ABSTRACT

GODBOLE, CHAITANYA UMESH. Performance Evaluation of iSCSI Protocol for Mirroring Application. (Under the direction of Dr. Michael Devetsikiotis).

FC SAN has been the work-horse of the storage industry for quiet sometime now. It has prevailed and prospered in the enterprise-level storage environment due to its high performance and reliability. Data mirroring for disaster management and data recovery is gaining immense importance as the amount of data being stored increases exponentially. FC has been the default transport protocol for mirroring due to its performance advantages. But now with the demand for mirroring solutions for small & medium sized businesses being on the rise, the high acquisition and maintenance costs of FC have propelled iSCSI to be a cost-effective and viable alternative to FC. Traditionally iSCSI has been deemed unfit for delay sensitive applications due to its slow nature and lower throughput.

In this thesis, we attempt to evaluate the performance of mirroring over iSCSI and show that it can be performed satisfactorily and economically, without requiring the costlier FC option. This study involves the use of a customized caching algorithm deployed on a SAN in order to exploit any performance enhancements when it comes to the response time for an application running on iSCSI. The cache tries to reduce the response-time for a write request for mirroring by deploying a two-level primary cache with a faster, smaller backup cache or a faster primary cache with a two-level backup cache. A comparative study of the results obtained for mirroring after deploying these caches over FC and iSCSI show that iSCSI provides adequate performance and reliability for a successful deployment of the mirroring application over it.
Performance Evaluation of iSCSI Protocol for Mirroring Application

by
Chaitanya Umesh Godbole

A thesis submitted to the Graduate Faculty of
North Carolina State University
In partial fulfillment of the
Requirements for the degree of
Master of Science

Computer Engineering

Raleigh, North Carolina

2008

APPROVED BY:

_________________________________  ___________________________________
Dr. Michael Devetsikiotis             Dr. Wenye Wang
Committee Chair

______________________________
Dr. Khaled Harfoush
DEDICATION

Dedicated to my parents, who mean everything to me in this world. They have not only done what is usually expected from a parent for their child, but they have in fact surpassed any and every expectation for an ideal parent. It is their constant caring, support and encouragement that enabled me to come to NCSU for my Masters and complete it successfully.

Dedicated to Him, The Almighty, for giving me such great parents and everything else in my life thus far.

Dedicated to all my teachers who have always guided me towards the correct path in achieving my goals.

Dedicated to my fiancée who has been an immense support throughout.

Dedicated to all my family friends and relatives for their well-wishes and blessings.
BIOGRAPHY

Chaitanya Umesh Godbole was born in Dombivli, Mumbai, India to Umesh and Medha Godbole. He completed his undergraduate program in Electronics from Mumbai University, India and entered North Carolina State University for his Masters in Computer Engineering in Fall 2006. While working on his Masters degree, he worked on his thesis under the guidance of Dr. Michael Devetsikiotis.
ACKNOWLEDGMENTS

I would like to thank everyone who has contributed to the success of this thesis and for every encouragement given by them. I would like to profoundly thank my advisor Dr. Michael Devetsikiotis for giving me an opportunity to conduct thesis research under him. I also thank him for his invaluable support and insightful advice throughout the duration of my thesis. I also thank Dr. Srikant Nalatwad, former Doctoral student of my advisor, for his technical expertise in the SAN environment and his support towards conducting my research. I also thank Dr. Wenye Wang and Dr. Khaled Harfoush for agreeing to be on my thesis committee. I also thank the ECE Department and NCSU for providing me an opportunity to pursue my Masters at this esteemed institute of learning.
# TABLE OF CONTENTS

List of Figures .................................................................................................................................................... vi
List of Tables .......................................................................................................................................................... viii
Chapter 1 ............................................................................................................................................................... 1
Introduction ........................................................................................................................................................... 1
Chapter 2 ............................................................................................................................................................... 13
Background ........................................................................................................................................................... 13
  2.1 The Small Computer System Interface Protocol .................................................................................... 13
  2.2 The Fibre Channel Protocol ........................................................................................................................ 22
  2.3 The Storage Area Networks ........................................................................................................................ 33
    2.3.1 Fibre Channel Storage Area Networks ............................................................................................. 33
    2.3.2 Internet Protocol Storage Area Networks .......................................................................................... 38
    2.3.2.1 Internet Fibre Channel Protocol ................................................................................................. 41
    2.3.2.2 Fibre Channel over Internet Protocol ........................................................................................... 45
  2.4 Internet Small Computer System Interface ............................................................................................... 48
    2.4.1 Protocol Overview ............................................................................................................................... 48
    2.4.2 Login and Session Establishment ........................................................................................................ 51
    2.4.3 Connection State Diagrams ................................................................................................................ 58
Chapter 3 ............................................................................................................................................................... 64
Related Work ......................................................................................................................................................... 64
Chapter 4 ............................................................................................................................................................... 74
Algorithm Implementation .................................................................................................................................... 74
  4.1 DCD caching algorithm ............................................................................................................................... 74
  4.2 RAPID caching algorithm ............................................................................................................................ 78
  4.3 Modified-DCD cache and Modified-RAPID cache algorithms ................................................................. 81
Chapter 5 ............................................................................................................................................................... 90
Experiments and Results .................................................................................................................................... 90
  5.1 Experimental Setup ..................................................................................................................................... 90
  5.2 Data Collection and Analysis ....................................................................................................................... 94
    5.2.1 mDCD Experiments for iSCSI .......................................................................................................... 95
    5.2.2 mRAPID Experiments for iSCSI ....................................................................................................... 110
    5.2.3 mDCD v/s mRAPID comparisons for iSCSI .................................................................................. 124
    5.2.4 FC v/s iSCSI for mDCD and mRAPID ............................................................................................. 129
  5.3 Results and Inferences ............................................................................................................................... 133
    5.3.1 For mDCD ........................................................................................................................................ 133
    5.3.2 For mRAPID .................................................................................................................................... 135
    5.3.3 For mDCD v/s mRAPID over iSCSI ................................................................................................. 136
    5.3.4 For FC v/s iSCSI comparisons of the two algorithms ....................................................................... 136
Chapter 6 ............................................................................................................................................................... 141
Conclusions and Future Work ........................................................................................................................... 141
List of References ................................................................................................................................................. 144
LIST OF FIGURES

Figure 1: Data flow for synchronous remote mirroring .............................................................. 7
Figure 2: Data flow for asynchronous remote mirroring ............................................................ 7
Figure 3: The SCSI client-server architecture ........................................................................... 14
Figure 4: SCSI structural/domain model .................................................................................. 15
Figure 5: SCSI protocol distributed communication service reference model ......................... 17
Figure 6: Request-response service model .............................................................................. 18
Figure 7: SCSI CDB format ...................................................................................................... 19
Figure 8: Parallels between OSI reference model & SAM-2 ..................................................... 21
Figure 9: Fibre Channel protocol stack .................................................................................... 25
Figure 10: Transition from SCSI to FC SAN ........................................................................... 26
Figure 11: Parallels between OSI reference model & FC model ............................................. 27
Figure 12: FCP login process .................................................................................................... 29
Figure 13: FCP’s frame structure ............................................................................................. 32
Figure 14: Types of arbitrated loops ....................................................................................... 35
Figure 15: Fibre topology FC SAN ........................................................................................... 36
Figure 16: Frame formats comparison for FCP, iFCP, FCIP and iSCSI ................................. 40
Figure 17: Protocol stacks comparison for FCP, iFCP, FCIP and iSCSI .................................. 40
Figure 18: iFCP network architecture .................................................................................... 42
Figure 19: Fabric login process of iFCP ............................................................................... 43
Figure 20: SNS request processing for iFCP .......................................................................... 44
Figure 21: Standard FCIP connection ..................................................................................... 46
Figure 22: iSCSI initiator to target command flow ................................................................. 49
Figure 23: iSCSI layering structure ......................................................................................... 50
Figure 24: iSCSI PDU format .................................................................................................. 51
Figure 25: iSCSI login request PDU ......................................................................................... 54
Figure 26: iSCSI login response PDU ..................................................................................... 55
Figure 27: Initiator connection state diagram ......................................................................... 61
Figure 28: Target connection state diagram ......................................................................... 63
Figure 29: mDCD cache architecture ..................................................................................... 82
Figure 30: mRAPID cache architecture .................................................................................. 83
Figure 31: Experimental setup ............................................................................................... 90
Figure 32: Average response time v/s cache sizes for various watermark levels mDCD. ........ 97
Figure 33: IOps v/s cache sizes for various watermark levels mDCD ...................................... 98
Figure 34: MBps v/s cache sizes for various watermark levels mDCD .................................. 98
Figure 35: %CPU Utilization v/s cache sizes for various watermark levels mDCD .............. 99
Figure 36: Average response time v/s page sizes mDCD ....................................................... 100
Figure 37: IOps v/s page sizes mDCD .................................................................................... 101
Figure 38: MBps v/s page sizes mDCD .................................................................................. 101
LIST OF TABLES

Table 1: FC auxiliary services. ........................................................................................................... 31
Table 2: Data disk specifications. ........................................................................................................... 92
Table 3: Varying cache sizes for 80-60 watermark (mDCD). ............................................................. 96
Table 4: Varying cache sizes for 80-40 watermark (mDCD). ............................................................. 96
Table 5: Varying cache sizes for 60-40 watermark (mDCD). ............................................................. 97
Table 6: Page size variation for mDCD ideal configuration ................................................................. 100
Table 7: Varying cache sizes for no-prefetch (mDCD). ................................................................. 102
Table 8: Prefetch and segment multiplier variation for mDCD ideal configuration. ....................... 105
Table 9: Prefetch and segment size variation for mDCD ideal configuration .................................... 107
Table 10: Varying cache sizes for 80-60 watermark (mRAPID). .................................................... 110
Table 11: Varying cache sizes for 80-40 watermark (mRAPID). .................................................... 111
Table 12: Varying cache sizes for 60-40 watermark (mRAPID). .................................................... 111
Table 13: Page size variation for mRAPID ideal configuration ....................................................... 114
Table 14: Varying cache sizes for no-prefetch (mRAPID). ............................................................. 117
Table 15: Prefetch and segment multiplier variation for mRAPID ideal configuration. ................... 119
Table 16: Prefetch and segment size variation for mRAPID ideal configuration. ............................. 122
Table 17: Average response time comparisons with varying cache sizes for mDCD and mRAPID ideal configurations ......................................................................................... 125
Table 18: Average response time comparisons with varying page sizes for mDCD and mRAPID ideal configurations ................................................................................................. 126
Table 19: Average response time comparisons with varying cache sizes for mDCD and mRAPID with no-prefetch. ................................................................................................. 127
Table 20: Average response time comparisons with varying prefetch and segment multiplier values for mDCD and mRAPID ideal configurations ......................................................... 128
Table 21: Average response time comparisons with varying prefetch and segment size values for mDCD and mRAPID ideal configurations. ........................................................... 129
Table 22: FC v/s iSCSI no cache. ........................................................................................................... 130
Table 23: FC v/s iSCSI mDCD ideal configuration ............................................................................. 130
Table 24: FC v/s iSCSI mRAPID ideal configuration ........................................................................ 130
Table 25: FC v/s iSCSI no-prefetching mDCD. .................................................................................... 131
Table 26: FC v/s iSCSI no-prefetching mRAPID. .............................................................................. 131
Table 27: FC v/s iSCSI variable prefetch mDCD. .............................................................................. 132
Table 28: FC v/s iSCSI variable prefetch mRAPID. ........................................................................... 132
Table 29: FC v/s iSCSI constant prefetch mDCD. ............................................................................. 132
Table 30: FC v/s iSCSI constant prefetch mRAPID. ........................................................................... 132
Chapter 1

Introduction

The Storage Area Network (SAN) market has been traditionally dominated by the Fibre Channel (FC) enabled networked storage. The FC SANs have entrenched themselves deep into the enterprise-level market and have been the storage-of-choice for most of the high-end users and mission-critical applications. The reason for this tremendous success of FC SANs over any other prevalent storage technology like Direct Attached Storage (DAS) or IP SANs is due to the fact that FC provides excellent overall performance, higher data availability, reliability, scalability and ease of storage management as a transport protocol for a storage network and can virtually suffice the performance need of any application possibly used on a SAN. FC is a fairly mature technology and has been supported by every major player in the SAN industry. It’s been constantly improved and that has enabled it to provide very high bandwidth, high throughput and high availability for excellent business continuity. It can support a host
of applications like data backup, mirroring, servicing large databases, medical imaging, video on demand, etc. and many other business critical applications. But with development of other storage technologies and due to higher costs involved in owning and maintaining a FC SAN, which is unaffordable for the middle and lower end of the storage market; FC SAN is facing significant competition from IP SAN implementations.

There is an appreciable rise in the demand for storage from within the small and medium sized market (SMB) in the past few years. The projection for this demand is to the tune of thousands of Terabytes in next few years. The fact that even a common household can generate a few Gigabytes of storable information cannot be undermined in this age of Internet proliferation and information overdrive. How are we going to provide an economically feasible solution to this SMB market? So should we aim for an all together new approach or should we investigate the existing ones and determine whether they deserve a new pair of eyes looking at them? So the question in fact is should we even bother about iSCSI for that matter? Let’s not consider this research to be a show-down between the “high-end market, high maintenance heavy-weight” Fiber Channel (FC) versus the “low-cost, compromised performance” challenger iSCSI. It’s not about projecting iSCSI as reliable or viable alternative for mission critical applications. We do not expect the big enterprises with existing FC backbone and involved in “data sensitive” sectors like the banking and financial institutions, Defense
Department, etc. to replace all their FC investments with iSCSI solutions, hoping that it would work just as well as the previous one, but at a much lesser cost. As aforementioned, we are reviewing iSCSI as a SMB market solution to the “heavy on the wallet” FC option. We also are looking at iSCSI as the protocol of choice for the high-end enterprises when they are weighing their options for expanding their existing storage capabilities. We think that the areas like web servers/web sites, email, archiving of data, mirroring of data, backups, remote offices and many other low usage applications would be mostly benefited by using iSCSI.

iSCSI is Internet Small Computer System Interface, developed by Internet Engineering Task Force to enable transport of SCSI commands over the TCP/IP network. It enables a block-level initiator-target communication over the TCP/IP network. The iSCSI protocol has been around for nearly a decade now and was initially seen as something which could overhaul the costly to acquire and difficult to maintain FC storage. Though iSCSI has the potential, the market for FC that already is existent, has the inertial property which is difficult to overcome so easily. Instead of trying to displace the FC from its market, we should think of this in a different way. Let's target the untapped SMB market. A company lacking a proper SAN infrastructure or the expertise required for FC will find it compelling today to go for iSCSI due the higher acquisition costs, complexity and management costs involved with FC. Also the customers with already existing FC deployment in their SAN are also looking towards
iSCSI for increasing the reach of the network; put more servers on to the SAN. Today we find many of the big players in the storage solutions industry are starting to provide their own iSCSI product line. Microsoft came out in support of iSCSI with their iSCSI Architecture and iSCSI software initiator. Microsoft has extended its “Designed for Windows Logo Program” to include iSCSI devices to enable independent hardware vendors like Equallogic Inc., Network Appliance Inc., Qlogic Corp., Cisco Systems Inc., Intransa Inc., etc. to qualify their iSCSI hardware components for use with Windows.

iSCSI’s biggest selling points are its overall low acquisition costs, low administrative costs, easily available skilled workforce, global reach by using Ethernet and IP technologies, better storage utilization and manageability. With the need for consolidated storage on the rise, iSCSI provides an excellent affordable solution for the small and medium-sized businesses. According to the latest market predictions, the market share of iSCSI SANs which was mere 4% in 2006 ($369 million), would jump up to 20% by 2010 (total revenues of around $2 billion, with an annual growth rate of over 70%) and thus would truly start competing with the FC SANs as a serious storage alternative even in the high-end region. Experts feel that generally for over a period of 5 years, the total cost of ownership (TCO) for an IP SAN out-performs the FC SAN (IP SAN: $75,000; FC SAN: $750,000). iSCSI can support applications such as databases that require block level access, to use networked storage systems with comparable system performance to that of the FC. It can also be used for backups, replications and
mirroring purposes. Some of the typical applications for iSCSI products could be file system mirroring, graphical image storage, providing disk space on demand, CAD/CAM, onsite repository for backup data, replacing traditional backup methods, snap shot critical data, etc. Some of the performance experiments conducted by the storage community for comparing iSCSI and FC, have shown that iSCSI offers nearly 80% the performance of the 2Gb Fibre Channel, which is acceptable for most of the applications. The 4Gb FC has a little better performance, but has very high acquisition costs too. Also with the advent of 10Gb/sec Ethernet and dynamic TCP offload capabilities being added to those adapters, this performance gap could be narrowed down even further. The future of iSCSI lies in not being confrontational with the FC SANs, instead being complimentary to them in case of existing infrastructure and by reaching out to those markets that lack the FC infrastructure. With the pace of data growth being on the increase in the recent times, iSCSI has plenty of room to grow without the need of FC being displaced by iSCSI. The reduced cost and availability of 10GbE as and when it comes in next 2-3 years will act as an enabler for iSCSI storage to become more prevalent and enhance its overall performance.

Remote mirroring provides excellent solution for disaster-recovery, high data availability and business continuity in face of catastrophes which would otherwise severely compromise the primary data and result in tremendous data and revenue losses. Remote mirroring enables creating exact copies of the primary data at a site far away
from the primary site, but connected to it via some kind of an underlying network. The mirroring process is invisible to the application servers and so does not affect the allocated resources for the application in any way, but as the process is dependent on the underlying network and the participating disk-subsystems, the resulting reduction in performance could get reflected at the application level too. There are two techniques of remote mirroring; synchronous mirroring waits for the secondary storage array to acknowledge the receipt of the mirrored data before itself sending an acknowledgement to the host for its current request. Asynchronous mirroring sends an acknowledgement to the host for its current request as soon as it has serviced that request, but carries out the actual mirroring process at a later point in time with the secondary array. The advantage of synchronous mirroring is that it ensures an always up-to-date secondary data copy, but has a disadvantage of higher response times for the application server requesting service from the primary storage site. The advantage of asynchronous remote mirroring is a much better response time for the application host as compared to synchronous mirroring, but the disadvantage is the fact that the secondary data copy could very well be inaccurate, as the data is always mirrored at a later period, and so if the primary site goes down before being able to carry out this process, then there would be data inconsistencies for the host as the secondary doesn’t have the required data from the primary as yet [23]. Figure 1 shows the data flow for synchronous remote mirroring and figure 2 shows the data flow for asynchronous remote mirroring. Here both the host-
primary array and primary-secondary array connections could be over a large distance and can traverse multiple networks.

![Figure 1: Data flow for synchronous remote mirroring [23].](image1)

![Figure 2: Data flow for asynchronous remote mirroring [23].](image2)

Commercially available synchronous remote mirroring software has been used as a part of the experimental setup for this thesis research. The software duplicates production site data (primary) to one or two secondary storage sites through online synchronous data mirroring and data protection. It is host, network, application and
operating system independent. The host sends a read/write request to the primary array. If the request is a read request, the data is fetched into the read cache of the primary and subsequently transferred to the host and no mirroring action takes place. But if it’s a write request to the primary, then it is first stored into the write cache of the primary (the storing technique and the actual cache depends on the caching algorithm being used and they will be explained in detail in the due course of this document) and immediately the mirroring process begins which will send a copy of this write data to the secondary array. There it too will be stored into the write cache. As soon as the mirrored data is stored safely in the secondary’s cache, the secondary acknowledges the receipt of the mirrored data to the primary, which in-turn promptly acknowledges the receipt of the current write request to the host. The secondary in the mean time will go through all the exact same steps as the primary went through to store the data on to its backend disks.

Traditionally Fibre Channel has been the protocol of choice for carrying out any kind of remote mirroring due to its previously mentioned advantages, but we are using the iSCSI as the transport protocol for the remote mirroring sessions during the scope of this thesis’s experiments and are attempting to show that even iSCSI can perform satisfactorily during the mirroring sessions and can be considered as a viable option to the FC implementation. For a non-clustered Windows host system, the default disk timeout registry value is 10 seconds and for a clustered Windows host, its 20 seconds. This value is the amount of time Windows will wait for a hard disk to respond to a
command (i.e. in case of mirroring, it’s the time for which the Windows host will wait for an acknowledgement of its read/write request from the primary array). For a standard Online Transaction Processing (OLTP) workload, the maximum timeout value after which if no acknowledgement is obtained for a transaction is 2 seconds. In case of our worst-case result (with cache enabled) for a particular IOMeter (benchmark tool) I/O load, the maximum average response time is \( \sim 101\text{msec} \), which is much lesser than both the above timeouts for any standard workload transactions for a mirroring application. If a proper failover mechanism is in place, then the secondary can even take up the roles of a primary in case of the primary being rendered functionally in-active. This is possible as the secondary will always have a current and up-to-date copy of all the data at its end too and so the only change required here is to sever the host connections from the non-functional primary and connect it to the secondary. The primary can then be fixed and brought-up again, and a data-sync can be carried out with the secondary to bring the primary to a current-state and resume its services to the hosts. All along this time, the services to the hosts suffer a minimum degradation or unavailability and also there is no data loss. As the data has to travel over a physical network during synchronous mirroring, a considerable amount of delay is experienced by the host to get an acknowledgement for the request and if its excessive, then the mirroring application fails. Hence ways to reduce this delay are being sought after for a better mirroring experience. One way would be to speed-up the underlying transport network and another
is to process the data faster within the array to give out a quicker response to the incoming request. The latter can be achieved by refining the caching algorithm implemented on the storage array for data processing.

The storage disks on which the actual data resides have very high access times and hence operate at slower speeds as compared to the speeds at which the storage processors can process the data or service a command. This causes a slowing up of the application being run and can cause it to crash. The computer architects came up with a brilliant but a simple idea of introducing a smaller, faster memory between the processor and the physical memory to speed up this data access process. This memory is called a “cache” memory. They improve the write and read performance of the hard disk by buffering the requests from the processor; but giving the processor an acknowledgement for the request received and then transferring that request to the slower hard disk memory when it is idle or as per the destaging algorithm’s design. As long as the cache is able to service the processor’s requests, the cache is invisible to the application which thinks that it has a large, very fast memory servicing its requests. The cache and the hard disk form a memory hierarchy and could have multiple levels within each of them. We have implemented two caching algorithms and have compared their individual performances for various tunable cache-parameters for iSCSI transport and for FC transport. Both the algorithms and the overall cache behavior have been described in subsequent sections.
The highlights of this thesis project can be described as:

- Extending a few existing caching algorithms to a SAN environment after modifying them suitably for the same.

- Evaluating the performance of iSCSI protocol for remote synchronous mirroring with the modified caches which will help determine the effects of each of the algorithms on the performance variables and as to whether iSCSI is good enough to compete with FC for the selected application.

We have been discussing about FC, iSCSI, IP/FC SAN, caching techniques, mirroring, etc. so far. But what exactly is a Storage Area Network, what is a FC SAN or an IP SAN, what is the FC protocol or the iSCSI protocol, how do the interact with all the complex components of the SAN, how has all this come together in this thesis project, what are the experimental results and observations? All these questions are answered sequentially in the subsequent chapters of the thesis report. The rest of the document is structured as follows: Chapter 2 discusses about the background topics like the SCSI protocol and its working; FC protocol and its working; SANs and its types; iSCSI protocol and its working; Chapter 3 discusses about the literature review done at the initial stages of the thesis to determine the current status of the research being conducted in SAN and to identify any possible areas of interest for further investigation during the thesis process; Chapter 4 discusses about the caching algorithms and various modifications implemented for conducting iSCSI remote mirroring on that SAN box;
Chapter 5 talks about the experimental setup, the experiments conducted, the results and conclusions drawn from them; Chapter 6 talks about the conclusions drawn from the overall thesis implementation and experiments; and about possible future work which can be carried out to enhance the overall motive of the thesis topic.
Chapter 2

Background

2.1 The Small Computer System Interface Protocol

Small Computer System Interface (SCSI) performs the task of exchanging commands, status messages and block data between various host systems and storage devices. It acts as a link between the operating system and the peripheral devices [21]. Broadly speaking, SCSI protocol is used to request services from various types of I/O devices. It is basically a client-server architecture, wherein the client is called an “initiator” and the server is called “target” (figure 3). The initiator issues SCSI commands to the target requesting services from them. The “device server” present on the targets accepts these commands and processes them [20].
The SCSI-3 Architecture Model-2 (SAM-2) provides application compatibility for all SCSI interconnects & SCSI transport protocol mechanisms. The SCSI domain consists of SCSI devices & the service delivery subsystems which transport SCSI commands, data and other related information between the initiator and target. The main functional/structural components of the SCSI domain are depicted in figure 4. The SCSI domain contains one or more SCSI initiator device which services or initiates SCSI commands transmitted through one or more initiator ports. The domain also contains one or more SCSI targets which consist of logical units that service these commands received through the target ports transported to it through the service delivery subsystem. The service delivery subsystem appears as a single link to the SCSI devices for the transport of commands, responses and data. The application client within the initiator device generates the request/command traffic & also carries out task management functions. The SCSI commands, also known as “SCSI task”, within the initiator could be
invoked by an application or a file system, which is carried in the Command Descriptor Block (CDB) and are delivered to an object within the target known as logical unit (LU). In the target device for every logical unit, the task manager oversees the ordering/sequencing of all its tasks. The routing of the commands and task management functions between a LU’s task manager and the delivery subsystem is carried out by the task router. Each task set contains either or both tagged and untagged tasks [26].

Figure 4: SCSI structural/domain model [26].
The SCSI protocol distributed communication service reference model is depicted in figure 5. In each of its layers, the peer entities communicate with each other with the help of their respective layer protocols. The service request originates from the top-most layer also known as upper level protocol (ULP) & the layer which provides the service is called lower level protocol (LLP). The SCSI application protocol, which is a part of the SCSI application layer, invokes and processes SCSI I/O requests for clients & servers. These requests are transported via the SCSI transport protocol, which is also used by the client & server to communicate with each other. The service delivery subsystem resides at the interconnect layer level and it defines the interconnect subsystems, physical parameters of the signaling mechanism for the actual data transport between the two devices. The data may be transported in parallel or broken-up into frames for serial transport. Each SCSI transport protocol standard defines its own interconnect service interface [26].
Figure 5: SCSI protocol distributed communication service reference model [26].

The request-response transactions at the ULP level and its associated LLP level services for a client-server process at the SCSI application layer is shown in figure 6.

At the client side, the outgoing service request is generated at ULP level, which invokes a LLP procedural call (SCSI transport protocol service request) and the incoming response is perceived by the ULP as a signal sent by LLP which either indicates that the request has been serviced with an end to that transaction or that the service did not traverse the underlying interconnect fabric (SCSI transport protocol service confirmation). It could contain data or other parameters related to the status of the service request. At the server side, the incoming service request form the client is
perceived by the ULP as an asynchronous event indicator from the LLP (SCSI transport protocol service indication). This indication invokes a ULP reply which generates a procedure call to the LLP layer (SCSI transport protocol service response). This is a peer-to-peer response at the ULP level [26].

Figure 6: Request-response service model [26].

The SCSI CDB is up to 16 bytes in length and contains the command/request to be serviced by the device server. The overall CDB format is shown in figure 7. The group code & command code are part of the operation code field which determines the
kind of operation being requested in the CDB and the length of the CDB. The last byte is
the control byte which contains a vendor specific 2 bits, 3 reserved bits, 1 normal ACA
bit, 1 obsolete bit to request interrupts between sequential commands and 1 link bit
which indicates that the request/task is to be continued for multiple CDBs. The rest of
the n-2 bytes contain command specific parameters [26].

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Group code</td>
<td></td>
<td>Command code</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Command specific parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n &lt;= 16</td>
<td>Vendor specific</td>
<td>Reserved</td>
<td>NACA</td>
<td>Obsolete</td>
<td>Link</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7: SCSI CDB format [26].

The SAM-2 draws parallels with the Open System Interconnection (OSI) model.
Figure 8 describes the same and also about the SCSI parallel interface (SPI) mapping to
the OSI layers. The first four OSI layers are clubbed into a common SAM layer called
SCSI Interconnects which contains the transport technologies like Fibre Channel (FC),
TCP/IP, Ethernet, SCSI Parallel Interface (SPI), etc. The rest of the three OSI layers are
consolidated to form the SCSI Transport Protocols layer in SAM. It describes the
transport protocols which could be used for transporting the SCSI data like FCP, iSCSI, FCIP, iFCP, etc. If the SPI is used for transport, then the SCSI Transport Protocol layer is not used in the communication flow. The SPI actually combines the functionality of the physical layer & a part of the data-link layer of the OSI model. The SCSI application layer contains device specific commands and commands common to all SCSI implementations and accessed devices [24].

The service and device discovery process is very simple when using the SPI communication model. The SPI allows less number of attached devices and that too for a relatively short distance for a storage environment. The device discovery takes place via a test signal sent out on the SPI bus. It does not employ name resolution or address resolution as SPI technology does not support device names and supports only OSI Layer 2 addressing. The attached devices respond to this test signal, after which the initiator sends out an inquiry command to these responding devices. The devices reply back with all necessary configuration information pertaining to them. Once the initiator receives response to the LUN discovery message from the devices, the SCSI I/O operations can commence [24].

We have seen from the above discussion that SCSI forms the basic building block for any client/server storage model, but has some limitations when being used independently as a SAN transport protocol. There is a limit to the number of servers that can be connected to a SCSI bus, which reduces the network density greatly. Also the
maximum length of the SCSI bus is a major constraint when it comes to building a large SAN. The use of the link-extenders is a possible work-around to tackle the SCSI bus length issue, but even those cannot be used excessively to provide a good and reliable SAN. But these limitations do not in any way diminish the importance of SCSI protocol for any storage system.

**Figure 8: Parallels between OSI reference model & SAM-2 [24].**
The simplicity of the SCSI technology as compared to other networking technologies remains the prime selling point for SCSI. The FC SAN and iSCSI only replace the SPI with an underlying network, over which the SCSI protocol is used for communication. This arrangement works smoothly due to the fact that during the whole communication process, the switching of the data from SCSI buses/cables to the underlying transport network is well concealed from the applications and higher layers of the operating system.

2.2 The Fibre Channel Protocol

With the above constraints being true for SCSI, the Fibre Channel (FC) protocol was used to construct the storage networks that we know off today. Let’s be clear that though FC is not the only technology being used currently for realizing a SAN, it is still the most widely used and most popular choice in the high-end SAN market.

The FC protocol stack is shown in figure 9. It is comprised of five layers, FC-0 to FC-4. The FC-0 layer defines the physical parameters of the signaling and transmission channel being used. FC uses serial transmission in contrast to the SCSI parallel bus transmission. In the SCSI parallel buses, the signal skew limits the speed with which the bits can be successfully transmitted over the data lines, which is totally avoided in FC
due to serialized transmission. FC SANs use both copper as well as optic-fibre cables (OFC) as the physical medium, of which the latter is more prevalent now-a-days due to its obvious merits like bandwidth, link-length, reliability, etc. The FC-1 layer defines the encoding techniques to be used for data transfer over the FC link. It uses the 8b/10b binary encoding technique for higher transmission rates and to reduce the jitter introduced in the communication link. This technique allows for a better clock synchronization too.

The FC-2 layer regulates flow control, maximum data unit size and defines the application specific service classes. It could in itself be subdivided into three sub-layers. The highest of these sub-layers defines a logical communication connection for the end devices involved in the transaction. This layer also ensures an in-order delivery at the receiver of all the frames being sent form the transmitter and also carries out defragmentation of larger data units to fit into the agreed frame size. It also handles error correction. The flow control is implemented using the “credit model” i.e. the receiver specifies the number of frames that the sender can send without waiting for an acknowledgement, also called as “credit”. After this credit limit is reached, the sender has to wait for acknowledgement for at least a few of the previously sent frames before sending any new frames. The two end devices decide upon the end-to-end credit before the actual data exchange begins for an “end-to-end flow control” mechanism, whereas it is done on a per-link basis for the “link flow control” mechanism. FC-2 layer also
defines six data exchange service classes. Class 1 is for connection-oriented communication between two ports and is rarely supported by the products available in the market. Class 2 and 3 are for packet-switched/datagram services and are most supported service class by nearly all the products in the market. Class F is reserved for the inter-switch data transport within a network. The FC-3 layer is currently a non-functional layer but has been proposed to be expanded to carry out functions like data compression, multipathing, data encryption, path striping for multiport end devices, mirroring and other RAID technologies.
The FC-4 and the ULP layers define the kind of application protocols that would be serviced by the underlying network. They map the application protocols onto the underlying FC network by defining the service classes to be used and the juxtaposition of the upper layer data onto the FC exchange sequence mechanism.
Figure 10 depicts an example of this mapping mechanism for an easy transition from one protocol to the FC network.

![Figure 10: Transition from SCSI to FC SAN](image)

The SCSI to FC transition is achieved by replacing the parallel SCSI protocol with the Fibre Channel Protocol (FCP) to map it onto the underlying FC network. This FCP device driver serializes the SCSI parallel data transmission and implement the SCSI daisy-chaining too. The FC model is a very loosely defined model on its own and does not map smoothly to the OSI reference model. It is worth noting that even though in the ANSI standards, the FC-3 layer is placed above the FC-2 layer, in practicality; the two layers could be termed as peer-layers. This is so because the FC-2 node-addressing feature acts as the building block for FC-3’s Hunt Group functionality, but at the same time, FC-2 for its smooth operations requires the Link Services which are a part of the FC-3 layer. Thus FC-2 & FC-3 are co-dependent layers and thus do not strictly follow
the OSI paradigm of layer-abstraction. Also it can be seen that the network layer from the OSI model is not entirely replicated in the FC model, but it has a few additional external protocols like the link services and fabric services layers which execute some of these network layer functionalities [24]. Figure 11 shows the parallels between the OSI reference model and the FC model.

![Figure 11: Parallels between OSI reference model & FC model [24].](image)

The link services and the fabric services layers complement the FC protocol stack and are equally important in maintaining the FC network infrastructure. They oversee the essential services like login, addressing, name servers, etc. The three staged
login mechanism sets up the initial authentication and connection verification processes between the two communicating ports before beginning the actual data exchange process. The first stage is called fabric login (FLOGI), is an essential phase for the point-to-point and fabric topologies. After the initialization of the link a session is established between the Node-port and its corresponding Fibre-port. The buffer-to-buffer credit information is also exchanged during this phase. After this phase the N-port login (PLOGI) phase is executed which sets up a session between two Node-ports and is a must for FC-4 data exchange. The last phase is the Process login (PRLI) which sets up a session between two FC-4 processes for the two different Node-ports [23]. Figure 12 shows the FCP login process.
The FCP uses a 64-bit name identifier for its devices. There are actually two name formats specified in FCP, the first one where every 64-bit identifier is unique throughout the world, World Wide Names (WWN) and the other which are re-assignable within each separate network, Fibre Channel Names (FCN). But generally this nomenclature is seldom followed and all the 64-bit tags are termed as WWNs. The WWN is further classified into a World Wide Port Name (WWPN) which is given to every port and that entire device is assigned a World Wide Node Name (WWNN). This
arrangement facilitates the distinction between the ports involved currently in the communication for a multiport device. The future proposed application for this arrangement is the multiport path striping for distributing the data over several redundant ports to increase the effective communication throughput. For the fabric topology, the transmitter and receiver of a FC frame is distinguished using a 24-bit port address (N-Port identifier and N-Port_ID) which is issued for each WWPN during fabric login process. In the arbitrated loop topology, each WWPN is assigned an 8-bit Arbitrated Loop Physical Address (AL_PA) address.

The network management and maintenance is carried out by the fabric services which are serviced using the FC-2 frames and have predefined addresses. All switches must service any fabric login requests under the fabric login server’s address. The information about the states of all the active devices in the FC network is managed using the State Change Registration (SCR) issued by an active N-Port and the subsequent Registered State Change Notification (RSCN) sent out by the fabric controller service to all its registered N-Ports. The name server facility is provided at the Simple Name Server (SNS) address. A list of all the registered N-Ports and their properties like their WWPN, WWNN, port address, supported service classes, supported FC-4 protocols, etc. can be obtained by logging into the above address and querying its database. Table 1 gives a list of the fabric services and their addresses. The service-requesting port needs to log into the service’s address before it can use that service.
Table 1: FC auxiliary services [23].

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF FF FF</td>
<td>Broadcast addresses</td>
</tr>
<tr>
<td>0xFF FF FE</td>
<td>Fabric Login Server</td>
</tr>
<tr>
<td>0xFF FF FD</td>
<td>Fabric Controller</td>
</tr>
<tr>
<td>0xFF FF FC</td>
<td>Name Server</td>
</tr>
<tr>
<td>0xFF FF FB</td>
<td>Time Server</td>
</tr>
<tr>
<td>0xFF FF FA</td>
<td>Management Server</td>
</tr>
<tr>
<td>0xFF FF F9</td>
<td>Quality of Service Facilitator</td>
</tr>
<tr>
<td>0xFF FF F8</td>
<td>Alias Server</td>
</tr>
<tr>
<td>0xFF FF F7</td>
<td>Security Key Distribution Server</td>
</tr>
<tr>
<td>0xFF FF F6</td>
<td>Clock Synchronization Server</td>
</tr>
<tr>
<td>0xFF FF F5</td>
<td>Multicast Server</td>
</tr>
<tr>
<td>0xFF FF F4</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFF FF F3</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFF FF F2</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFF FF F1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0xFF FF F0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

A FC host receives its address and operating parameters during its initialization process while logging into an FC switch when it comes online for the very first time.

Target discovery is carried out using FC SNS after which it establishes an FC connection with the discovered devices followed by FCP session establishment and LUN discovery.

During the protocol translation, each SCSI command issued by the operating system in a host/target is converted into an FCP I/O operation, identified by its Fully Qualified Exchange Identifier (FQXID) [24]. The basic communication takes place in form of frames which contains payload, transmitting port’s address, receiving port’s address, link
control information, frame delimiters and CRC information. Figure 13 shows the FCP’s frame structure.

![Figure 13: FCP’s frame structure [27].](image-url)
2.3 The Storage Area Networks

2.3.1 Fibre Channel Storage Area Networks

The term Storage Area Network (SAN) is generally associated with the FC technology as the underlying transport mechanism. A SAN could be generalized to be any network designed specifically to transport only the block-level storage protocols and their data [24]. It provides a very high data availability, performance, reliability and scalability. But at the same time it also has a considerably high acquisition cost and maintenance costs and so is generally affordable by large enterprises and not so much by the small and medium sized businesses. A FC SAN is preferred instead of an IP SAN in case of a mission-critical application or for servers that mandate high bandwidth like data centre environments which service business-critical data.

There are three main topologies associated with the FC SAN. The “arbitrated loop” topology, though not used very frequently now-a-days, still remains an important FC topology for a few installations like the I/O buses within the disk subsystems. In an arbitrated loop, the servers and the storage devices are connected in the form of a ring with each other i.e. communication can take place in only one direction at a given instant of time with in that ring and that too only between any two devices of the ring and other
devices have to wait/arbitrate for the bus until it is free. A single arbitrated loop supports a maximum of 126 individual devices (servers or storage devices), but can be expanded by cascading the “hub devices” in each of the loops. The hub devices only simplify the ring-cabling of the loop and are in fact invisible to the end devices themselves. There are two types of arbitrated loops, public and private loops. Figure 14 shows these types. A public loop, as the name suggests, can be connected to a fabric topology via a switch, whereas a private loops is a closed loop and cannot be interfaced with the fabric. In general, the devices with NL-ports on them are called public loop devices as they can communicate in both arbitrated loop protocols and fabric protocols and thus can negotiate the loop first to establish a connection with the switch in the loop and then can negotiate the fabric to establish connection to the end device. The devices with just the L-ports are called private loop devices as they can talk only the arbitrated loop protocol and so cannot communicate with the fabric even if they are a part of a public loop [23].
The “point-to-point topology” for a FC SAN is basically a direct connection between two devices which cannot be expanded further. Its only main advantages over a SCSI cabled server-centric IT architecture is the maximum transmission distance and robustness of transmission. Due to this the shared storage can be kept at a safe distance from the application server in a hazardous production environment [23].

The “fabric topology” for a FC SAN is the most commonly used and is highly flexible and scalable of the three FC topologies. The end devices are attached to the network fabric using the FC switches. One or more switches (generally not more than five) could be used depending upon the sale of the SAN in question. During the routing
of packets, the FC switch in fact can forward an incoming frame even before being fully received by the switch itself; this is called cut-through routing. In case of more than one switch being used in the network, the Inter-switch links (ISLs) become the bandwidth bottlenecks, but that can be somewhat alleviated by using frame buffers within the switches. A FC switch also provides additional services like name server, zoning and aliasing [23]. Figure 15 shows a common fibre topology with three servers connected to three other disk/storage devices through two FC switches. The switches are connected using ISLs.

Figure 15: Fibre topology FC SAN [23].
Aliasing is nothing but using alias names for the 64-bit WWNs and 24-bit port addresses for the sake of convenience. Every end device that wants to attach to a network must first register itself with a switch to which it reports all its information. The switch then stores this information and provides a name-server service to all its attached devices who want to discover other active devices connected in the SAN. The zoning services are used to divide up a SAN into sub-networks for better network administration. Zoning also allows to isolate incompatible host bus adapters (HBAs) within a single zone, allows limited visibility and access for the devices in a specific zone, thus providing data access control and security. SANs running critical applications generally use complementary enterprise-class, highly resilient FC switches with mutual traffic failover to avoid any single points of failure in the system which could possibly affect the whole SAN and all the applications running on it and thus costing millions of dollars in lost business or even material losses.

For old devices or tape libraries, they cannot be directly interfaced with an FC SAN. So we need to use FC-to-SCSI bridges which as the name suggests, creates a connection between FC and SCSI. Even a FC SAN is limited by the length of the FC cabling to up to several tens of kilometers. For a larger distributed SAN, we need to use link-extenders which basically transmit the FC frames over a MAN/WAN using ATM, SONET or TCP/IP protocols. Such a SAN is severely affected by large communication latencies due to the bandwidth bottlenecks of the MAN/WANs and so cannot be used in
time-critical applications, but can be profitably used for other less exacting applications. In a FC SAN, the main interoperability issues arise with respect to the FCP device drivers. For a SAN, it is essential that the single storage device is able to service parallel requests from several servers simultaneously. With so many different manufacturers for every component being available, it is imperative that these components are interoperable. Also the FC SAN manufacturers must support various configurations as per the customer’s requirement and the availability of the components. FC SANs still occupy a major market share when it comes to enterprise-level storage needs and for applications with very high performance and throughput requirements. But with advent of newer breakthroughs (10GbE links, TOE cards, product support being offered by most of the major players, etc.) in the IP technology and with maturing of the Ethernet technology, the IP SAN solutions are fast becoming popular with enterprise and SMB users.

### 2.3.2 Internet Protocol Storage Area Networks

The Internet Protocol SAN (IP SAN) is easier and more affordable to implement and maintain than traditional a FC SAN. As pointed out earlier, the performance of an IP SAN is surely getting to a point where it can be comparable to that of the high-performing FC SAN. IP SAN leverages the advantages of using the ubiquitous TCP/IP...
network and so has no distance limitation (but practically it needs to have an acceptable latency for the application being serviced), lower costs (hardware used here is much cheaper), easily available expertise and trained personnel, easy to administer and manage, high availability (multiple connection paths), maximum resource utilization by sharing devices across heterogeneous environments. While expanding an already existing storage network, using an IP SAN is a better option for quicker recovery costs with operational savings and a quicker deployment time. Most of the major operating systems have now started supporting many IP storage protocols, especially iSCSI. Even Microsoft has come up with a number of tools that work in synchronization with external storage systems and so a Windows Administrator can now easily manage and administer an IP SAN. This reduces the cost of training new personnel even more and also reduces the learning curve while implementing a new SAN. Out of the many IP storage protocols available today, Internet SCSI (iSCSI), Internet FCP (iFCP) and Fibre Channel over IP (FCIP) are more prevalent, with iSCSI being the most popular amongst them. Figure 16 compares the frame formats and figure 17 compares the protocol stacks for FCP and the three protocols.
Figure 16: Frame formats comparison for FCP, iFCP, FCIP and iSCSI [23].

Figure 17: Protocol stacks comparison for FCP, iFCP, FCIP and iSCSI [23].
2.3.2.1 Internet Fibre Channel Protocol

Internet FCP (iFCP) is a gateway-to-gateway protocol which maps FCP onto the TCP/IP network. The supposed value behind doing so was to only replace the existent FC network and FC storage infrastructure with IP/Ethernet network, thus protecting the investment and also expanding the infrastructure at the same time. This was actually an elegant idea, but the complexity of the protocol itself and the fact that it doesn’t provide any new additional features when compared to the other two IP storage protocols (FCIP, iSCSI), have undermined the progress of this technology. But nevertheless, it is an important IP storage technology and warrants at least some introduction in the context of this literature. Figure 18 shows iFCP’s network architecture. It basically emulates the FC switch connection for a FC end device, and then maps its 24-bit fabric address to a unique IP address and uses TCP/IP for reliable transmission of data over the network instead of the traditional FC-2 layer.
iFCP uses two modes of addressing, address transparent mode and address translation mode. The former accepts normal FC addresses and thus emulates a conventional FC fabric. Now in this case, if new switches or separate fabric is added to the existent SAN, then a new domain ID needs to be assigned to that switch and also its native devices need to re-login to obtain the new 24-bit N_Port addresses. This creates excessive convergence delays for large fabrics and causes transients in the network, which is always undesirable. To avoid such a situation, address translation mode is used. Here, the iFCP gateway assigns the N_Port address to its local devices as usual and also assigns one to the remote devices too, thus they appear as local resource to the native
devices. These proxy addresses remain local to that particular gateway and are not propagated across the SAN. Hence the task of address translation lies squarely upon the local gateway. The gateway maintains a mapping of the actual N_Port address of the remote device and its unique IP address, with the proxy N_Port address as the key to this lookup table. So in the outbound FC frame, the proxy address is replaced by the actual port address and the IP address in the frame header by the local gateway.

As iFCP emulates a FC network on the TCP/IP, it needs to also emulate all the standard FC services like fabric login, SNS registration, etc. So for the login phase, the protocol must first determine the locality of the destination port. For a local destination port, the PLOGI request is just forwarded to it, but for a remote destination port, before the request is forwarded to it, a TCP/IP connection to that device must be setup first, with the maximum data transfer size adjusted to be less than 1500 bytes (TCP/IP standard for avoiding packet fragmentation) [21]. Figure 19 shows this above process.

![Fabric login process of iFCP](image)

**Figure 19: Fabric login process of iFCP [21].**
Similar interception and translation is required for the SNS queries too. Here the gateway queries the external iSNS server with the FC SNS request. The external iSNS server is analogous to an IP DNS server. It returns a list of permissible targets to the gateway. The gateway formats the SNS response for the initiator and also keeps a copy of this list for updating the address translation table for future communication between the initiator and potential targets. For all these services, the respective traditional standard addresses are maintained even by iFCP [21]. Figure 20 shows the SNS request processing for iFCP.

Figure 20: SNS request processing for iFCP [21].

The iFCP gateway must administer connection establishment and maintenance, error handling and security along with its other functions. The TCP connections are managed by exchanging TCP link service messages between the iFCP gateways. For an active connection between the initiator and the target, the gateway provides a bound
dedicated TCP connection. Data security between initiators and targets can be accomplished by use of public/private key authentication managed by iSNS servers or by using discovery domains for the authorized initiators which restrict the session establishment to specific target devices or by means of IP Security (IPSec) for authentication and data encryption or by even using firewalls between the iFCP gateways and the IP router interfaces. The metro FCP (mFCP) is a variant of iFCP which uses UDP/IP as the transport protocol. Its architecture, addressing modes and other services are similar to iFCP except for the flow control mechanisms. Combining mFCP and iFCP could provide a high performance network for the data centers and metropolitan storage networks.

2.3.2.2 Fibre Channel over Internet Protocol

Fibre Channel over IP (FCIP) is another IP storage protocol that we are going to take a brief look at. FCIP basically creates a tunneling connection between the two FC SAN islands (each having its own IP address) over the IP network cloud. FCIP encapsulates the FC frame from the initiator FC SAN and tunnels it across the IP network to the remote SAN island where it is de-encapsulated to give back the native FC frame. Here the FCIP devices and the IP cloud is transparent to the FC switches and at
the same time the FC content being transported over the IP network remains transparent to the network. FCIP protocol has the least amount of IP content in its functioning. FCIP uses the standard fabric E_Port switch interface and the FC Fabric Shortest Path First (FSPF) routing protocol for FCIP bridge to FC SAN interfacing. Figure 21 shows a standard FCIP tunneling connection joining two FC SAN islands by means of connection abstraction.

During encapsulation process, the FC frame is left unaltered. It then adds an encapsulation frame header which includes frame length value, protocol and version level fields. End device synchronization is maintained by validating the FC EOF, FCIP Header and FC SOF. In case of a connection failure (tunnel collapse), there is no automatic link recovery, as is the case with homogenous IP networks, due to which the two SANs will reestablish their own isolated states and send relevant state change
notifications to the end devices. The session is recovered only with an external intervention by the FCIP devices. Errors created by the end devices are not handled by the FCP devices. They handle only the IP network errors, by dropping the erroneous packet. Routing or switching services have to be carried out by the intermediate IP routers and is not the responsibility of the FCIP devices. For congestion control and flow control, FCIP relies on the inherent flow and congestion control mechanisms of the underlying TCP network. For ensuring quality of service over the IP network, FCIP uses the enhancements to the IP technology like differentiated services architecture (DiffServ) or multiprotocol label switching (MPLS). The transparent nature of the IP tunnel to the end devices and that of the FC data to the tunnel makes it difficult to use IP network management tools to their fullest capabilities and also to trouble shoot any problems arising within the FCIP transit. Inability of the protocol to automatically recover from a temporary IP network connection interruption between the FCIP end devices makes it vulnerable to considerable business discontinuity which could be a major disadvantage while marketing a technology.
2.4 Internet Small Computer System Interface

2.4.1 Protocol Overview

As we have seen earlier, FCIP has its fair share of problems with vendor-specificity and interoperability issues being the foremost off them. Also iFCP is still regarded a nascent technology with lot of testing needed for a full-fledged launch as a viable solution and is not as prevalent as iSCSI as an IP storage option for FC SANs. The overall protocol interaction is fairly simple. The application or a file system within the initiator device generates a service request, which is converted by the SCSI driver into simple SCSI commands (SCSI CDBs). Now as we are not using the SPI or the FCP for transport of the CDBs, the iSCSI device driver intercepts these commands and with the help of a host bus adapter card (HBA), transmits them across the TCP/IP network in the form of iSCSI PDUs wrapped in an Ethernet frame. The HBA at the target device converts these iSCSI packets into SCSI CDBs with the help of the iSCSI driver, passes them over to the SCSI device driver which then sends it to the appropriate logical unit (LU) for request processing. The resultant data is transported back to the initiator in a similar fashion, but in the other direction [20]. Figure 22 shows this initiator-to-target command flow. Figure 23 shows the iSCSI layering structure. In figure 23, the I/O
request flows from the application, downwards through the various layers in the initiator, through the Ethernet network and then upwards through the protocol layers in the target to the LU. If data is to be sent back for the initiator’s request, it follows the same path in the reverse direction [20].

Figure 22: iSCSI initiator to target command flow [20].
The iSCSI Protocol Data Unit (PDU) is the basic unit of transfer and carries the SCSI CDBs, LUN number and various data components from the initiator to the target and the response or reply from the target to the initiator. The PDU is then encapsulated in a TCP segment, IP header and Ethernet frame (in that order). Figure 24 shows a typical PDU format. The first 48 bytes is the Basic Header Segment (BHS), which contains opcode, a few flags, lengths of the additional header segment (AHS) and data segment. The opcode determines as to whether the target LU number, initiator task tag and CDB would be carried in the PDU. An “immediate” request is signified by setting
the “I” bit to 1 in the PDU. The rest of the PDU is made up of the optional fields like AHS, header digest (a CRC value), data segment, data digest (a CRC value) [20].

![Figure 24: iSCSI PDU format [20].]

### 2.4.2 Login and Session Establishment

To setup an iSCSI connection between an initiator and a target, firstly a TCP/IP connection needs to be established successfully between the two, then the endpoints need to be verified and authenticated and then the communication parameters need to be exchanged by the endpoints. This above described process called the “login” process, which is a part of the session establishment process. The session establishment process is
comprised of three phases. They are security negotiation phase (SNP), login operational negotiation phase (LONP), full-feature phase (FFP). Unless the FFP is reached, the actual communication of SCSI data cannot take place between the endpoints. There could be multiple login requests, each of which is met with a login response which either accepts the request or rejects it. In the normal login session, the initiator and target are predetermined and have their own iSCSI names. In the discovery login session, the initiator tries to identify the iSCSI targets available to it and query their port addresses and IP addresses. During login, the login request PDU and the login response PDU are exchanged between the initiator and target [20].

Figure 25 shows the login request PDU format. The T or Transit bit indicates readiness of the initiator to move to the next phase of the session establishment process when set to 1. The C or Continue bit indicates that the text in the current login PDU will be continued on subsequent PDU when set to 1. CSG specifies the current stage and NSG specifies the next stage of the session establishment phase. NSG is valid only when T is set to 1. Version-Max indicates the maximum version supported and Version-Min indicates minimum version supported. DataSegmentLength indicates the length of the data payload contained in the current login request PDU. Initiator session ID (ISID) is an initiator-defined field which classifies the session as unique within the initiator system and is used on the subsequent session connections to tie them to this base session. The Target session identifying handle (TSIH) helps in linking any subsequent connection
logins originating from the same physical portal as the previous connection for which it was set in the last target response during the leading login, or from a different physical connection from the same portal group. The Initiator task tag (ITT) is used by the participating devices to identify the command to which they are responding and is used in every command PDU. Connection ID (CID) is the unique connection identifier, common for all the login requests within the particular session. Command sequence number (CmdSN) is initialized (usually to a non-zero value) for initial leading login or subsequent commands in the command stream after the session establishment. It remains static for entire initial login process for a session. Expected status sequence number (ExpStatSN) denotes the status sequence number for an old connection which has been restarted. DataSegment may contain some basic parameters specified by the initiator for enabling the target to verify the validity of the initiator for letting it use the target’s resources [20].

Figure 26 shows the login response PDU format. The T, C, CSG, NSG, Version-Max, TSIH, ITT and DataSegmentLength have same functionality as in the login request PDU. Version-Act field indicates highest version which can be supported by both target and initiator. If the target cannot support the specified version, it rejects the login by setting Status-Class and Status-Detail fields and sets the Version-Act to the lowest version that the target can support. Status sequence number (StatSN) is the starting status sequence number for the connection in the first login response PDU. Expected
CmdSN (ExpCmdSN) specifies the current value of the CmdSN register. Maximum CmdSN (MaxCmdSN) is sent to the initiator by the target device to keep the CmdSn registers with in the target and the initiator at sync. Status-Class and Status-Detail fields indicate the progress of the login phase; a zero-value indicates success, whereas a non-zero value indicates the occurrence of an exception. They help in facilitating the exception handling for the initiators, by identifying the various exceptions using a combination of unique values in the above fields [20].

![Figure 25: iSCSI login request PDU][20].
Each iSCSI connection session process starts off with a TCP/IP connection establishment for a particular address and port using the discovery process using a normal socket call. The Internet Key Exchange (IKE) process may or may not be invoked depending upon the IPsec function being invoked by that particular socket call. IPsec will establish the requisite security environment at either side based upon their requirements. Once a working session is established, the above described login phase can begin at the initiator side. Before the actual login begins, an authentication or verification phase is carried out. It includes “to-from” verification stage in which the target checks the login request text field to verify that specifically its name is mentioned
in it or else the connection is dropped. Other authentication routines like Challenge Handshake Authentication Protocol (CHAP), Secure Remote Password (SRP) protocol, Kerberos V5, Simple Public Key Mechanism (SPKM) 1 or 2 can also be implemented at both the ends [20]. Error handling in iSCSI is an important requirement for reliable IP SAN functioning. FC SAN can assume a very low error-free and trouble-free dedicated messaging network, but such a liberty cannot be granted to iSCSI networks, as its main selling point is being able to use the ubiquitous Ethernet network which is far from being error and trouble free. In the iSCSI network, both the initiator and target have the ability to buffer commands and responses till they are acknowledged and thus be able to rebuild the missing or corrupted PDU (digest error or format error) selectively for retransmission which is in-line with the inherent packet-level recovery facility of TCP, which has been extended now to SCSI block data. For recovering the missing PDUs, sequence number acknowledgement (SNACK) PDU is used which contains the number of missing PDUs calculated from last valid PDU received. This error handling takes place at the iSCSI protocol level. If the error occurs at the TCP connection layer, the connection recovery takes place through command restart. For multiple TCP connections the errors might require connection reconstruction through session re-login [21].

The device discovery in iSCSI takes place using the Internet Storage Name Server (iSNS) lightweight protocol, deployed on centralized iSNS servers, IP storage switches and target devices; which leverage database objects of FC SNS and the DNS
facilities of IP networks. It enables features like name registration, discovery, zoning, state-change management and management of IP storage resources. The device discovery process begins with device registration, which is later used to build the database of iSNS clients for discovery domains. The value registered includes WWN or iSCSI name, device’s IP address, alert-initiating state change bitmap. Once the discovery domains are defined, the server notifies the existing clients that a network reconfiguration had occurred using a State Change Notification (SCN) message. This prompts the client to query the iSNS for available resources. The iSNS server uses an entity status inquiry (ESI) message to regularly poll all the registered devices at predetermined intervals for their availability status and state changes. If a response to this is not obtained from a particular device even after a specific number of retries, then that device is deregistered. This helps to maintain an updated database of all the active devices and facilitates to report the changes to the clients in the network. iSNS servers also facilitates provision of host security services by storing and later distributing the X.509 public keys of registering devices to other registered devices in the same domain. Hence during the login phase (i.e. right after device discovery), the devices can exchange and authenticate the public keys, private keys and digital signatures [21].
2.4.3 Connection State Diagrams

There are well-defined and distinct states that an iSCSI connection and session transitions through during its lifetime i.e. right from its setup to its cleanup. For a single session, its state transitions could be dependent on the states of other iSCSI connections participating in that session. The iSCSI RFC [25] describes the standard state transitions for an initiator and a target during for an iSCSI session. The state descriptions for the standard connection state diagram for an iSCSI session are:

S1= FREE => State on instantiation or after successful connection closure.

S2= XPT_WAIT => State for the initiator while it is waiting for a response to its transport connection establishment request from the target. This is an illegal state for the target.

S3= XPT_UP => State for the target while it is waiting for login process to commence. This is an illegal state for the initiator.

S4 = IN_LOGIN => State while waiting for the login process to conclude, possibly involving several PDU exchanges.

S5 = LOGGED_IN => State while in full feature phase, waiting for all internal, iSCSI and transport events.

S6 = IN_LOGOUT => State for the initiator while it waits for a logout response; for the target, while waiting for an internal event signaling completion of logout process.
S7 = LOGOUT_REQUESTED => State for the initiator while it waits for an internal event signaling readiness to proceed with logout; for the target, while it waits for the logout process to start after having requested a logout via an async message.

S8 = CLEANUP_WAIT => State for the initiator while waiting for the context and/or resource to initiate the cleanup processing for this CSM; for the target, while waiting for the cleanup process to start for this CSM.

Figure 27 shows the connection state transition diagram at the initiator side. The state transition description described in the RFC [25] for the initiator states are:

T1: Transport connection request was made like sending TCP SYN.

T2: Transport connection requested timed out, transport reset was received or successful logout response was received.

T4: Transport connection established, start of iSCSI login.

T5: Final iSCSI login response with a state-class of zero was received.

T7: Events like non-zero status-class login response received or login timed out or transport disconnect indication received or transport reset received or transport timeout received, or successful logout response received can cause this transition.

T8: An internal event of receiving a successful logout response on another connection for a "close the session" logout request was received, thus closing this connection requiring no further cleanup.
T9, T10: Internal signal to start logout process received, thus prompting an iSCSI logout to be sent by the initiator.

T11, T12: Async PDU with AsyncEvent “Request Logout” was received.

T13: An iSCSI successful logout response was received, or an internal event of receiving a successful logout response on another connection for a "close the session" logout request was received.

T14: Async PDU with AsyncEvent “Request Logout” was received again.

T15, T16: Events like transport reset or transport disconnect or receiving async PDU with AsyncEvent “Drop Connection” or “Drop all connections” can cause this transition.

T17: Events like receiving failed logout response or logout timeout or any of the events specified for T15, T16 can cause this transition.

T18: An internal event of receiving a successful logout response on another connection for a "close the session" logout request was received.
Figure 28 shows the connection state transition diagram at the target side. The state transition description described in the RFC [25] for the target states are:

T3: Received a valid transport connection request to establish transport connection.

T4: Initial iSCSI login request was received.

T5: Final iSCSI login request to conclude login phase was received, after which target sends final iSCSI login response with a status-class of zero.

T6: Events like timed out waiting for an iSCSI login or received transport disconnect or reset can cause this transition.
T7: Events like non-zero status-class login response received or login timed out or transport disconnect indication received or transport reset received or transport timeout received, or successful logout response received can cause this transition.

T8: An internal event of sending a successful logout response on another connection for a "close the session" logout request was received, or an internal event of a successful connection/session reinstatement is received, thus prompting the target to close this connection cleanly.

T9, T10: An iSCSI logout request was received.

T11, T12: An internal event that requires the decommissioning of the connection is received, thus causing an Async PDU with an AsyncEvent "Request Logout" to be sent.

T13: An internal event was received that indicates successful processing of the logout, which prompts a successful logout response to be sent; an internal event of sending a successful logout response on another connection for a "close the session" logout request was received; or an internal event of a successful connection/session reinstatement is received. In all these cases, the transport connection is closed.

T15, T16: Events like transport reset or transport disconnect or internal emergency events which cause sending of async PDU with AsyncEvent “Drop Connection” or “Drop all connections” can cause this transition.
T17: A failure of logout processing causes a failure logout response to be sent. Any of the event specified for T15, T16 can also cause this transition.

T18: An internal event of sending a successful logout response on another connection for a "close the session" logout request was received, or an internal event of a successful connection/session reinstatement is received. In both these cases, the connection is closed.

Figure 28: Target connection state diagram [25].
Chapter 3

Related Work

With belief in the current industry trends and the various opinions expressed by the industry experts regarding the promising future of iSCSI, it is important to review some of the primary performance evaluation-related literature and literature on various modifications suggested to improve the performance of the iSCSI protocol. We can broadly classify the reviewed literature as those related to:

(a) iSCSI overview: [1];

(b) PE of iSCSI v/s FC: [2], [3], [4] and [11];

(c) PE of iSCSI v/s other IP solutions: [12], [13] and [14];

(d) Modifications for performance improvements for iSCSI and their PE: [5], [6], [7], [8], [9], [10] and [15];

In [1], authors explain the design aspects and decisions considered during the iSCSI protocol design and various issues related to the same. They justify the choice of TCP as the underlying transport protocol by sighting TCP’s reliable and in-order packet
delivery, ubiquity, scalability. They also propose use of multiple/parallel iSCSI connections for better utilization of available bandwidth and apparent benefits with respect to performance, throughput and QoS improvements in the future. This approach has been further explored in [5] and [7]. They also explain various iSCSI data recovery mechanisms with their respective pros and cons. The symmetric and asymmetric iSCSI models along with the clever use of Direct Data Placement mechanisms have also been explained. As iSCSI runs on IP, it can utilize the existent security mechanisms in IP family of protocols like IPSec. According to the authors, the main selling point of iSCI is its use of the existent TCP/IP infrastructure and its well established mechanisms for use of transporting the storage data over Ethernet and thus reducing the establishment and maintenance cost of SANs and providing a viable option to Fibre Channel (FC) SANs.

In [2], a commercial iSCSI software and hardware implementations is compared against FC protocol in a commercial environment. They have also analyzed the performance of iSCSI in SAN and WAN environments and have investigated the effects of underlying network conditions and the attached devices too. Their findings indicate that: (A) With in a SAN environment, for larger block sizes, the software implementation of iSCSI performs much better than the hardware version when compared to the FC; also the software implementation is affected by disk throughput and would benefit from specialized iSCSI target-specific hardware. Physical layer parameters (Ethernet frame size) affect the performance more than the network layer
parameters (TCP window buffer size). (B) With in a WAN environment, the network
layer properties and implementation of the iSCSI protocol affect the overall performance
more significantly as compared to the physical and protocol layer properties.

In [3], the QoS and delay requirements for asynchronous mirroring over MANs
and WANs have been evaluated. They have conducted experiments to determine the
effects of RTT, network QoS, request size, distance, congestion window size, etc. on the
performance of iSCSI over long, medium and short distance scenarios for the Ethernet
networks. Their analysis for single iSCSI operation showed that the absolute write-time
depends on throughput, RTT, bottleneck-link bandwidth, request size, distance,
congestion control algorithm of TCP, loss rate and processing time at the target. For the
dynamic iSCSI model, the interleaving of the transmissions performs much better than
non-interleaved model with respect to the mean write time for any network QoS and
traffic.

In [4], performance measurements were conducted on typical iSCSI subsystem
configurations to examine both block-level and file-level I/O performance and find out
the factors that affect it and identify the performance bottlenecks. Like the other studies,
the authors were able to demonstrate that Gigabit Ethernet-based iSCSI provides a
bandwidth and throughput performance comparable to direct FC disks for block-level
I/O access, but is also affected by the distance between the target and the initiator. Also
the iSCSI-based file access provides better performance than NFS, SMB/CIFS for smaller file sizes, but the performance deteriorates as the file size increases.

The authors in [5] have proposed performance and reliability improvements in iSCSI by using S-iRAID and P-iRAID. In this iRAID topology, the iSCSI RAID nodes with in the array form a high-speed LAN. The striped iRAID (S-iRAID) increases iSCSI performance and security considerably through parallelism. The parity iRAID (P-iRAID) improves reliability and performance by using rotated parity, which helps in easy data-recovery in case of a node failure.

Authors have introduced “iCache”, a cache scheme to improve the performance of current iSCSI implementations by using two-level hierarchical cache (NVRAM and log-disk) in [6]. iCache helps in increasing the reliability, storage data rate and reduces network bandwidth bottleneck by localizing the transactions and acting as a filter for disallowing unnecessary data traffic on the network. Experiments on popular benchmarks against typical iSCSI implementations, clearly validates the performance gains of using iCache, but with a possible bottleneck created by an un-optimized destage algorithm.

In [7], authors evaluate the behaviour of iSCSI initiators and overheads introduced by iSCSI in the I/O path. The numerical results show that iSCSIx3 (multiple iSCSI GbE targets/parallel iSCSI connections) outperform traditional iSCSI setups and is comparable to direct attached configuration on all the important industrial benchmarks.
due to large target buffer cache sizes, more number of physical disks. It also shows that TCP/NIC interrupt processing and buffer cache processing contribute significantly to the kernel overheads in the I/O paths.

In [8], the authors have proposed a novel approach to implement IP SAN, i.e. use of IP-addressable disks directly attached to the Ethernet network and acting as storage space. This provides unprecedented flexibility, scalability and availability which are comparable to FC SANs. The architecture uses a new proprietary protocol “xBlock” for communication between the storage controller modules (SCM) and the IP disk controllers (IPDC). IPDC gives the disks an IP access by providing low latency protocol conversion between a media generic storage protocol (xBlock) and disk protocol (ATA) at gigabit speeds. Using this architecture, the storage controllers could be accessed using iSCSI or NFS or CIFS.

In [9], the authors explore the performance of iSCSI against traditional FC SAN for the remote mirroring applications. The numerical results clearly show that the DCD enhanced iSCSI provides a cost-effective option to dedicated FC SANs without taking any performance hits. This DCD enhanced iSCSI along with write coalescing on the initiators side, reduces network traffic. It also provides better performance than iSCSI mirroring over WAN, LAN and local mirroring. The DCD architecture is relatively an old architecture and could well be substituted by the iCache technique in [5].
In [10], authors provide in-depth analysis of iSCSI and its overheads, system characterization of iSCSI initiator, performance comparisons against local disk access and remote NFS file access. Results on “Oprofile” benchmark show that writes are slower than reads for local disks and the reverse is true for iSCSI device. When compared to NFS, an unfair comparison as confessed by the authors, single reads are performed better for NFS than for iSCSI and opposite for single writes. Also iSCSI scales much better than NFS without a drop in system performance. On the basis of above observations, a few modifications were done to the iSCSI system to improve its performance. They resorted to use of jumbo packets on the GbE interface which improved the iSCSI write performance. A distributed file system designed and optimized for iSCSI which allows access to data directly on networked storage can have tremendous scalability but will be plagued by unacceptable delays while crossing the network for each access and due to excessive interrupt processing overheads, unless caching is used at both client and server side. It should also use a consistency model similar to the one used in AFS, like the “callback” mechanism. This file system can be used for data intensive applications on clusters. But using it for high performance servers is not a good idea.

We take a look at a new asynchronous remote mirroring protocol, “Seneca”, which supports write coalescing, asynchronous propagation and in-order delivery in [11]. Further, they evaluate the performance of their protocol and also verify its
correctness. It aims at utilizing the relatively low-speed and low cost WANs and still meet high availability goal and recover from most of the important failure scenarios. Seneca also uses a NVRAM and log-disk cache structure at both primary and secondary sites, which we have seen earlier. Update records and data blocks in primary log are transported in-order and in parallel to the secondary log. In case of failover of the primary, the secondary takes over and when the primary is up again, fallback occurs. During this failover, data inconsistencies between the primary and secondary are possible. In such a case, a failover might be completely prohibited resulting in data unavailability or continue with the inconsistency and minimize the data loss or live with data loss but remove the inconsistency. Authors conducted the protocol correctness tests to verify the coverage i.e. Seneca converges to a valid state after any sequence of events; safety i.e. sequence updates are consistent and occur in a prefix-manner; liveness i.e. under normal conditions, data will eventually be written in both mirrors. The performance analysis tests showed that the overall cost would be less by using writes coalescing and by delaying I/O during the mirroring operation.

Application throughput performance is a key metric in data replication along with reliability and availability. In [12], authors have carried out reliability and availability analysis of the IP-based (iSCSI) and optical-based (FCP) SAN extension solutions. The reliability metrics considered are down time and service failure rate. The analysis shows that the model which has just one link between a FC switch and edge IP
router has the lowest reliability and highest down time and so should not be used for mission critical applications. An IP network configuration consisting of dual redundant links between the FC switch and edge IP router via different aggregation points performs not as well as a similar optical-based extension; but if we are able to reduce the layer3 protocol convergence time, then we can get acceptable performance for mission-critical applications. With modifications similar to above, to a fully redundant configuration for IP solutions i.e. two FC switches connected to two different IP routers, provides the most resilient configuration if the cost aspect is in check and a performance comparable to the optical-based solutions. Thus we can infer that redundancy at the edge plays an important role to improve network reliability and with improvements to the layer3 convergence times for IP solutions, it can be successfully deployed in mission critical applications as a cheap and viable option for remote mirroring and replications instead of the optical solutions.

Authors compare the application throughput for iSCSI, iFCP, FCIP under various network conditions for replications in [13]. It has previously been observed that IP-based SAN extensions are generally suited for asynchronous replications over long distances rather than for synchronous ones due to lower delay tolerances for the latter. The experimental analysis showed that the throughput decreases hyperbolically for FCIP, iFCP and is even lower for iSCSI due to very high TCP processing and disk latencies, for which solutions are already in-place. Packet loss is minimal for SONET-based
networks, but a bit higher for IP networks. Available bandwidths have similar effects on iSCSI and iFCP. With increase in the maximum TCP window size, the throughput increases in varying degrees for iSCSI, iFCP and FCIP. The throughput also increases with increase in the number of TCP connections for iSCSI and iFCP, but is not supported in FCIP. The authors concluded that the IP solutions provide very high throughputs for distances up to 300km with default TCP features. With advanced features, they can provide higher throughputs than SONET-based solutions.

Authors in [14] carried out a battery of tests to characterize the distance data sharing using both native FC and IP based technologies. The authors state that the experimental results for on-campus distances with bonie++, PostMark, Imdd, CVFS and GFS shared file systems show encouraging signs about iSCSI’s performance if we consider the implementation ease and overall cost even though traditional FC outperforms it. Also the availability of TCP offload engines (TOEs) is believed to aid boost the performance, but bandwidth could take a hit at the same time depending upon the application being run. For larger off-campus distances, iSCSI performed well enough for a remote connection on the above test-benches. The authors conclude that iSCSI does provide a promising cost-effective alternative to FC for use in shared file systems enabling operation in heterogeneous environments.

In [15], authors have first identified the processing cost of each iSCSI protocol component; then replace the industry-standard CRC algorithm (Sarwate CRC) with a
newer and better “slicing-by-8” (SB8) algorithm; finally came up with a novel technique to implement interaction between TCP and iSCSI layers. Through their protocol analysis, they found that CRC generation and data copy operations between TCP and iSCSI were the main bottlenecks in iSCSI processing. So they infact came up with a new algorithm which interleaves the above two processes and thus saves on the overall iSCSI processing time. The new CRC generation algorithm, unlike the industry standard, slices the CRC values in each iteration and data bits into small terms and performs parallel lookups using these as indexes. So in effect this algorithm carries out lesser number of operations per byte of input stream and hence is faster. For the earlier mentioned interleaving scheme, the authors apply the principle of ILP (Integrated Layer Processing). Their experimental results show that the SB8 algorithm took lesser cycles to process an iSCSI read as compared to the Sarwate CRC for various read sizes. The processing costs were similar for both algorithms though. But with interleaving of data copy and CRC, the performance improvement is much more for SB8 than for Sarwate CRC. Also the experiments with the entire storage stack showed that the new optimizations and algorithm worked much better then the conventional standards used. The authors also plan to extend their study to analyze performance and scalability of their modifications for multiple CPU-cores, transaction-oriented applications as well as backup and recovery applications.
Chapter 4

Algorithm Implementation

4.1 DCD caching algorithm

In [16], the authors have described an efficient caching architecture called DCD, Disk Caching Disk, for optimizing I/O performance with very high reliability. The DCD architecture uses a three-tiered hierarchy for its memory organization. The first tier is a very small (hundreds of KBs to a few MBs) solid-state RAM buffer which is completely transparent to the file system application and so requires no operating system modifications for incorporating this new tier in the memory hierarchy. The second tier is a “cache-disk” drive, also called “log disk” which is fairly larger than the RAM buffer, but is much slower than the buffer because of the fact that disk drives have much higher access time than a solid state memory device like a RAM buffer. The third tier is the data-disk itself which is another set of slow disk drives on which all the data/files will
finally reside. Data disks are much larger than any of the above tiers and could be in the range of hundreds of Gigabytes to hundreds of Terabytes. The buffer will absorb in all the small and random write requests coming from the application above and will transfer it to the log disk (in a single write process) whenever the latter is idle or as per the destaging policy. This frees up the buffer at the earliest be available to service any further writes that arrive, creating an illusion of a large RAM for the host. The log disk then collects all these writes together (forms a “log”) and transfers it to the data disk whenever the latter is idle or as per the destaging policy. This log disk could either be a separate physical disk drive (better performance) or a logical disk partition on the data disk at the backend. The latter case gives a lower speed of transfer as the data has to now travel over the SCSI cabling between the storage processor to the backend disks rather than the SPI buses connecting the various parts within the storage processor during the first destaging stage (i.e. buffer to the log disk).

A write request is processed intelligently in this architecture. A fairly large write request will be directly sent to the data disk (bypassing the buffer and log disk all together) as if there is no caching mechanism in place at all. But for smaller requests, they are sent to the buffer and will follow the usual flow of data through the memory hierarchy. Now, there are two types of write-complete acknowledgements that could be employed here. The first one is when the data is buffered by the RAM, an acknowledgement of the receipt of the data could be sent to the host, and this is called
“immediate report”. The second one is when the data is committed to the data disk from
the log disk, that an acknowledgement will be sent to the host, and this is called “report
after complete”. The former has higher request service rate, hence higher throughput and
hence higher performance but a lower reliability than the latter. This is so because the
host can start sending more write/read requests to the disks as it gets faster
acknowledgements for its previous requests. The reliability is lesser because if the
communication between the buffer and the log disk fails before the request can be
written to the data disks or if there is system/power failure before this final transaction
occurs, then the data on the data disk will be inconsistent as the host will never know
about the data loss as it has already got an acknowledgement. This scenario occurs very
rarely as the time for which the data resides in the buffer is extremely small before it is
transferred to the log disk, which is a non-volatile memory disk. It can be avoided all
together by using a non-volatile RAM buffer (much costlier than conventional RAM) or
by employing the above stated report-after-complete acknowledgement scheme. The
engineering/office workload environments and the online transaction processing the
writes are usually small, random, bursty and have temporal locality. These writes are
first collected in the RAM buffer and then destaged to the log disk as soon as the log
disk is free, during this destaging, the controller continues to accept incoming writes and
buffers them in the RAM. This arrangement ensures minimum idle time for the log-disk
and maximum buffering capacity to the RAM buffer. The maximum time that a write
request would reside in the RAM would be the time required to write a log to the cache-disk which is not more than a few hundred milliseconds. For lower write traffic intensity, the rotation latency is larger but is compensated by availability of greater idle disk time due to the low traffic intensity. For higher write traffic intensity, the log collected is large enough so that the rotation latency becomes a very small percentage of the total actual writing time.

The destaging algorithm used for transfer between the log-disk and the data disk is Last-Write-First-Destage (LWFD). When an idle time is detected by the algorithm’s idle detector, it reads back a fixed length of data from the log-disk to an intermediate destage buffer, during which it reorders the data and then writes it to the data disks physical location. The destaging is suspended if a read or write request arrives and resumes only at the next data disk idle time slot. This algorithm ensures that the disk head is always physically close to the blank track on the log-disk, thus reducing the rotation latency for the first destage. The reading operation is a very simple in DCD. During a read request, the controller will first search the RAM buffer, then the log-disk and then the data disk in that order. If the data is found higher up in the hierarchy, better is the read latency. The study shows that the DCD cache performs better than the traditional disk systems.
4.2 RAPID caching algorithm

It stands for Redundant Asymmetrical Parallel and Inexpensive Disk Cache. This architecture provides redundancy with the help of a two-level backup cache and a single level primary cache for a write cache, along with a separate large DRAM read cache on top of the disk subsystem. This is an asymmetric architecture because the primary cache is a larger fast-write-fast-read NVRAM or DRAM and a smaller fast-write-slow-read two-level backup cache which is much cheaper than deploying the same redundancy by using costlier dual-write NVRAM caches. The backup cache is made up of a small NVRAM and a cache disk drive which could be implemented as a separate physical drive or as a logical partition on the backend data drives. This architecture of the backup cache is similar to the DCD cache in [16]. The backup cache NVRAM consists of a LRU Cache which stores the frequently-accessed data, two to four Segment Buffers and a Hash Table which contains location information for every valid data block in the backup cache. The cache-disk contains the less frequently data in it. During a write request, the controller sends the data simultaneously to the primary cache and the LRU Cache of the backup cache only after invalidating the data copy from the read cache. The data is copied to the empty location (if available) of the primary cache and a hash entry is created in the backup cache to indicate its presence in the LRU backup cache. An acknowledgement is sent to the host for the write request only after having cached it
successfully in the primary and backup NVRAM buffer. In case of a full primary cache, Least-Recently-Used (LRU) algorithm is used to make space for the new entry and that replaced data is also invalidated from the backup cache. Now in case of a full LRU backup cache, the Segment Buffers are utilized to move one of the data blocks from the LRU cache into it to make room for a new entry. The hash table and disk segment table are accordingly updated for both the transactions. Whenever a segment buffer gets full, all its data is written in a single operation to the backup cache-disk. This makes the segment buffer quickly available for accepting any further writes. It also makes the small NVRAM cache and large cache-disk appear as a large NVRAM backup write cache to the controller. The destaging of data from the primary cache to the data disks occurs whenever the high water-mark is reached for the primary or the amount of data is more than the low water-mark, but there is a idle period on the data-disk (which ever of the two scenarios occur first). During this operation, the data and the old parity from a dirty LRU block in the primary is read, its new parity calculated and is written to the data-disks along with the data, after which that block in the primary and the backup cache is marked clean. During this destaging process of the primary cache, the data in the backup cache is never written to or read from. The fragmented segments in the backup cache-disk are identified, its corresponding primary cache data is written to a segment buffer in RAM from which they are written to a new contiguous segment in the cache-disk and the old segments are invalidated, thus reducing the fragmentation. This is called
“garbage collection”. For fairly large backup cache-disks, this process is seldom invoked by the controller.

During a read request, the read cache and the primary write cache are searched and in case of a cache-hit the data is made available immediately. But in case of a cache-miss, the LRU replacement algorithm is used to free-up space in the read cache and copy the requisite data from the data-disks to the read cache. During a read request, the backup write cache is never involved and so its slower read access times do not affect the read-performance. In case of a power failure or system crash, the reliability and redundancy is provided by the NVRAM or cache-disk of the backup cache. In case of a data loss on one of the caches, it can be rebuilt by using the data in the other cache as both have consistent copies of data. Also if the primary cache is made up of NVRAM, it provides additional protection (but increases the overall cost of the hierarchy). The study shows that RAPID cache generally has higher reliability and similar performance with lesser costs than a single-copy NVRAM cache or a dual-copy NVRAM cache.

As mentioned earlier, this research work is an extension of the work done in [16] and [17]. The caching algorithms implemented in this work are not exactly the same as the ones in the above literature, instead have been modified to suite the hardware being used and the existent system.
4.3 Modified-DCD cache and Modified-RAPID cache algorithms

The general essence of the caching algorithms mentioned in [16] and [17] is prevalent even in the Modified-DCD (mDCD) and Modified-RAPID (mRAPID) cache implementations. The major changes in the above algorithms are in their individual cache hierarchies and are as follows:

- Instead of a two-stage primary cache as in DCD implementation, we have a two-stage primary cache and a single stage backup (mirrored) cache in our mDCD implementation.
- Also instead of a single stage primary and a two-stage backup cache in RAPID implementation, we have a two-stage primary and a two-stage backup (mirrored) cache in our mRAPID implementation.
- In both the implementations, the log-disk is a logical partition on the system-disks at the backend and is striped across them.
- For mDCD implementation, the destaging of cache data from the RAM buffer to the (logical) log-disk and then from the log-disk to the data-disks depending upon the idle time slots for the data-disk or when the high watermark level is reached, which ever occurs earlier.
- But for the mRAPID implementation, the destaging is carried out by a dedicated destaging-thread process which will destage the data from log-
disk to data-disk at regular time intervals or at the idle time slots for the data-disk rather than giving precedence to the watermark levels.

- The read cache operation and the prefetching are somewhat different from the original algorithms, in a way that it is application agnostic i.e. it does not know what the host is doing.

Figure 29 shows the mDCD cache hierarchy and figure 30 shows the mRAPID cache hierarchy. This section will explain the general working of the cache being implemented and will be more or less the same for both the algorithms mentioned above, any specific differences will also be duly pointed out in the scope of this section.

![mDCD cache architecture](image-url)
In a typical SAN environment, we find two storage processors (SPs) for better reliability, failover, high availability, higher redundancy and load balancing. Each of these SPs have essentially the same hardware and software specifications so that any of them can take over the SANs functioning in case of the other one failing. So in such an arrangement the backup cache cannot be kept in the same SP or else that wouldn’t serve the purpose of redundancy. Hence the backup cache, which is mirrored, is located in the peer SP (this is true for both SPs). So basically each SP contains a write cache, a read cache and the mirrored backup cache for the peer SP. The operating system which oversees the functioning of the SAN on the whole resides on some of the data disks on the storage backend along with other disks that store the user data. These especially reserved disks are called “system disks” and usually are implemented in a RAID-5 type technology i.e. 4+1 RAID, with a hot-spare disk. As an alternative approach to the above
mentioned of the log-disk, the log-disk was attempted to be placed on the hot-spare disk of the RAID-5 technology used for the system-disks, but the approach wasn’t successfully implemented due to some technical difficulties. These system disks and the SPs usually have a battery backup in case of an emergency situation, which allows the SPs and the disks to be functional for a limited period of time so that the system can perform the state-backup operations and collect any essential logs for future trouble shooting before the system shuts down. In the event of a SP panic (wherein the SP detects an operational fault and has to take extreme measures like reboot or powering off), the peer SP detects this and one of the SPs starts a cache-dump of its write cache onto the system disks as both the SPs essentially have the same data.

The write cache which has been implemented here is a truly mirrored write cache wherein the peer SP is an active participant in the mirroring process. The general working of the write cache remains similar to both mDCD and mRAPID. During the allocation of the write (or the read) cache, the user specifies the total amount of the memory to be used for each. This allocated memory is then shared equally amongst the two SPs. The write cache is then fragmented into multiple write cache pages which are also shared equally amongst the SPs. Each of these pages comprises of three parts; the page header, the data block and the meta-data per data block. The header is 88 bytes long and contains page number LUN and LBA number, dirty bits (which indicates that the data in the cache sector has been written to but hasn’t been updated in the data-
disks), valid bits (which indicates that the data in the cache sector is replaceable during the block replacement for the new request, as it has been updated in the data-disks), various pointers which puts the page on various lists. The data block is of variable size and depends upon the total page size specified by the user (2, 4, 8 or 12KB page sizes) and is divided into 512 bytes/sector. The meta-data (8 bytes/sector) contains checksum information, timestamp information and other information related to the RAID type in use (like shed stamp). The cache page headers are stored separately from the data blocks that they describe. The write cache table manages the pages that are owned by that SP or some un-owned but live cache pages like the broken units. The peer write cache table passively manages the pages owned by the peer SP. Write cache data table manages the memory which is being used as the building block of the write cache. The bit-map contains bits which represent each page as clean or dirty. It is used at SP start up to determine which cache pages contain live data after a system-disk dump and SP reboot.

When a write request from a host arrives to at the SP, it is directly sent to the write cache (if enabled). The write cache then allocates an unused cache page from its list of free pages. A scatter-gather (SG) list is generated and returned to the front-end of the SP so the miniport (Fibre Channel or iSCSI) can program the hardware and start the SCSI data phase. Once the data has been stored in the cache pages, the checksums are generated and then the page is added to the actual cache. During the insertion of the write cache pages, the write cache is searched to see if the LUN and LBA range of this
request is already present in the write cache. If the pages are not already present in the write cache, the new pages are inserted into the write cache hash table and then mirrored to the peer SP. If the pages are already present in the write cache for this LUN and LBA range, then those pages which overlap this request are either combined with the new pages (partial overlap) or replaced by the new pages (because the data in the cache is old and so needs to be refreshed by this new one). Once all of the pages have been inserted and any combining done, the cache pages for this request need to be mirrored to the peer SP. If the request is aborted at anytime up to this point, the write cache can simply throw away the pages associated with the request without overwriting any data currently in the write cache (this is called an atomic write). The mirroring process takes place in two steps, first the host data is mirrored and then the cache page meta-data is mirrored. The host data part is mirrored directly to the peer’s mirror of the same cache page. The cache page header is also mirrored but it is mirrored to a ring buffer in the communications manager interface (PCI-E). The PCI-E handles communications between the peer SPs and guarantees in-order delivery of the data. The initiating SP can’t send an acknowledgement to the host for its write request until it has received an acknowledgement from the PCI-E interface that the mirrored data has been successfully delivered to the peer SP i.e. the host data has been archived in the peer’s cache pages and the page header is in the PCI-E ring buffer of the peer. But this doesn’t mean that the initiating SP has to wait for the peer SP to actually process the data before
sending an acknowledgement. So as soon as the data has reached the peer PCI-E, it's enough for the initiator to return the acknowledgement immediately. This guarantees that if the initiating SP panics after acknowledging the host, the peer SP has all of the host data. When the peer SP processes the cache mirror message, the peer SP will go through the same processing that the initiating SP originally did as part of the page insert. As a sanity check, the peer SP also checks to make sure that it inserts the cache page in the same place as the initiating SP. If they don't agree, the peer SP panics with an invalid peer set panic.

The read cache algorithm will scan through the existing pages in the read cache to compare it with the incoming request and determines whether the request is sequential or not. It does so with the help of the current request’s LBA. The current LBA minus the cache page size is searched for in the cache to see if the data for the sectors before this request are already present in the cache. If the cache page containing this LBA is present in the cache, then the request is sequential, and only then a prefetch is generated for the request otherwise the algorithm aborts the prefetching operation and only fetches the requisite data page. There are obvious merits and demerits of this approach. The algorithm does not consume any extra amount of memory for logging the history of the previous requests i.e. it doesn’t try to predict the request pattern based on heuristics. This also allows tracking of multiple independent request streams to a single LUN (multiple user accessing the same data base at the same time), as this method of read-prefetching
relies on spatial detection of request sequentiality and not on the temporal detection mechanism. The demerit to this approach is that in case of a large read cache size or for lesser number of storage LUNs being present on the backend, the large random read workloads could be misidentified as sequential and could initiate unnecessary prefetches. But in case of requests which are less then the size of a single cache page, the adjacent sectors of the cache are searched to determine sequential access instead of searching for the previous page. When the current read request uses a prefetched page i.e. there is a prefetch-hit, the prefetch algorithm now searches the cache with current request’s transfer length instead of scanning one cache page at a time. This reduces the number of cache page searches for large sequential reads. There are two types of read prefetching implemented here. The fixed-size prefetch will prefetch data in a pre-determined fixed quantity. The size of constant-prefetched data is determined by two variables, “prefetch size”, which specifies the size of prefetch in terms of number of sectors to be prefetched and “segment size”, which specifies the size of each block to be prefetched from within those sectors. The prefetch size is limited to a maximum of 2MB. Variable prefetch, as the name suggests, prefetches data in multiples of the current host request length. Its data size is also determined by two variables, “segment multiplier”, which determines the size of the prefetch segment and “prefetch multiplier”, which determines how many number of times of the host transfer the data should be prefetched. So if the prefetch and segment multipliers are of same value, then a single prefetch of the size of the prefetch
multiplier is issued. If the segment multiplier is greater than prefetch multiplier, then the prefetch multiplier is used as the prefetch length. If the segment multiplier is less than the prefetch multiplier, then the total prefetch length is broken up into segment multiplier length pieces. If the total prefetch length is larger than a 128 element scatter-gather list can describe using the current page size, the prefetch is broken up into smaller blocks. If a prefetch is broken up into multiple prefetches, then these are issued in parallel to the backend to take advantage of the drive prefetch mechanisms and the heads being on track already.
Chapter 5

Experiments and Results

5.1 Experimental Setup

Figure 31: Experimental setup.
As been discussed earlier, the experimental setup (figure 31) consists of a host (Windows/Linux) connected to a primary SAN array via either FC or TCP/IP network. The primary array is connected to the secondary SAN array via FC or iSCSI communication channels. Commercially available storage arrays were used for these experiments, which were enabled with commercially available synchronous remote mirroring software. The host was a Dell Server PE 1850 containing an Intel Xeon processor CPU operating at 3GHz, with 2GB RAM with physical address extension. The host also had a LSI Logic 1020/1030 Ultra320 SCSI Adapter, Intel 82801EB LPC Interface Controller, Intel PRO 1000/MT network adapter card. The host could be attached to the primary storage array either through a FC link or through iSCSI link. The primary array runs a synchronous remote mirroring application in the background and mirrors all the data which is being sent to it by the host to the secondary as well. Each storage processor contains a 1.8 GHz, 1GB DDR DIMM (double data rate, dual in-line memory module) system memory (with 1 MB L2-cache) Intel Xenon processor. The two SPs are interconnected with a PCI Express-x4 channel which is used for communication, message exchange and to maintain the mirrored peer-write cache in each of the SPs. Each SP contains QLogic QLA2460 (4Gb, PCI-X 2.0) FC HBA cards. Each of the storage arrays contained 15 data disks, each of 146 GB (4Gb/s bus speed) SATA II (Serial Advanced Technology Attachment) drives with following specifications:
The SAN operating system or the system-disks are a part of these 15 data disks. Hence the total storage on the array is (135GB*15) 2 TB, but user-available storage is around 1.5 TB, SAN operating system and other utilities take up the rest of the available storage. There are 2 FC front-end IO ports which can operate with speeds of up to 4GB/s, 2 iSCSI 1GbE front-end ports and 1 FC back-end disk port which can operate at 2GB or 4GB/s speeds. The SP is connected to the disks through FC-AL (Fibre Channel Arbitrated Loop) links which are controlled by Link Control Cards (LCC).

RAID-5 disk configuration is used for better performance and redundancy at disk-level. Logical units (LUNs) are created on the primary array and are assigned to the host. As a security feature, a host can access only those LUNs which have been assigned to it or are in its user-group. For the experiments, we had created 25 LUNs, each of 50GB size on the primary array, which are the primary mirror images. The mirroring application requires us to create a LUN of equal size of that of the primary image for the secondary image on the secondary array. So 25 LUNs, each of 50GB were also created on secondary array, which were the secondary mirror images. When the mirroring session is active i.e. the host is running the workload session, the secondary image is

---

**Table 2: Data disk specifications.**

<table>
<thead>
<tr>
<th>Formatted capacity</th>
<th>Rotational speed</th>
<th>Data buffer</th>
<th>Average seek time</th>
<th>Rotational Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>135GB</td>
<td>10,000rpm</td>
<td>32MB</td>
<td>4.7ms (read)</td>
<td>3.0ms</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.4ms (write)</td>
<td></td>
</tr>
</tbody>
</table>
considered to be in a “consistent” state, which means the two images do not contain the same data and that the secondary image is being constantly synchronized with the primary. When the host’s workload session ends, both the image have the same data on them and the secondary is considered to be in a “synchronized” state. If due to some reason the mirror-session gets disrupted (like SP reboot, power failure, etc), the secondary goes into a “fractured” state. Once the connection is repaired and reinstated, the images transition into “queued to be synchronized” and then to “synchronized” state.

A NIST Net network emulator box was deployed between the primary and the secondary array to emulate the delay introduced by a large network. The box was a Dell Server PE 1850 containing an Intel Xeon processor CPU operating at 3GHz, with 1GB RAM. The box contained two Intel GbE server cards & Integrated 3Com 3C940 on Asus P4P800 motherboard. We introduced a delay equivalent of a 500 miles distance between the two arrays for getting some sense of remote mirroring. NIST Net is an open-source PC-based router tool to emulate numerous complex network performance scenarios, and is basically a kernel module-extension to LINUX OS, with an X-window based user interface. We have used IOMeter (open source benchmark) benchmark for our experiments, which was run on the host and it sent I/Os to the primary array which had remote mirroring enabled on it. The workload under test had four parallel worker threads writing 8KB I/Os to the 25 target LUNs concurrently. The I/O operations were 60% random and 40% sequential in nature; and comprised of 60% reads and 40% writes. The
reads dominate here as a typical office application contains more reads than writes, with being predominantly random in its access. Each of the test and sub-test case was carried out for the duration of 65 minutes to provide enough time for the initial transients to average out.

5.2 Data Collection and Analysis

The test cases executed for the performance testing of the caching algorithms for iSCSI v/s FC can be briefly classified into five types:

- Type 1: Finding the best value of the high and low watermark levels and the cache sizes for the write and read caches. This was the first task because; the cache performance is most affected by the watermark (wm) values and the actual cache sizes. The cache size because that determines as to how much data can actually be cached at the maximum by the system; the watermark levels because that determines how often the destaging phase would be executed which in turn determines how often the cache would be flushed to the data-disks due to a full cache, which is a bad arrangement. During cache flushing, the cache stops accepting any write requests which is obviously an unfavorable event. These two values are especially important for the performance of the write cache.
• Type 2: Finding the best value of the cache page size for the ideal configuration values of the watermarks and cache sizes.

• Type 3: Finding the best cache sizes when no read prefetching is enabled for the ideal watermark levels.

• Type 4: Finding the best prefetch multiplier and segment multiplier values when variable prefetching is enabled for the ideal configuration values of the watermarks and cache sizes.

• Type 5: Finding the best prefetch size and segment size values when constant prefetching is enabled for the ideal configuration values of the watermarks and cache sizes.

5.2.1 mDCD Experiments for iSCSI

For the mDCD caching algorithm, following experiments were performed:

• Test 1: Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high \( \text{wm}=80 \), low \( \text{wm}=60 \). The plots for average response times,
IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.

Test 2: Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high wm=80, low wm=40. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.

Table 3: Varying cache sizes for 80-60 watermark (mDCD).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>100.875132</td>
<td>123.2203</td>
<td>0.962659</td>
<td>35.862025</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>97.441352</td>
<td>127.903</td>
<td>0.999242</td>
<td>28.26008</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>98.973403</td>
<td>126.9159</td>
<td>0.991531</td>
<td>28.463142</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>96.658976</td>
<td>127.7675</td>
<td>0.998183</td>
<td>28.228645</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>96.060617</td>
<td>128.8195</td>
<td>1.006402</td>
<td>29.318805</td>
</tr>
</tbody>
</table>

Table 4: Varying cache sizes for 80-40 watermark (mDCD).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>99.148211</td>
<td>128.102787</td>
<td>1.000803</td>
<td>29.207041</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>98.522468</td>
<td>127.849998</td>
<td>0.998828</td>
<td>29.274693</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>97.907486</td>
<td>128.258134</td>
<td>1.002017</td>
<td>29.284393</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>97.165011</td>
<td>128.747952</td>
<td>1.005843</td>
<td>29.224683</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>96.121056</td>
<td>130.123759</td>
<td>1.016592</td>
<td>29.155013</td>
</tr>
</tbody>
</table>
Test 3: Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high \(wm=60\), low \(wm=40\). The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.

Table 5: Varying cache sizes for 60-40 watermark (mDCD).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>96.778948</td>
<td>128.339944</td>
<td>1.002656</td>
<td>29.242922</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>98.749485</td>
<td>127.710844</td>
<td>0.997741</td>
<td>29.309986</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>96.237168</td>
<td>128.703334</td>
<td>1.005495</td>
<td>29.307369</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>95.202945</td>
<td>129.388173</td>
<td>1.010845</td>
<td>31.715912</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>95.637871</td>
<td>129.100872</td>
<td>1.008601</td>
<td>29.247417</td>
</tr>
</tbody>
</table>

Figure 32: Average response time v/s cache sizes for various watermark levels mDCD.
Figure 33: IOps v/s cache sizes for various watermark levels mDCD.

Figure 34: MBps v/s cache sizes for various watermark levels mDCD.
From tests 1, 2 and 3, the configuration with the lowest average response time was chosen as the ideal configuration and so its cache sizes and the watermark levels were used in further tests wherein the page sizes and the read prefetch options were varied. In all the above three test cases, the pagesize value was kept at 8KB as that was the I/O size of the benchmark being used. The above mentioned ideal configuration was write cache=73 MB, read cache=218 MB, high wm=60, low wm=40.

- Test 4: Using the ideal configuration values, the cache page size values were varied. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these page sizes.
Table 6: Page size variation for mDCD ideal configuration.

<table>
<thead>
<tr>
<th>Page size (KB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>96.048234</td>
<td>128.1678</td>
<td>1.001311</td>
<td>29.363221</td>
</tr>
<tr>
<td>4</td>
<td>96.356404</td>
<td>127.9628</td>
<td>0.999709</td>
<td>29.258992</td>
</tr>
<tr>
<td>8</td>
<td>95.202945</td>
<td>129.3882</td>
<td>1.010845</td>
<td>31.715912</td>
</tr>
<tr>
<td>12</td>
<td>98.55026</td>
<td>127.1894</td>
<td>0.993667</td>
<td>29.365851</td>
</tr>
</tbody>
</table>

Figure 35: Average response time v/s page sizes mDCD.
Figure 36: IOps v/s page sizes mDCD.

Figure 37: MBps v/s page sizes mDCD.
Test 5: Using the ideal configuration values for the water mark levels, the cache sizes were changed from 0 to 291 MB, with the read prefetching option disabled.

Table 7: Varying cache sizes for no-prefetch (mDCD).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>98.493931</td>
<td>127.8746</td>
<td>0.99902</td>
<td>29.307339</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>99.15105</td>
<td>127.4442</td>
<td>0.995658</td>
<td>29.351182</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>97.495034</td>
<td>129.1963</td>
<td>1.009346</td>
<td>29.243118</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>97.429644</td>
<td>129.2379</td>
<td>1.009671</td>
<td>29.232835</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>98.474786</td>
<td>127.8885</td>
<td>0.999129</td>
<td>29.349847</td>
</tr>
</tbody>
</table>
Figure 9: Average response time v/s cache sizes for no-prefetch mDCD.

Figure 10: IOps v/s cache sizes for no-prefetch mDCD.
**Figure 11:** MBps v/s cache sizes for no-prefetch mDCD.

**Figure 12:** %CPU Utilization v/s cache sizes for no-prefetch mDCD.
• Test 6: Using the ideal configuration values, the prefetch and segment multiplier values were varied simultaneously by enabling the variable prefetch option. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these prefetch and segment multiplier values.

Table 8: Prefetch and segment multiplier variation for mDCD ideal configuration.

<table>
<thead>
<tr>
<th>pmsm</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>95.202945</td>
<td>129.388173</td>
<td>1.010845</td>
<td>31.715912</td>
</tr>
<tr>
<td>8</td>
<td>97.843848</td>
<td>127.643168</td>
<td>0.997212</td>
<td>29.323929</td>
</tr>
<tr>
<td>16</td>
<td>96.716046</td>
<td>128.382264</td>
<td>1.002986</td>
<td>29.276722</td>
</tr>
<tr>
<td>32</td>
<td>97.231704</td>
<td>128.042392</td>
<td>1.000331</td>
<td>29.291446</td>
</tr>
</tbody>
</table>

Figure 13: Average response time for variable prefetch mDCD.
Figure 14: IOps for variable prefetch mDCD.

Figure 15: MBps for variable prefetch mDCD.
Test 7: Using the ideal configuration values, the prefetch and segment size values were varied simultaneously by enabling the constant prefetch option. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these prefetch and segment size values.

Table 9: Prefetch and segment size variation for mDCD ideal configuration.

<table>
<thead>
<tr>
<th>psss</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>98.866519</td>
<td>126.656682</td>
<td>0.989505</td>
<td>29.378371</td>
</tr>
<tr>
<td>1024</td>
<td>96.348855</td>
<td>128.628976</td>
<td>1.004914</td>
<td>29.254276</td>
</tr>
<tr>
<td>2048</td>
<td>97.413866</td>
<td>127.926388</td>
<td>0.999425</td>
<td>29.299906</td>
</tr>
<tr>
<td>4096</td>
<td>96.013234</td>
<td>129.18481</td>
<td>1.009256</td>
<td>29.293765</td>
</tr>
<tr>
<td>8192</td>
<td>97.072288</td>
<td>128.81249</td>
<td>1.006348</td>
<td>29.23428</td>
</tr>
</tbody>
</table>
Figure 17: Average response time for constant prefetch mDCD.

Figure 18: IOps for constant prefetch mDCD.
Figure 19: MBps for constant prefetch mDCD.

Figure 20: %CPU Utilization for constant prefetch mDCD.
5.2.2 mRAPID Experiments for iSCSI

For the mRAPID caching algorithm, following experiments were performed:

- **Test 8:** Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high wm=80, low wm=60. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>93.529297</td>
<td>129.1707</td>
<td>1.009146</td>
<td>29.238564</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>91.6482</td>
<td>129.7492</td>
<td>1.013666</td>
<td>29.294884</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>93.830854</td>
<td>127.657</td>
<td>0.99732</td>
<td>29.304987</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>94.343364</td>
<td>126.6789</td>
<td>0.989679</td>
<td>31.860332</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>93.819294</td>
<td>128.9767</td>
<td>1.007631</td>
<td>29.245287</td>
</tr>
</tbody>
</table>

- **Test 9:** Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high wm=80, low wm=40. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.
Table 11: Varying cache sizes for 80-40 watermark (mRAPID).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>92.984997</td>
<td>128.86834</td>
<td>1.006784</td>
<td>29.427892</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>94.725133</td>
<td>128.37868</td>
<td>1.002958</td>
<td>29.362361</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>93.93991</td>
<td>128.23987</td>
<td>1.001874</td>
<td>29.299382</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>90.707282</td>
<td>128.39241</td>
<td>1.003066</td>
<td>29.326591</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>90.236981</td>
<td>129.36179</td>
<td>1.010639</td>
<td>29.276585</td>
</tr>
</tbody>
</table>

- Test 10: Changing the read and write cache sizes simultaneously for a particular watermark level. The cache sizes were changed from 0 to 291 MB for high wm=60, low wm=40. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these cache sizes.

Table 12: Varying cache sizes for 60-40 watermark (mRAPID).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>91.934299</td>
<td>128.896217</td>
<td>1.007002</td>
<td>29.34382</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>94.6672</td>
<td>127.109205</td>
<td>0.993041</td>
<td>29.374048</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>93.894199</td>
<td>127.611996</td>
<td>0.996969</td>
<td>29.385908</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>94.181533</td>
<td>127.423973</td>
<td>0.9955</td>
<td>29.283752</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>93.163149</td>
<td>128.091976</td>
<td>1.000719</td>
<td>29.268292</td>
</tr>
</tbody>
</table>
Figure 21: Average response time v/s cache sizes for various watermark levels mRAPID.

Figure 22: IOps v/s cache sizes for various watermark levels mRAPID.
Figure 23: MBps v/s cache sizes for various watermark levels mRAPID.

Figure 24: %CPU Utilization v/s cache sizes for various watermark levels mRAPID.
From tests 8, 9 and 10, the configuration with the lowest average response time was chosen as the ideal configuration and so its cache sizes and the watermark levels were used in further tests wherein the page sizes and the read prefetch options were varied. In all the above three test cases, the pagesize value was kept at 8KB as that was the I/O size of the benchmark being used. The above mentioned ideal configuration was write cache=73 MB, read cache=218 MB, high wm=80, low wm=40.

- Test 11: Using the ideal configuration values, the cache page size values were varied. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these page sizes.

<table>
<thead>
<tr>
<th>Page size (KB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>92.808927</td>
<td>128.325239</td>
<td>1.002541</td>
<td>29.313125</td>
</tr>
<tr>
<td>4</td>
<td>91.752855</td>
<td>128.362593</td>
<td>1.002833</td>
<td>29.354642</td>
</tr>
<tr>
<td>8</td>
<td>90.707282</td>
<td>128.392413</td>
<td>1.003066</td>
<td>29.326591</td>
</tr>
<tr>
<td>12</td>
<td>94.004476</td>
<td>127.545925</td>
<td>0.996453</td>
<td>29.280862</td>
</tr>
</tbody>
</table>
Figure 25: Average response time v/s page sizes mRAPID.

Figure 26: IOps v/s page sizes mRAPID.
Figure 27: MBps v/s page sizes mRAPID.

Figure 28: %CPU Utilization v/s page sizes mRAPID.
• Test 12: Using the ideal configuration values for the water mark levels, the cache sizes were changed from 0 to 291 MB, with the read prefetching option disabled.

### Table 14: Varying cache sizes for no-prefetch (mRAPID).

<table>
<thead>
<tr>
<th>Write cache size (MB)</th>
<th>Read cache size (MB)</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>93.569903</td>
<td>129.1449</td>
<td>1.008945</td>
<td>29.377854</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>94.977444</td>
<td>128.2106</td>
<td>1.001645</td>
<td>29.360105</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>93.642159</td>
<td>129.1003</td>
<td>1.008596</td>
<td>29.349017</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>94.688851</td>
<td>128.4042</td>
<td>1.003158</td>
<td>29.309157</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>94.732706</td>
<td>128.3692</td>
<td>1.002884</td>
<td>29.316394</td>
</tr>
</tbody>
</table>

Figure 29: Average response time v/s cache sizes for no-prefetch mRAPID.
Figure 30: IOps v/s cache sizes for no-prefetch mRAPID.

Figure 31: MBps v/s cache sizes for no-prefetch mRAPID.
• Test 13: Using the ideal configuration values, the prefetch and segment multiplier values were varied simultaneously by enabling the variable prefetch option. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these prefetch and segment multiplier values.

Table 15: Prefetch and segment multiplier variation for mRAPID ideal configuration.

<table>
<thead>
<tr>
<th>pmsm</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>90.707282</td>
<td>128.3924</td>
<td>1.003066</td>
<td>29.326591</td>
</tr>
<tr>
<td>8</td>
<td>94.199594</td>
<td>128.0734</td>
<td>1.000573</td>
<td>29.557516</td>
</tr>
<tr>
<td>16</td>
<td>90.270407</td>
<td>128.88</td>
<td>1.006875</td>
<td>29.321472</td>
</tr>
<tr>
<td>32</td>
<td>94.217772</td>
<td>128.0393</td>
<td>1.000307</td>
<td>29.323805</td>
</tr>
</tbody>
</table>
Figure 33: Average response time for variable prefetch mRAPID.

Figure 34: IOps for variable prefetch mRAPID.
Figure 35: MBps for variable prefetch mRAPID.

Figure 36: %CPU Utilization for variable prefetch mRAPID.
• Test 14: Using the ideal configuration values, the prefetch and segment size values were varied simultaneously by enabling the constant prefetch option. The plots for average response times, IO/sec, MB/sec (throughput) and %CPU utilization were plotted for these prefetch and segment size values.

<table>
<thead>
<tr>
<th>psss</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>91.698801</td>
<td>129.06352</td>
<td>1.008309</td>
<td>29.295603</td>
</tr>
<tr>
<td>1024</td>
<td>93.043255</td>
<td>127.51441</td>
<td>0.996206</td>
<td>29.319137</td>
</tr>
<tr>
<td>2048</td>
<td>92.92373</td>
<td>128.25115</td>
<td>1.001962</td>
<td>29.379973</td>
</tr>
<tr>
<td>4096</td>
<td>93.047738</td>
<td>127.51586</td>
<td>0.996218</td>
<td>29.404805</td>
</tr>
<tr>
<td>8192</td>
<td>92.147218</td>
<td>128.75442</td>
<td>1.005894</td>
<td>29.294304</td>
</tr>
</tbody>
</table>

Figure 37: Average response time for constant prefetch mRAPID.
Figure 38: IOps for constant prefetch mRAPID.

Figure 39: MBps for constant prefetch mRAPID.
5.2.3 mDCD v/s mRAPID comparisons for iSCSI

The auxiliary comparisons that can be obtained between mDCD and mRAPID from the above test cases are as follows:

- Comparison 1: The average response times (ART) for the two algorithms for their respective ideal configurations.

  For mDCD; ideal configuration is write cache=73 MB, read cache=218 MB, high wm=60, low wm=40.

  For mRAPID; ideal configuration was write cache=73 MB, read cache=218 MB, high wm=80, low wm=40.
Table 17: Average response time comparisons with varying cache sizes for mDCD and mRAPID ideal configurations.

<table>
<thead>
<tr>
<th>Write cache (MB)</th>
<th>Read cache (MB)</th>
<th>ART_mDCD (ms)</th>
<th>ART_mRAPID (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>96.77895</td>
<td>92.985</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>98.74949</td>
<td>94.72513</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>96.23717</td>
<td>93.93991</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>95.20295</td>
<td>90.70728</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>95.63787</td>
<td>90.23698</td>
</tr>
</tbody>
</table>

avg.resp.time v/s cache sizes for mDCD and mRAPID

Figure 41: Average response time v/s cache sizes for mDCD and mRAPID.

- Comparison 2: The average response times for the two algorithms for various cache page sizes at their respective ideal configuration settings.
Table 18: Average response time comparisons with varying page sizes for mDCD and mRAPID ideal configurations.

<table>
<thead>
<tr>
<th>Page size (KB)</th>
<th>ART_mDCD (ms)</th>
<th>ART_mRAPID (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>96.04823</td>
<td>92.80893</td>
</tr>
<tr>
<td>4</td>
<td>96.3564</td>
<td>91.75286</td>
</tr>
<tr>
<td>8</td>
<td>95.20295</td>
<td>90.70728</td>
</tr>
<tr>
<td>12</td>
<td>98.55026</td>
<td>94.00448</td>
</tr>
</tbody>
</table>

Figure 42: Average response time v/s page sizes for mDCD and mRAPID.

- Comparison 3: The average response times for the two algorithms for disabled read prefetching option for various cache sizes at their respective ideal watermark configuration values.
Table 19: Average response time comparisons with varying cache sizes for mDCD and mRAPID with no-prefetch.

<table>
<thead>
<tr>
<th>Write cache (MB)</th>
<th>Read cache (MB)</th>
<th>ART_mDCD (ms)</th>
<th>ART_mRAPID (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>0</td>
<td>98.49393</td>
<td>93.5699</td>
</tr>
<tr>
<td>218</td>
<td>73</td>
<td>99.15105</td>
<td>94.97744</td>
</tr>
<tr>
<td>146</td>
<td>145</td>
<td>97.49503</td>
<td>93.64216</td>
</tr>
<tr>
<td>73</td>
<td>218</td>
<td>97.42964</td>
<td>94.68885</td>
</tr>
<tr>
<td>0</td>
<td>291</td>
<td>98.47479</td>
<td>94.73271</td>
</tr>
</tbody>
</table>

Figure 43: Average response time v/s cache sizes for no-prefetch for mDCD and mRAPID.

- Comparison 4: The average response times for the two algorithms for the prefetch and segment multiplier values when variable read prefetch was enabled and at their respective ideal configuration values.
Table 20: Average response time comparisons with varying prefetch and segment multiplier values for mDCD and mRAPID ideal configurations.

<table>
<thead>
<tr>
<th>pmsm</th>
<th>ART_mDCD(ms)</th>
<th>ART_mRAPID(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>95.20295</td>
<td>90.70728</td>
</tr>
<tr>
<td>8</td>
<td>97.84385</td>
<td>94.19959</td>
</tr>
<tr>
<td>16</td>
<td>96.71605</td>
<td>90.27041</td>
</tr>
<tr>
<td>32</td>
<td>97.2317</td>
<td>94.21777</td>
</tr>
</tbody>
</table>

Comparison 5: The average response times for the two algorithms for the prefetch and segment size values when constant read prefetch was enabled and at their respective ideal configuration values.

Figure 44: Average response time for variable prefetch for mDCD and mRAPID.
Table 21: Average response time comparisons with varying prefetch and segment size values for mDCD and mRAPID ideal configurations.

<table>
<thead>
<tr>
<th>psss</th>
<th>ART_mDCD(ms)</th>
<th>ART_mRAPID(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>512</td>
<td>98.86652</td>
<td>91.6898</td>
</tr>
<tr>
<td>1024</td>
<td>96.34886</td>
<td>93.04326</td>
</tr>
<tr>
<td>2048</td>
<td>97.41387</td>
<td>92.92373</td>
</tr>
<tr>
<td>4096</td>
<td>96.01323</td>
<td>93.04774</td>
</tr>
<tr>
<td>8192</td>
<td>97.07229</td>
<td>92.14722</td>
</tr>
</tbody>
</table>

Figure 45: Average response time for constant prefetch for mDCD and mRAPID.

5.2.4 FC v/s iSCSI for mDCD and mRAPID

The system cache parameters from the the best case scenarios for each of the previously mentioned five basic test case types for the iSCSI tests were now replicated with FC being the transport protocol instead of iSCSI. The tests were performed for both the caching algorithms. The results of these comparisons are as follows:
• Case 1: No cache in use.

Table 22: FC v/s iSCSI no cache.

<table>
<thead>
<tr>
<th>Type</th>
<th>Average Response Time (ms)</th>
<th>IOps</th>
<th>MBps</th>
<th>% CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>102.2387</td>
<td>1145.098</td>
<td>9.54365</td>
<td>1.356747</td>
</tr>
<tr>
<td>iscsi</td>
<td>225.221</td>
<td>124.4124</td>
<td>0.831972</td>
<td>33.38602</td>
</tr>
</tbody>
</table>

• Case 2: mDCD ideal configuration v/s mRAPID ideal configuration from iSCSI test cases.

Table 23: FC v/s iSCSI mDCD ideal configuration.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mDCD (ms)</th>
<th>IOps_mDCD</th>
<th>MBps_mDCD</th>
<th>Utilization_mDCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>70.43321</td>
<td>1232.85</td>
<td>9.77453</td>
<td>1.30877</td>
</tr>
<tr>
<td>iscsi</td>
<td>95.202945</td>
<td>129.3882</td>
<td>1.010845</td>
<td>31.715912</td>
</tr>
</tbody>
</table>

Table 24: FC v/s iSCSI mRAPID ideal configuration.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mRAPID (ms)</th>
<th>IOps_mRAPID</th>
<th>MBps_mRAPID</th>
<th>Utilization_mRAPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>64.90553</td>
<td>1275.976</td>
<td>9.69881</td>
<td>1.289776</td>
</tr>
<tr>
<td>iscsi</td>
<td>90.707282</td>
<td>128.3924</td>
<td>1.003066</td>
<td>29.326591</td>
</tr>
</tbody>
</table>
• Case 3: From iSCSI test cases, it was observed that the best value of the page size for the respective ideal configurations was 8KB which was the default size during case 2 and so case 3 would have same results as case 2. This case has been mentioned here to maintain the continuity of the test cases being compared from the iSCSI testing rounds.

• Case 4: mDCD v/s mRAPID best cache sizes for no prefetching from iSCSI test cases. Here for mDCD: wc=73, rc=218; mRAPID: wc=146, rc=145.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mDCD (ms)</th>
<th>IOps_mDCD</th>
<th>MBps_mDCD</th>
<th>Utilization_mDCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>72.7612</td>
<td>1178.019</td>
<td>9.70089</td>
<td>1.28989</td>
</tr>
<tr>
<td>iscsi</td>
<td>97.429644</td>
<td>129.2379</td>
<td>1.009671</td>
<td>29.232835</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mRAPID (ms)</th>
<th>IOps_mRAPID</th>
<th>MBps_mRAPID</th>
<th>Utilization_mRAPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>65.88121</td>
<td>1288.113</td>
<td>9.681128</td>
<td>1.28768</td>
</tr>
<tr>
<td>iscsi</td>
<td>93.642159</td>
<td>129.1003</td>
<td>1.008596</td>
<td>29.349017</td>
</tr>
</tbody>
</table>

• Case 5: mDCD v/s mRAPID best pmsm values for variable prefetching from iSCSI test cases. Here for mDCD: pmsm=4; mRAPID: pmsm=16.
Table 27: FC v/s iSCSI variable prefetch mDCD.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mDCD (ms)</th>
<th>IOps_mDCD</th>
<th>MBps_mDCD</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>70.43321</td>
<td>1232.85</td>
<td>9.77453</td>
<td>1.30877</td>
</tr>
<tr>
<td>iscsi</td>
<td>95.202945</td>
<td>129.3882</td>
<td>1.010845</td>
<td>31.715912</td>
</tr>
</tbody>
</table>

Table 28: FC v/s iSCSI variable prefetch mRAPID.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mRAPID (ms)</th>
<th>IOps_mRAPID</th>
<th>MBps_mRAPID</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>64.19871</td>
<td>1335.11</td>
<td>9.7129</td>
<td>1.27771</td>
</tr>
<tr>
<td>iscsi</td>
<td>90.270407</td>
<td>128.88</td>
<td>1.006875</td>
<td>29.321472</td>
</tr>
</tbody>
</table>

- Case 6: mDCD v/s mRAPID best psss values for constant prefetching from iSCSI test cases. Here for mDCD: psss=4096; mRAPID: psss=512.

Table 29: FC v/s iSCSI constant prefetch mDCD.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mDCD (ms)</th>
<th>IOps_mDCD</th>
<th>MBps_mDCD</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>72.9871</td>
<td>1300.654</td>
<td>9.6998</td>
<td>1.29191</td>
</tr>
<tr>
<td>iscsi</td>
<td>96.013234</td>
<td>129.1848</td>
<td>1.009256</td>
<td>29.293765</td>
</tr>
</tbody>
</table>

Table 30: FC v/s iSCSI constant prefetch mRAPID.

<table>
<thead>
<tr>
<th>Type</th>
<th>ART_mRAPID (ms)</th>
<th>IOps_mRAPID</th>
<th>MBps_mRAPID</th>
<th>% CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>fc</td>
<td>66.0119</td>
<td>1398.121</td>
<td>9.73113</td>
<td>1.278192</td>
</tr>
<tr>
<td>iscsi</td>
<td>91.689801</td>
<td>129.0635</td>
<td>1.008309</td>
<td>29.295603</td>
</tr>
</tbody>
</table>
5.3 Results and Inferences

5.3.1 For mDCD

For tests 1, 2 and 3 the cache configuration where one of the two caches is absent really doesn’t mean much as we can’t have one cache non-existent and expect the algorithm to be a generic one. So for the rest of the configurations we find that when the high watermark is set too high, the cache fills up rather quickly without getting destaged often enough as a result of which cache-flushing occurs which disrupts the cache operation, thus causing higher average response times for the larger values of high watermark. That’s the same reason why the IOps and MBps numbers are higher for the lower values of high watermark. The % CPU Utilization number is also higher here in the above case because now the array SP has higher activity due to the destaging taking place rather frequently along with more number of I/O being processed per second. So basically they are opposite to the trend for the average response time. For test 4 we see that for page size values of 2, 4 and 12KB, the average response time is higher than that for 8KB. The I/O size being used is also 8KB and so according to the sequentiality detection algorithm, the 2 and 4KB page sizes cannot encompass the whole of the previous request and the 12KB page size being larger than the request is not termed to contain sequential data even though it can potentially contain the whole previous request
leading to lesser read prefetching when it comes to 2, 4 and 12KB page sizes. The IOps, MBps and % CPU utilization values will have an opposite trend to the average response time values yet again. For test 5 we find that the optimum average response time occurs for the ideal cache configuration itself and follows a distribution similar to test 3, but has higher overall values. The reason for this being that the workload is predominantly read intensive and so no read prefetch leads to more read-cache misses and so higher overall response times. The IOps and MBps values have opposite trends to average response times but are again lower than the corresponding overall values for test 3. For test 6 the average response time is lowest for value of 4 for both prefetch and segment multiplier (pmsm) values than for the other values. The reason could be because with larger data being prefetched for higher pmsm values, the predominantly random workload reaches a saturation point with respect to the prefetch read hits after 32KB of total prefetched data. The IOps, MBps and % CPU Utilization values follow the opposite trend as expected. For test 7 the average response time is lowest for value of 4096 sectors which is equal to the maximum allowable read memory prefetch of 2MB; with IOps, MBps and % CPU Utilization following the opposite trend. But the overall values are higher than that for the variable prefetch option, making variable prefetch a more preferred method of read prefetching.

From above test cases, the over all best values of the cache parameters for the mDCD cache on our test rig are: high watermark = 60, low watermark = 40, write cache
size = 73MB, read cache size = 218MB, page size = 8KB, prefetch multiplier-segment multiplier values for variable prefetch = 4, prefetch size-segment size values for constant prefetch = 4096.

5.3.2 For mRAPID

For tests 8, 9 and 10 the cache configuration where one of the two caches is absent really doesn’t mean much as we cant have one cache non-existent and expect the algorithm to be a generic one. As destaging occurs periodically in this implementation, the watermark levels don’t play as crucial a role as compared to mDCD test cases. Hence we find that for a larger watermark difference (high wm=80, low wm=40), the response time is lowest. IOps and MBps curves follow the opposite trends as expected. The results for test 11 follow the similar trends as for test 4 for mDCD and has the same logic to it as well. For test 12 minimum average response time is obtained for an equally distributed cache size for the read and write cache. For test 13 minimum average response time is obtained for value of 16 for both prefetch and segment multiplier (pmsm) values than for the other values. For test 14 average response time is lowest for 512 sectors of prefetched data. But again as in the case of mDCD, variable prefetch is the preferred mode of read prefetching. From above test cases, the over all best values of the cache parameters for the mRAPID cache on our test rig are: high watermark = 80, low watermark = 40, write cache size = 73MB, read cache size = 218MB, page size =
8KB, prefetch multiplier-segment multiplier values for variable prefetch = 16, prefetch size-segment size values for constant prefetch = 512.

5.3.3 For mDCD v/s mRAPID over iSCSI

From comparisons 1 through 5, we find that mRAPID has lower average response times than mDCD implementation. The reason behind this could be the physical architecture of the respective implementations itself. For the two-stage backup cache in mRAPID, as soon as the data goes into the smaller, faster NVRAM, the ACK is sent, as compared to the monolithic, larger, slower DRAM for mDCD. Also the individual destaging algorithms implemented in mRAPID is better than mDCD, as mRAPID has a dedicated destaging utility and is not totally dependent on idle-slot availability of the data-disk or the high watermark being exceeded. This overall performance difference is not too significant though.

5.3.4 For FC v/s iSCSI comparisons of the two algorithms

Average response time is the main result parameter of interest for us here when it comes to remote mirroring. The average response time that the tool provides is the average time between initiation and completion of an IO operation, averaged over the length of the test run. The total IOps is also an average number of IO operations per
second, averaged over the length of the test run. The total MBps is the average number of megabytes read and written per second, averaged over the length of the test run.

Hence the average response time which the tool reports is actually the average of all the individual response time for each IO transaction which is executed during the test run.

So for example, if the IOps=129 and test-run time=65*60=3900 seconds; hence total number of IO operations that are carried out over the period of this test run are 129*3900=503100. So our sample space for the average response time is quiet large (i.e. > 500000). The tool however doesn’t provide the individual response times for each IO transaction from the test run. So with such a large sample value, the confidence level of the statistical data provided by the tool is also very high. We find that for both the implementations, iSCSI performs up to 70% of that of the FC for the average response time parameter.

- For the ideal cache configuration, mDCD-iSCSI performs up to 73.98% of that of mDCD-FC.
- For no read-prefetch, mDCD-iSCSI performs up to 74.68% of that of mDCD-FC.
- For best variable prefetch values, mDCD-iSCSI performs up to 73.98% of that of mDCD-FC.
- For best constant prefetch values, mDCD-iSCSI performs up to 76% of that of mDCD-FC.
• For the ideal cache configuration, mRAPID-iSCSI performs up to 71.56% of that of mRAPID-FC.
• For no read-prefetch, mRAPID-iSCSI performs up to 70.35% of that of mRAPID-FC.
• For best variable prefetch values, mRAPID-iSCSI performs up to 71.1% of that of mRAPID-FC.
• For best constant prefetch values, mRAPID-iSCSI performs up to 72% of that of mRAPID-FC.

Figure 46: FC v/s iSCSI I/O Mirror Average Response Time for mDCD.
Thus from above results we can see that mRAPID performs better than mDCD for their iSCSI implementations, but when we compare their FC v/s iSCSI implementations, mDCD-iSCSI is able to approximate its mDCD-FC performance better than what mRAPID-iSCSI does against mRAPID-FC i.e. on an average for mDCD, its iSCSI implementation performs up to 74.5% of that of its FC implementation, whereas on an average for mRAPID, its iSCSI implementation performs up to 71.25% of that of its FC implementation. The value in the disk-timeout registry on the windows host is the amount of time for which the host adapters will wait for an I/O request (read or write) before timing out and failing the request. The default value for this register is 10 seconds for Windows. Hence, from all the above results we can see that all the average response
time values for any type of the test run is well within the 10sec. value and so both the algorithms are performing the mirroring application without any errors for both iSCSI and FC implementations. So even though the iSCSI mirrors are slower than FC, it is still within the host-allowed delay values. This is proof-enough that we can successfully and economically carry out remote synchronous mirroring using iSCSI for our specified network conditions and experimental setup.
Chapter 6

Conclusions and Future Work

In this thesis we have shown that iSCSI can perform with acceptable performance values for the synchronous remote mirroring application when pitted against its FC counterpart. The results do show that iSCSI is at least 70% as effective a protocol for synchronous remote mirroring as compared to FC when deployed on a commercial storage array. It was also observed that the two caching algorithms which have been implemented here actually play a lesser role to an extent when it comes to the overall response time parameter for the mirroring as compared to the data transit time over the TCP/IP network. The main contributions from this thesis project are as follows:

- Modifying the existing two caching algorithms and extending them to a commercially available SAN environment.
- Evaluating the effects of these modified caches on the performance parameters for a synchronous remote mirroring application deployed on the SAN and thus trying to ascertain as to whether there are any
significant performance gains by changing the storage array's caching algorithm for the above application.

- Evaluating the performance of the underlying iSCSI transport protocol (individually for the two caches) for the above application and comparing it with the traditionally-preferred FC protocol to see as to whether we can actually profitably utilize the lower acquisition cost and lower maintenance costs of iSCSI as compared to FC, without having to excessively compromise on overall application performance.

The performance analysis carried out in this thesis could be further extended by integrating some of these future work areas which are worth exploring, along with a number of other areas as well:

- The iSCSI protocol is highly impaired by the underlying TCP/IP network bandwidth and speed limitations. Even though for the synchronous remote mirroring, average response times of iSCSI are somewhat comparable to its FC counterpart, the IOps and MBps languish way behind. The main reason for this is that FC operates on a 4GB FC network which is fairly exclusive i.e. no general traffic flows through it; whereas the iSCSI operates on a general-use 1GB Ethernet network. So due to the huge resource difference between the two physical media,
iSCSI naturally has to some how compensate for its shortcomings elsewhere. A possible improvement and future work are could be to modify the TCP/IP network stack especially for iSCSI traffic on the iSCSI hosts and iSCSI storage arrays. Use of 10GB Ethernet media could really pave the way for a huge improvement in overall iSCSI performance values.

- Also, better host bus adapters, TCP offload engines and network interface cards could be used on the servers and the storage arrays, which could reduce the TCP/IP processing, required to be done by the storage processors and thus reduce the CPU utilization too. Putting the TCP code and processing into hardware could make iSCSI faster.

- Another issue of future interest could be to explore the iCache algorithm mentioned by authors in [6], which could be specialized for caching the iSCSI specific data, and thus further reduce the response time disparity between iSCSI and FC. A point worth considering here is that enhancement in caches especially for iSCSI data is not as big a factor which could boost iSCSI performance in the future, as compared to enhancing the underlying network resources and protocol implementations.
List of References


[19] http://searchstorage.techtarget.com/tip/0,289483,sid5_gci1203763,00.html


