a particular write request must be directed in order to optimize both, the overall performance, and the fairness. Two schemes, which organize data in a hybrid store, have been presented: an offline scheme (which makes two passes over the input, the first to study the workloads and compute certain statistics, before the actual simulation in the second pass) and an online scheme (which makes a single pass over the input, computing statistics and making decisions on the fly). The schemes partition the logical address space into clusters of sectors, and maintain certain statistics for each cluster. The flash model used by the two schemes assumes that writes to the flash are slow, and improvements result only from future fast reads on the writes.

The simulation results of implementations of the two schemes show that flash in a hybrid store can help to achieve more symmetric QoS between a random and a sequential workload (i.e. it has the potential to preferentially increase the throughput and decrease the response time of a random workload). It can also help to improve overall QoS by increasing both, the overall performance, and the fairness. However, the results are workload dependent. An analysis of some of the important workload characteristics which affect the performance of the two hybrid store schemes has been provided in the thesis.

Bibliography

1. P. Shenoy and H. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. *In Proceedings of the ACM SIGMETRICS'98, June 1998*.

2. J. Zhang, A. Riska, A. Sivasubramaniam, Q. Wang and E. Riedel. Storage Performance Virtualization via Throughput and Latency Control. *In Proceedings of the 13th IEEE MASCOTS'05, ACM Transactions on Storage, August 2006*.

3. A. Gulati, A. Merchant, and P. Varman. pClock: An Arrival Curve Based Approach for QoS Guarantees in Shared Storage Systems. *In Proceedings of the ACM SIGMETRICS'07, June 2007.*

4. W. Jin, J. Chase and J. Kaur. Interposed Proportional Sharing for a Storage Service Utility. *In Proceedings of the SIGMETRICS/Performance'04, June 2004*.

5. A. Gulati, A. Merchant, M. Uysal, and P. Varman. Efficient and Adaptive Proportional Share I/O Scheduling. *In HP Labs External Technical Report, November 2007.*

6. A. Gulati, A. Merchant, and P. Varman. Proportionate Resource Allocation With Reservations Under Variable Capacity. *Earlier version appeared as a short paper in PODC, 2007.*

7. Y. Wang and A. Merchant. Proportional-Share Scheduling for Distributed Storage Systems. *In Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST'07), February 2007.*

8. C.. Waldspurger. Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management. *In a doctoral dissertation submitted to the Massachusetts Institute of Technology, September 1995.*

9. B. Verghese, A. Gupta, and M. Rosenblum. Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors. *In Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, October 1998.*

10. J. Wu, S. Banachowski and S. Brandt. Hierarchical Disk Sharing for Multimedia Systems. *In Proceedings of the NOSSDAV'05, June 2005*

11. The DiskSim Simulation Environment (version 3.0). *http://www.pdl.cmu.edu/DiskSim/*

12. Y. Kim. FlashSim Simulation Environment. *Courtesy, Computer Systems Laboratory, Pennsylvania State University*.

13. Y. Kim. Hybrid Store (a PhD thesis proposal). *Courtesy, Computer Systems Laboratory, Pennsylvania State University*.

14. The TPC-C on-line transaction processing benchmark. *http://www.tpc.org/tpcc/*

15. E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *In ACM Computing Surveys (CSUR), v.37 n.2, p.138-163, June 2005.*

16. T. Bisson and S. Brandt. Reducing Hybrid Disk Write Latency with Flash-Backed I/O Requests. *In Proceedings of the IEEE MASCOTS'07, October 2007.*

17. The TPC-H decision support benchmark. *http://www.tpc.org/tpch/*