Characterization of Incremental Data Changes for Efficient Data Protection

Hyong Shim, Philip Shilane, and Windsor Hsu
Backup Recovery Systems Division
EMC Corporation

Abstract

Protecting data on primary storage often requires creating secondary copies by periodically replicating the data to external target systems. We analyze over 100,000 traces from 125 customer block-based primary storage systems to gain a high-level understanding of I/O characteristics and then perform an in-depth analysis of over 500 traces from 13 systems that span at least 24 hours. Our analysis has the twin goals of minimizing overheads on primary systems and improving data replication efficiency. We compare our results with a study a decade ago [20] and provide fresh insights into patterns of incremental changes on primary systems over time.

Primary storage systems often create snapshots as point-in-time copies in order to support host I/O while replicating changed data to target systems. However, creating standard snapshots on a primary storage system incurs overheads in terms of capacity and I/O, and we present a new snapshot technique called a **replication snapshot** that reduces these overheads. Replicated data also requires capacity and I/O on the target system, and we investigate techniques to significantly reduce these overheads. We also find that highly sequential or random I/O patterns have different incremental change characteristics. Where applicable, we present our findings as advice to storage engineers and administrators.

1 Introduction

Protecting data on primary storage systems often requires periodically creating secondary copies by transferring changed data to external target systems, which may be in the same facility or remotely located. However, as the size of data to be protected continues to grow exponentially, the traditional approach to data protection, e.g., copying all the data on the primary storage system to a target system (such as backup servers) at regular intervals, is fast becoming infeasible. A better approach is to only copy the data blocks that have been modified since the last transfer, unlike standard backup software that copies modified files or whole directories. So, understanding how data changes on primary storage over time is key to both improving existing data protection solutions and enabling new solutions.

Specifically, we analyzed the size, rate, and pattern of data changes over time under various host I/O access patterns on EMC Symmetrix VMAX systems [8], a tier-1 block-based primary storage system. We analyzed over 100,000 traces from 125 enterprise systems from some of the world's largest corporations to gain high-level insights into storage characteristics. We then selected over 500 traces that spanned at least 24 hours from 13 systems to analyze various incremental transfer intervals. We believe the number of traces and systems used for analysis is substantially larger than in previously published studies and our results are of value to any organization designing or configuring data protection architectures.

Replicating changed data from a primary system to a target system may take a substantial amount of time, depending on the change rate and transfer throughput. During the transfer period, the primary system must maintain the point-in-time version of storage until the transfer completes, even while hosts write to the primary system. Snapshots [2, 5, 10, 22] are a general purpose mechanism to capture the point-in-time view of data, and transferring snapshots to target storage is one technique for data protection [20]. As two examples, snapshots kept within primary storage allow a user to recover accidentally deleted files, and snapshots are increasingly used to maintain a consistent state of the system to be copied to target storage while a primary system continues operation. We have focused our analysis on snapshot overheads when used for replication.

We found that using standard snapshots for replication incurs significant overhead in terms of space usage and I/O. We observe that only the point-in-time state of the changed blocks (instead of all of the blocks) needs to be maintained, so we can relax the semantics of snapshots, which we call a **replication snapshot**. A replication snapshot protects the changed blocks that need to be replicated without necessarily maintaining the values of blocks that do not need to be copied to target storage. Typical snapshot implementations are designed to create semi-persistent versions, while replication snapshots are designed specifically to support periodic replication and are then released. Also, implementing replication snapshots along with a replication protocol allows separate primary storage and target storage vendors to jointly support efficient replication.

Storage overheads on primary storage can be avoided when host writes are protected with a synchronous remote mirroring mechanism [14], in which host writes are, in effect, sent to both primary and target storage.

1. 8% of capacity needs to be reserved for snapshot overheads to support incremental transfers every 12 hours. The reserve is as low as 2% of capacity with replication snapshots.
2. Primary I/O should be over-provisioned by 100% to support copy-on-write related write-amplification of host writes during replication. The over-provision can be as low as 20% with a replication snapshot.
3. Having a write buffer effectively decreases snapshot I/O overheads but has little impact on storage overheads.
4. The daily transfer size with small blocks is generally 40% of what hosts write.
5. Scheduling at least 6 hours between transfers allows blocks to achieve nearly peak dirtiness.
6. Scheduling at least 12 hours between transfers drastically reduces peak network bandwidth requirements. Volume capacity is not predictive of bandwidth requirements.
7. Target storage must support as much as 20% of the I/O per second capabilities of primary storage when the replication interval is at least one hour.

Table 1: Rules-of-thumb from our analysis

Such a mechanism, however, typically requires that target system have storage capacity and I/O performance similar to those of the primary system, which does not scale well to transferring data changes over a long distance to protect against site disasters. Our analysis focuses on data-protection cases where target systems have larger capacity but potentially lower I/O capabilities than primary systems. This is because data protection systems must be large enough to hold multiple versions of primary storage such as daily copies for a month or longer. As guidance to storage engineers and administrators, we summarize our findings in a set of rules-of-thumb, which are presented in Table 1. Our contributions include: a detailed analysis of data change characteristics for a large set of traces collected from deployed systems, a design for replication snapshots to reduce overheads on primary storage, and an evaluation of overheads on primary and target storage to guide design and configuration.

A related study by Patterson et al. [20] investigated how to efficiently create primary system snapshots at remote systems. The main differences between the present work and Patterson's include our investigation of using various units of data aggregation to transport changed data and their impact on the size of transferred data and I/O rate on the target system. We also investigate how incremental data changes are impacted by different host write I/O patterns used to produce the data change. Importantly, it has been a decade since the earlier study, and it is worth revisiting this analysis to understand how I/O properties have changed using a newer, larger set of traces.

2 Collected Traces

We collected I/O traces from over 100,000 logical volumes from 125 EMC Symmetrix VMAX [8] systems installed at enterprise customer sites. The number of logical volumes captured for each primary storage system ranged from 12 to over 14,000. These systems supported database, email, file system, and other business applications. Unfortunately, no other information is available regarding which applications wrote to and read from which logical volumes. While such information would have been useful, enterprise primary storage systems should be designed to support a wide range of applications.

Traced data includes sector-level read/write I/O requests received by primary storage systems as applications performed I/O operations on their hosts connected to the primary systems in, for example, storage area networks (SANs). Traced data was collected into a trace file per volume. The trace file (or simply trace) contains a number of records, each of which contains the following data fields: timestamp from the beginning of the trace, read/write command, port at which I/O is received, logical volume number, logical sector address (ranging from 0 to largest address), and number of sectors to read or write.

Table 2 and Table 3 summarize I/O activities, rate, and throughput in the traced systems. See the captions of the tables for the descriptions of analyzed I/O properties. Each row of the tables corresponds to a subset of logical volumes that share some common properties and are analyzed together. The trace sets are:

1hr_1Wrt: logical volumes traced for at least 1 hour and that received at least 1 write I/O

1hr_1GBWrt: a subset of **1hr_1Wrt**, which includes volumes traced for at least 1 hour and that received at least 1GB worth of writes

24hr_1GBWrt a subset of **1hr_1GBWrt**, which includes volumes traced for at least 24 hours and that received at least 1GB worth of writes

24hr_1GBWrt_Random: a subset of **24hr_1GBWrt**, which includes volumes that received largely random write I/O requests (See Section 2.1)

24hr_1GBWrt_Sequential: a subset of **24hr_1GBWrt**, which includes volumes that received largely sequential write I/O requests (See Section 2.1)

The **24hr_1GBWrt*** trace sets were selected for detailed analysis because they provide a consistent basis for a wide range of simulations across replication intervals.

As the large standard deviations in the tables indicate, the traced volumes widely vary in host I/O activities they supported. The tables do confirm the long-held view that hosts issue more read I/O requests than write I/O requests

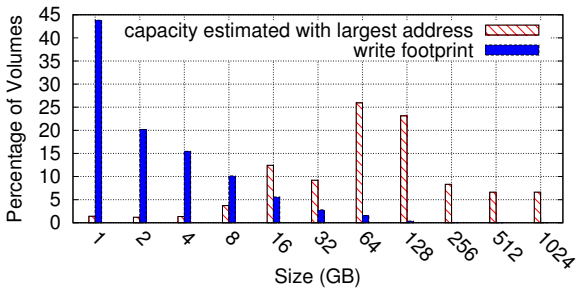


Figure 1: Storage capacity estimates and write footprint for **1hr_1GBWrt**.

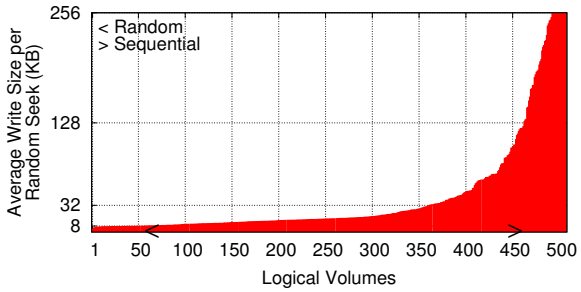


Figure 2: Average write size per random seek. We define random and sequential volumes as having <8.5KB and >102KB average writes per seek, respectively.

and (not surprisingly) read more data than they write. Note in the *W_rlen* column of Table 2 that the average run length of dirty data written after a random seek is much longer for **1hr_1Wrt** and **1hr_1GBWrt** trace sets than for the **24hr_1GBWrt** sets. This is because a small fraction of volumes included in these sets received long bursts of sequential writes that skewed the average values for the entire sets. This can be seen in the standard deviations, which are 10 times the corresponding averages. Such volumes are excluded from the **24hr_1GBWrt** sets as their trace periods are shorter than 24 hours.

We do not have access to the configured volume size, so the storage capacity of each volume is estimated with the largest logical address found in the corresponding trace. Figure 1 shows the estimated storage capacity distribution for the 16,100 volumes in the **1hr_1GBWrt** set. For comparison, we also show the write footprint distribution as percentage of volumes. The write footprint is the number of unique sectors written converted to bytes. Most volumes only had a few gigabytes of unique writes, though volumes were estimated as hundreds of gigabytes in capacity.

2.1 Sequential vs Random I/O

To determine if host I/O pattern has any significant impact on our major findings, we further distinguish traces in the **24hr_1GBWrt** set into **sequential** and **random**. Intuitively, a sequential trace is the result of a host writing data to consecutive locations. From surveying the literature, we have found multiple definitions of sequential I/O (e.g., [1, 4, 12, 17, 21, 23, 24]). For our metric, we measure how much data are written, on average, after seeking to a random sector. By random sector, we mean a sector that is not consecutive with the last sector written based on logical address.

Figure 2 shows the average sequential write size after a random seek for >500 logical volumes in **24hr_1GBWrt**. The volumes are arranged on the x-axis in increasing order of the average sequential write size. Towards the right end of the x-axis, hosts write >102KB of data in sequence after making a random seek in 11% of the volumes. Towards the left end of the x-axis, hosts write <8.5KB of data in sequence after making a random seek in 11% of the volumes. Unfortunately, there is not a clear division between sequential and random host I/O shown in the figure, so for the purpose of our analysis, we use the average sequential write sizes of >102KB and <8.5KB as threshold values in determining sequential volumes and random volumes respectively. The sequential and random volumes are denoted as **24hr_1GBWrt_Sequential** and **24hr_1GBWrt_Random** in Table 2. In the remainder of the paper, the 24hr_1GBWrt trace set is referred to as **All**, 24hr_1GBWrt_Sequential as **Seq** and 24hr_1GBWrt_Random as **Random**.

One drawback of our definition of sequential access is that it does not account for interleaving writes from different hosts because our tracing was lower in the storage system. Nevertheless, we have adopted this approach based on our observation that sequential write I/O requests often appear together in sequence in trace files. Another potential weakness of our trace analysis is that we have not specifically analyzed time-of-day effects. Partly, this is an artifact of our trace collection process that has retained relative, not absolute time stamps, so our replication intervals begin at the start of each trace. Since we have a relatively large number of traces, such effects are likely averaged out, but a future analysis could clarify the impact by comparing results after offsetting the start time.

3 Analysis Methodology

While analyzing logical volume traces, we have tracked incremental changes over time and measured various statistics. Our simulation entails three main components: **replication intervals**, **blocks**, and **transfer throughput**. The top half of Figure 3 illustrates host I/O as a sequence of writes and reads, and the bottom half shows affected sectors and blocks in a logical volume.

A **replication interval** simulates the fact that data protection mechanisms in primary systems often keep track of dirty data for a user-defined period of time and replicates dirty data to target storage at the end of each pe-

Trace Set	#Vol.	#Sys.	Dur. (hrs)	Est.Cap. (GB)	#W_reqs (1000s)	W_size (GB)	W_fp (GB)	W_rlen (KB)	#R_reqs (1000s)	R_size (GB)	R_fp (GB)	R_rlen (KB)
1hr_1Wrt	109263	125	30.4 [78.3]	71 [203]	72.2 [510.4]	1.7 [31.0]	0.7 [11.2]	947.0 [9230.3]	166.5 [1962.8]	5.2 [65.7]	2.2 [18.9]	667.0 [11301.9]
1hr_1GBWrt	16100	120	7.7 [6.7]	132 [262]	429.0 [1270.7]	10.7 [80.1]	4.6 [28.9]	948.5 [9270.8]	796.0 [4986.7]	24.9 [166.3]	9.8 [45.0]	491.2 [11065.7]
24hr_1GBWrt All	508	13	24.4 [1.2]	318 [439]	1802.8 [4838.7]	51.1 [337.6]	19.9 [103.7]	284.6 [256.1]	7824.3 [23875.4]	241.5 [763.2]	91.3 [172.1]	132.7 [3078.8]
24hr_1GBWrt Random	58	9	24.2 [0.8]	238 [328]	1365.5 [1819.7]	9.9 [13.8]	7.2 [12.1]	8.1 [0.4]	5677.1 [8587.6]	97.0 [111.2]	66.5 [84.2]	35.6 [27.3]
24hr_1GBWrt Sequential	54	9	24.9 [1.4]	343 [591]	2542.1 [7567.2]	280.2 [993.9]	102.6 [301.8]	461.4 [193.4]	2292.1 [7533.8]	247.8 [963.5]	64.1 [191.3]	687.9 [9118.1]

Table 2: Summary of I/O activities. The first four columns denote the number of logical volumes in a trace set, the number of primary systems, the average trace period, and the average estimated storage capacity. The rest of the columns show average I/O requests the host has issued. Footprint (*fp*) is the sum of unique sectors written or read at least once, while run length (*rlen*) indicates the average size of data accessed in sequence after a random seek. The values in square brackets are standard deviations for the corresponding averages.

	I/O Request Rate (1000s/sec)						I/O Request Throughput (MB/sec)					
Trace Set	Avg. W_rate	Peak W_rate 1 sec	Peak W_rate 10 ms	Avg R_rate	Peak R_rate 1 sec	Peak R_rate 10 ms	Avg. W_tput	Peak W_tput 1 sec	Peak W_tput 10 ms	Avg. R_tput	Peak R_tput 1 sec	Peak R_tput 10 ms
1hr_1Wrt	0.0007 [0.008]	0.2 [0.6]	1.8 [2.6]	0.002 [0.03]	0.3 [0.8]	1.7 [2.5]	0.02 [0.4]	6.5 [26.0]	64.0 [1669.7]	0.05 [0.8]	10.5 [85.5]	107.3 [8089.5]
1hr_1GBWrt	0.02 [0.04]	0.9 [1.3]	4.4 [4.4]	0.03 [0.1]	0.9 [1.4]	3.6 [4.1]	0.4 [1.8]	30.2 [60.6]	224.1 [4342.7]	0.9 [3.7]	32.8 [216.2]	359.7 [21K]
24hr_1GBWrt All	0.02 [0.06]	1.5 [1.8]	9.0 [8.2]	0.09 [0.3]	2.0 [2.5]	5.6 [7.0]	0.6 [3.9]	44.3 [76.7]	325.0 [460.7]	2.8 [8.8]	122.42 [1188.2]	5644.6 [119K]
24hr_1GBWrt Random	0.02 [0.02]	1.6 [1.4]	6.8 [5.6]	0.07 [0.1]	1.3 [1.0]	4.2 [3.9]	0.1 [0.2]	15.9 [13.5]	143.4 [326.6]	1.1 [1.3]	32.5 [50.0]	166.5 [316.9]
24hr_1GBWrt Sequential	0.03 [0.08]	1.2 [1.7]	5.3 [4.9]	0.03 [0.08]	1.5 [2.0]	4.3 [4.3]	3.2 [11.4]	98.1 [121.1]	584.7 [817.4]	2.8 [11.1]	70.4 [107.1]	517.6 [880.5]

Table 3: Summary of I/O rate and throughput. The peak values for each volume are selected by considering every 10ms and 1 second period. The peak values for a given set are the average of peak values of individual volumes.

riod. In our trace analysis, we model how host write I/O requests are collected for a given replication interval, and one or more dirty sectors are determined from those requests. Reads are ignored. We have used the following replication intervals for analysis in this paper: 24 hours, 12 hours, 6 hours, 3 hours, 1 hour, 30 minutes, and 15 minutes. We have performed some analysis down to 1 minute replication intervals, though to simplify figures, we generally do not show the intervals below 15 minutes. Organizations typically select a replication interval based on their recovery point objective, which defines the time period for which they can tolerate losing data changes due to a disaster. Organizations would like to shrink the replication interval to as short as possible while considering the cost and infrastructure requirements.

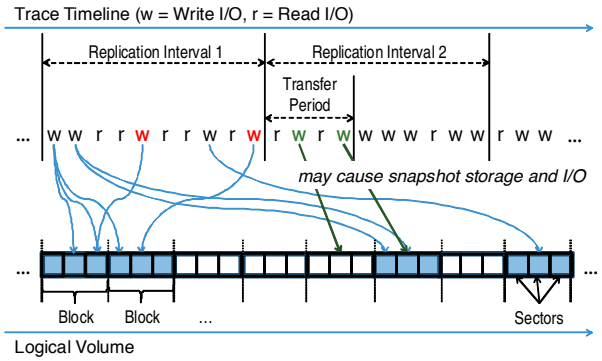


Figure 3: Example of processing a logical volume trace by removing read requests and recording affected sectors and blocks. The red 'w' indicates overwriting requests.

In addition, as shown in Figure 3, dirty sectors are mapped to a larger unit, called a **block** in our model. A block is a sequence of n consecutive sectors in logical volume space, where $n >= 1$. Blocks simulate the fact that many storage systems and data protection mechanisms aggregate dirty sectors into a larger unit and copy those units when replicating modified data to target storage. They do so to reduce memory and storage resources required to maintain, for example, a map of dirty sectors. Block sizes around 128KB are common in some storage systems [26]. For a 1TB volume, the memory requirements for bit vector tracking are: 512B blocks require 256MB of RAM, 128KB blocks require 1MB of RAM, and 1MB blocks require 128KB of RAM.

A block is called 'dirty' if it has one or more dirty sectors, and the figure shows dirty blocks in a darker shade

for Replication Interval 1. When determining the number of dirty sectors (and blocks) during a replication interval, over-writes to a given sector (and block) are counted once. For space and figure clarity reasons, we only show the results for extreme block-size values of 512B and 1MB, unless otherwise noted, because results changed in a gradual manner with block size.

We also consider the impact of **transfer throughput** within our model, which we define very broadly as the throughput from reading dirty blocks on the primary system, transferring across a network (LAN or WAN), and storing on a target system. Based on the transfer throughput and amount of data to transfer, we can determine the transfer period (see Figure 3) during which a primary system must maintain a consistent view of dirty blocks until transfer completes. I/O from the host during the transfer period may cause snapshot I/O (shown in the figure), and those modifications will be recorded and transferred at the end of Replication Interval 2. Note that all modified blocks will be transferred, but because of the point-in-time nature of transferring a snapshot, we have to carefully manage which version of a block exists when a snapshot is created. Managing multiple block versions causes snapshot I/O and storage overheads.

We have analyzed throughputs between 1.5Mb/s (T1) and 40Gb/s and typically discuss results for 1.5Mb/s (T1) and 1Gb/s representing WAN and LAN scenarios, respectively. Note that this throughput is per volume, and storage systems can have over 10,000 volumes. If all 10,000 volumes were replicated at T1 bandwidth individually, this would require 15Gb/s, which is impractical for many customers. Even with that consideration, our analysis provides general results for volumes selected for replication.

With the described trace analysis methodology and storage system model, we can determine how much data should be copied to a target system at the end of each replication interval. To determine the number of write I/O requests needed to copy the data, we assume the underlying data transfer protocol has an upper limit on transfer size, which is assumed to be 1MB, so a larger data run is split.

4 Findings for Primary Storage

At the end of a replication interval, the primary system begins transferring changed blocks to the target, which can take seconds to hours depending on the replication interval, the number of changed blocks, and transfer throughput. During that time, the primary system must maintain an accurate point-in-time representation of those changed blocks, while also supporting incoming host writes that may be directed at blocks that are in the process of being transferred as well as blocks not being transferred. In this section, we characterize storage and I/O overheads for primary storage while changed blocks are transferred to target storage under a variety of configurations.

For a logical volume, snapshots are a general purpose technique to preserve the values for all sectors, usually with a mapping from logical to physical sector addresses [2, 5, 10, 22]. Snapshots are often used to preserve copies on a primary system but are also increasingly being leveraged indirectly for data protection. While there are multiple ways snapshots could be implemented, copy-on-write and redirect-on-write are two prevalent implementations. Suppose snapshot s_t is created at time t. A host write to the volume at time $t+1$ causes the version of the block at time t to be copied into the snapshot (copy-on-write) or the write at $t+1$ is redirected to a snapshot (redirect-on-write). Snapshot s_t has meta data indicating whether the appropriate version of a block is in the main volume or exists in a snapshot region.

Depending on how sectors are modified, both snapshots techniques could be close to empty (no modified sectors) or as large as the active volume (all modified blocks). In terms of I/O, copy-on-write requires I/O to perform the read and write of the earlier block value. Redirect-on-write may require I/O for read-modify-write when a write is less than the block size, and redirect-on-write affects data locality. Creating a clone is an alternative to creating a snapshot, but a clone is less space efficient because it is a full point-in-time copy.

4.1 Replication Snapshot

While this paper focuses on transferring changed blocks, standard snapshot functionality is not designed for this purpose in that any incoming write I/O causes a copy-on-write or redirect-on-write. For replication snapshots, we finely track which blocks need to be transferred for a given replication interval (those that have changed since the last transfer). Only application writes to those blocks cause copy-on-write or redirect-on-write during the time it takes for a transfer to complete. Application writes to non-tracked blocks can happen normally, and all modifications will be transferred in the next replication interval. We present results from the baseline snapshot approach as well as from two versions of replication snapshots, which relax some of the requirements for generic snapshots such that only data that needs to be transferred are tracked.

When describing snapshot techniques, we refer to an example volume shown in Figure 4. Blocks shaded in blue are changed at the end of a replication interval and need to be transferred. Also, their values need to be preserved until replication completes while allowing host I/O to continue.

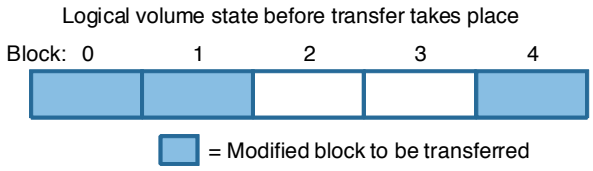


Figure 4: Changed blocks 0, 1, and 4 are transferred to target storage at the end of a replication interval, and a snapshot maintains their state while host I/O continues.

Baseline Snapshot: The standard approach performs snapshot I/O for all incoming host writes regardless of whether the affected block is being transferred or not. In Figure 4, host writes to any block $(0-4)$ result in snapshot I/O the first time. All blocks are released from snapshot protection once the three changed blocks are transferred.

Changed Block Replication Snapshot (CB): Only the changed blocks being transferred at the end of a replication interval are tracked, so host write I/O to these blocks causes snapshot I/O. Importantly, host write I/O to clean blocks is processed without snapshot I/O. In our example, host writes to blocks 0, 1, and 4 cause snapshot I/O, but writes to blocks 2 and 3 do not. All blocks are released from snapshot protection once the three changed blocks are transferred.

Changed Block with Early Release Replication Snapshot (CBER): Similar to the previous version, only changed blocks are tracked, but a block is released from replication snapshot tracking immediately once it is transferred, instead of waiting for the entire transfer to complete. In the figure, host writes to blocks 0, 1, and 4 will cause a snapshot I/O only if those blocks have not yet been transferred based on block-by-block tracking of transfer status.

Note that for all three snapshot versions, repeated host I/O to the same block only causes a single snapshot I/O. Also, the amount of data transferred is identical for all three snapshot techniques. The only difference is the overhead for snapshot I/O and storage. A property affecting snapshot performance is the transfer throughput, which affects how long a snapshot persists. In simulation, we have explored a range of throughputs described in Section 3 but only present a subset of results due to space limitations.

While CBER has lower overheads than CB in our experiments, there is extra tracking information required. There is also more communication with target storage to confirm when individual blocks have been transferred so that blocks can be released from replication snapshot tracking. We leave such analysis to future work. Depending on specific storage system implementations, one type of replication snapshot may be more appropriate than another.

4.2 Storage Overhead

We performed experiments to measure the amount of extra storage space required for blocks written due to snapshots, which is the same for copy-on-write and redirect-on-write. This storage overhead is required to maintain block values while changed blocks are transferred to a target system. Figure 5 shows results for a throughput of 1.5Mb/s for block sizes of 512B and 1MB and three snapshot alternatives for Random 5a, All 5b, and Sequential 5c hosts.

For all configurations, as the replication interval increases on the horizontal axis from 15 minutes to 12 hours, the average fraction of capacity required for snapshots increases. For Figure 5b, we see an average storage overhead of 8% for the Baseline approach with 1MB blocks at 12 hours, and we have even found a peak overhead of 100% in some traces. Unsurprisingly, we see a consistent pattern that the storage overhead is larger for 1MB blocks than 512B blocks. Replication snapshot techniques such as CB and CBER reduce storage overhead because of finer-grained tracking of block transfer state. Considering 1MB blocks at 12 hours, storage overheads decrease from 8% to 4% to 2% respectively, and we see the same trend for 512B blocks.

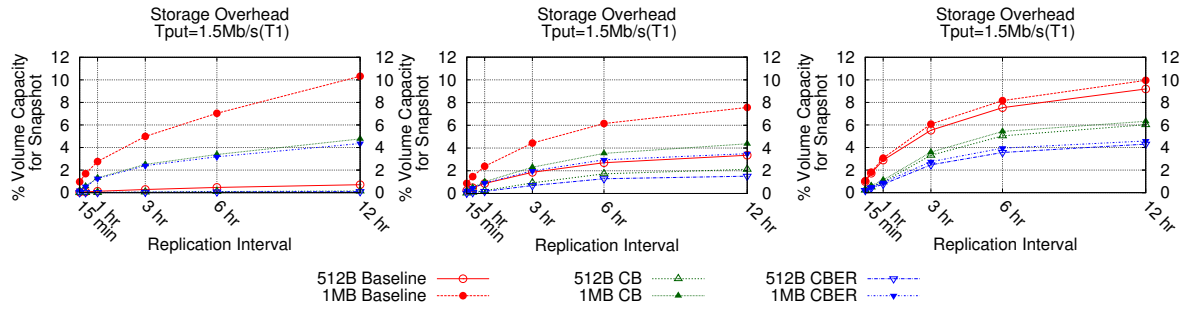
Our general conclusions hold for Random and Sequential traces, though there are several interesting differences. For Random traces, 512B blocks have very low capacity overheads because of the lower change rate for Random traces. For Sequential traces, the block size has little impact because blocks tend to be fully dirty.

Although not shown for space reasons, the trends are identical at a higher throughput of 1Gb/s. Larger blocks require more capacity overheads than smaller blocks, and finer-grained snapshots reduce overhead. Because of the higher throughput, transfer time is shorter (seconds versus minutes or hours), and storage overhead is a few percent on average for every configuration.

Rule-of-thumb 1: 8% of capacity needs to be reserved for snapshot overheads to support incremental transfers every 12 hours. The reserve is as low as 2% of capacity with replication snapshots.

4.3 I/O Overhead

We have further analyzed the I/O overhead for snapshots by measuring the fraction of host write I/O that causes a snapshot I/O during the transfer period. This can be thought of as I/O amplification because a host write can cause a read and second write for copy-on-write. For redirect-on-write, there may be a read-modify-write due to writes smaller than the block size as well as decreased data locality.



a. Random b. All c. Sequential

Figure 5: Snapshot storage overhead due to host write I/O for Random, All, and Sequentially written systems.

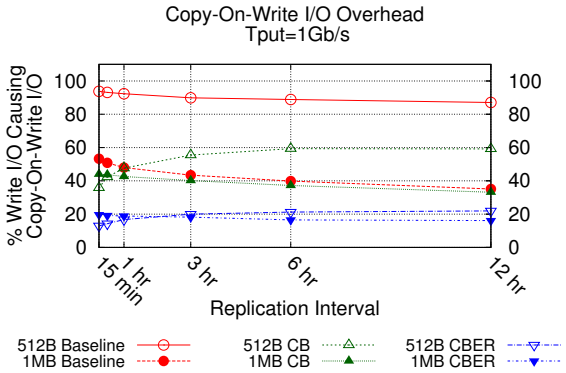


Figure 6: Fraction of host write I/O that causes copy-on-write I/O during the transfer period. The plotted lines are for **24hr_1GBWrt All**.

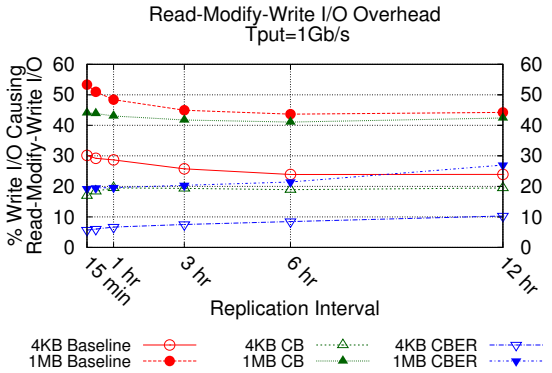


Figure 7: Fraction of host write I/O that causes read-modify-write I/O during the transfer period. The plotted lines are for **24hr_1GBWrt All**.

As shown in Figure 6, copy-on-write I/O can be almost 100% of the host I/O for 512B blocks and Baseline snapshots. In general, we find that smaller blocks cause a larger number of copy-on-write I/Os than larger blocks, though transferring larger blocks will include sectors that were not modified. This is because host write I/O tends to be at least somewhat sequential, and only the first I/O to a block causes a copy-on-write I/O. We also find a consistent pattern, in which improving the replication snapshot technique decreases the copy-on-write I/O overhead across block sizes and replication intervals.

In contrast, redirect-on-write has different patterns than copy-on-write, because redirect-on-write can cause read-modify-write operations as shown in Figure 7. We analyzed 4KB blocks instead of 512B blocks since there is never a read-modify-write for 512B blocks. We find that 1MB blocks have a higher fraction of read-modify-write I/O because host I/O sizes tend to be kilobytes.

These results presented for 1Gb/s throughput are qualitatively similar to results for lower transfer throughputs. One difference is that I/O overheads are larger for high throughput than low throughput, which may seem counter-intuitive. We present a representative transfer period with the Baseline snapshot technique in Figure 8 for one trace (System 1799). The horizontal axis shows

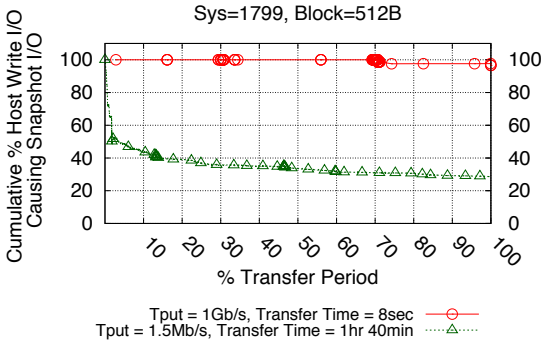


Figure 8: For high throughput, most host write I/Os cause a copy-on-write I/O, while at lower throughputs, there is less I/O overhead.

transfer time normalized to 100%, and the vertical axis shows the cumulative fraction of host I/O that causes a copy-on-write I/O for both 1.5Mb/s and 1Gb/s throughputs. For the 1Gb/s result, each mark represents a single I/O, while for 1.5Mb/s, each mark represents 1,000 I/Os.

Transfer periods can be quite short with 1Gb/s throughputs (8 seconds in this example) such that there are few I/Os during that time and those I/Os tend to be to unique blocks, which causes a copy-on-write I/O. At 1Gb/s throughput, 12-19% of systems did not experience

any host I/O during the transfer period for block sizes of 512B-1MB respectively. For slower throughputs, transfer time is longer (1 hour and 40 minutes for the T1 example), there are more I/Os, and many I/Os affect the same block. Over 99% of systems had at least some host write I/O during transfer at 1.5Mb/s.

Rule-of-thumb 2: Primary I/O should be over-provisioned by 100% to support copy-on-write related write-amplification of host writes during replication. The over-provision can be as low as 20% with a replication snapshot.

4.4 Analysis with Write Buffers

Our analysis thus far has not included the impact of buffering host write I/O on the primary storage server during incremental transfer. Write-buffering is common in practice [3], with flushes to disk either scheduled periodically or triggered through a storage API. To simulate the impact of buffering host write I/O, we have added a FIFO queue to our analysis throughout the replication interval. As host writes take place during transfer time, the corresponding blocks are added to the queue. When our queue fills, the oldest block is evicted from the queue and is written to storage, which causes a copy-on-write or redirect-on-write (for the first write to a block) with related snapshot I/O and storage overheads.

Snapshot I/O overhead for 1.5Mb/s throughput and a 12 hour replication interval is shown in Figure 9. Snapshot I/O overhead decreases rapidly as the write buffer increases from 0% to 1% of the volume's estimated capacity. Increasing the write buffer would further decrease overheads, but write buffers are typically much less than 1% of storage capacity due to differences in cost between memory and persistent storage. In contrast to snapshot I/O, we found that storage overhead for snapshots was nearly unaffected by buffer size because only the first write to a block requires snapshot storage space. We did find that both I/O and storage overheads decrease with improved replication snapshot techniques. A storage overhead figure is not shown due to space limitations.

Rule-of-thumb 3: Having a write buffer effectively decreases snapshot I/O overheads but has little impact on storage overheads.

5 Findings for Target Storage

Besides improving storage overheads for primary systems, we can also analyze how target data protection storage is impacted. How frequently can replication run? How much data will be stored? How much bandwidth is required? Answering these questions will guide the design of future data protection systems.

5.1 Transfer Size Analysis

We first investigate the amount of data to be transferred and stored for each replication interval. We investigate

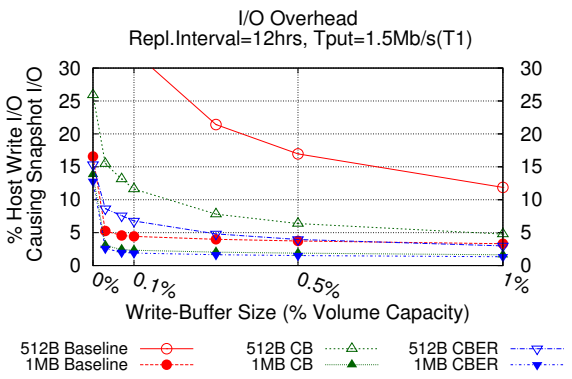


Figure 9: Snapshot I/O overhead decreases rapidly as write buffer size increases.

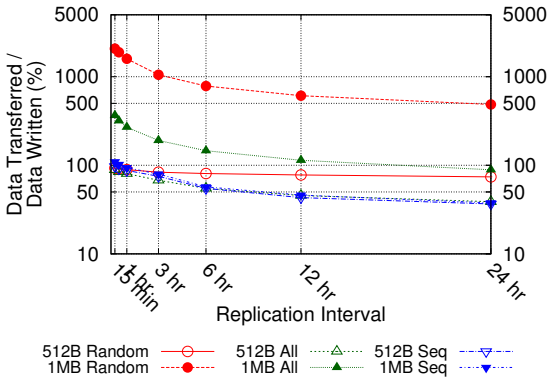


Figure 10: Data transferred as a fraction of data written gradually decreases as the replication interval increases.

the size of data transferred in Figure 10. Note that the vertical axis shows the normalized data transferred (log scale). For normalization, we divide the dirty blocks to be transferred by the amount of data written by the host to the primary system. Values will be less than 100% when a host writes to the same block multiple times, and the block only has to be transferred once because of write collapsing.

For a block size of 512 bytes across all volumes (the 512B All line overlaps with Seq), the data transferred starts at about 100% of the data written with the interval of 15 minutes and gradually decreases to about 40% with a 24-hour interval. For sequentially accessed logical volumes (Seq), results are consistent across block sizes: data transferred is >=100% of the data written when the interval is 15 minutes and gradually reaches about 40% of the data written at 24 hours. This is because sequential host write I/O tends to produce more completely dirty blocks than other I/O patterns.

Data transferred can be more than 100% because all of the sectors in a dirty block are transferred even if only a single sector in the block is actually dirty. As the interval increases, blocks are 'filled up' with more dirty data. Figure 11 shows that 512B blocks are always fully dirty

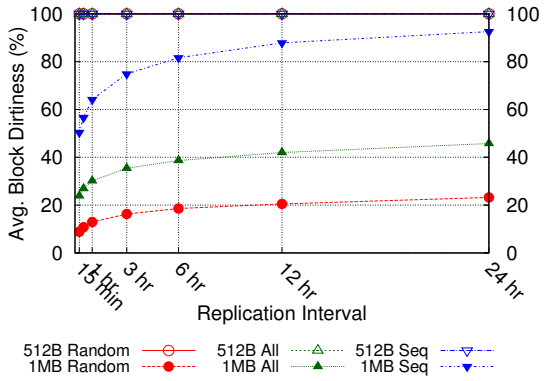


Figure 11: Except for the block size of 512B, dirty blocks are likely to contain various amounts of 'clean data,' with larger blocks more so than smaller blocks.

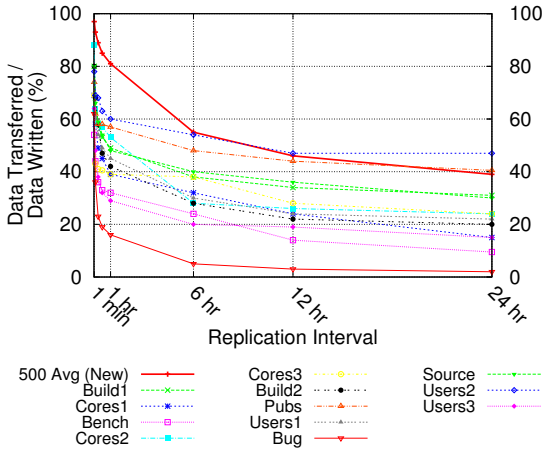


Figure 12: The **500 Avg (New)** line plots the data transferred normalized to data written by the host for each replication interval with our traces, while the other lines are reproduced from Patterson et al. [20]. All results are with 4KB blocks.

(line across the top). On the other hand, 1MB blocks become dirtier as the time between transfers increases, with most of the change in the first six hours. As expected, sequentially written volumes have much more fully dirty blocks than randomly written volumes, and block dirtiness is related to the reduction in normalized data transferred. Though not shown due to space limitations, we also found a distinct pattern that blocks were either fully dirty or dirty in multiples of 4KB or 8KB, likely due to file system and database allocation units.

These results suggest that using a large block size with a short interval can incur a significant overhead in transferring changed data to target storage. Even for small block sizes, >40% of data written daily is transferred to external systems.

Rule-of-thumb 4: The daily transfer size with small blocks is generally 40% of what hosts write.

Rule-of-thumb 5: Scheduling at least 6 hours between transfers allows blocks to achieve nearly peak dirtiness.

Comparison to previous study

A previous study in the SnapMirror system [20] of 12 file system servers examined reduction in data size to be transferred to a remote mirroring site over a range of replication intervals. In their study, the block size was fixed at 4KB. In Figure 12, our result for over 500 logical volumes (500 Avg (New)) with a 4KB block size is plotted along with a reproduction of their figure.

We find the reduction in data size to be much smaller than the SnapMirror results for intervals between 1 minute and 6 hours. Specifically, the SnapMirror study reports that all 12 systems achieve at least 30% reduction by 1 hour, while the average reduction for our traces is less than 20%. At longer intervals, our results are closer. For example, SnapMirror found a reduction in data size at 24-hour intervals to be between 53% and 98%, while we observe an average reduction of 60%.

In summary, our results are qualitatively similar, with transfer savings increasing with replication interval. The observed discrepancies are most likely due to different workloads used for analysis. The smaller number of systems studied for SnapMirror mostly supported software development and related applications, e.g., source code tree, bug tracking database, and engineer home directories, while the systems in our study support a mix of business and consumer applications and file systems.

5.2 Bandwidth Requirements

Transferring data requires sufficient bandwidth for the transfer to complete before the next replication interval or a cascade of failures occurs. Peak bandwidth was calculated for each trace, and the 90th percentile across traces is plotted in Figure 13. Results are per volume, so bandwidth for a storage system with many volumes would be higher. Logical volumes supporting sequential hosts require the most network bandwidth across all replication intervals. For replication interval > 6*hours*, the required bandwidth for the logical volumes in the Random set is similar to that for the volumes in the All set. For sequential hosts, the number of logical volumes that can simultaneously transfer changed data is largely bound by network bandwidth, while for the other volumes, the choice of block size has a significant impact. Based on the results from Figure 10, storage administrators can calculate how much bandwidth they will need to transfer changed data, which is a sizable fraction (approximately 40%) of what hosts write to primary storage.

There is clearly a relationship between the amount of data written by the host to primary storage and the

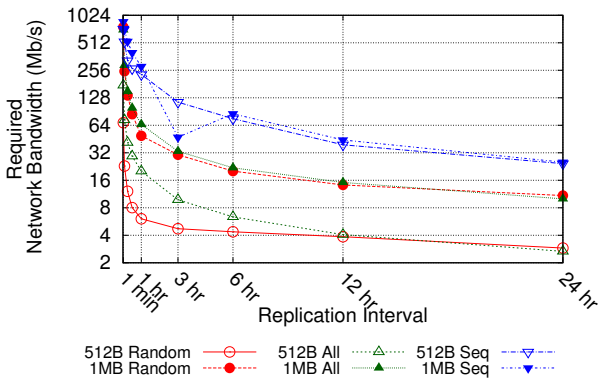


Figure 13: 90th-percentile peak network bandwidth needed to successfully transfer dirty blocks. Bandwidth is per logical volume, and the y-axis has a log scale of base 2.

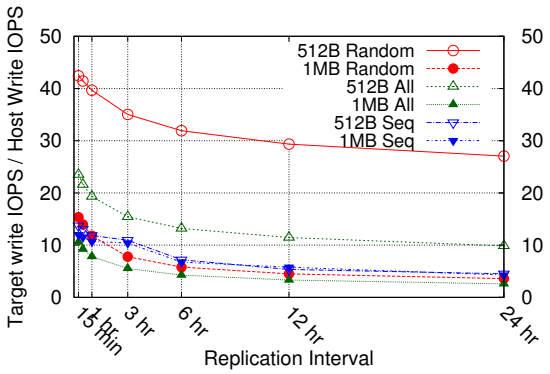


Figure 14: Ratio of target write IOPS to host write IOPS.

amount transferred to target storage, and analyzing 512B blocks shows a 99% correlation up to 30 minutes. The correlation is lower (62-85% depending on replication interval) for 1MB blocks, likely due to clean data also transferred in large blocks. We found a fairly low correlation (30-53%) between estimated volume capacity and transferred data, so capacity is less predictive of bandwidth requirements than other properties such as host write throughput.

Rule-of-thumb 6: Scheduling at least 12 hours between transfers drastically reduces peak network bandwidth requirements. Volume capacity is not predictive of bandwidth requirements.

5.3 I/O Analysis

A significant difference between primary storage and target storage designed for data protection is the I/O requirements of each system. Primary storage is designed to optimize for host I/O requirements related to email or web servers, shared file systems, or databases. While capacity matters, I/O per second is often a more critical feature. In comparison, target storage is designed for capacity and high throughput [26], so I/O per second may be of lower priority.

Figure 14 shows how the replication interval affects I/O per second requirements for target storage that is not log structured. The vertical axis is normalized relative to host I/O rates. Specifically, it shows the number of write I/O requests needed to transfer dirty blocks to the target as a percentage of the number of host write I/O requests for the same period in the original trace. See Section 3 for detailed information on how we compute write I/O to target storage.

For even a fifteen minute interval, the transfer I/O rate drops to between 10% and 40% of the host I/O rate, depending on the block size and write pattern. This sharp drop for a short interval is because we first order the dirty sectors accumulated over the interval by their logical addresses to compute write I/O needed for transfer to target storage. This ordering results in longer runs of sequential dirty sectors than created by original host write I/O requests (up to the assumed maximum 1MB transfer size). I/O savings continue up to 24 hours measured, though there is little change between 6 hours and 24, and the I/O rate for a larger block size is consistently lower than that of a smaller block size. Collecting host I/Os for a period of time is a well studied technique to reduce I/O requirements [3].

For sequentially accessed logical volumes, the transfer I/O rates for different block sizes are almost indistinguishable across all the intervals. This is because sequential host write I/O, along with our ordering of dirty sectors, produces runs of sequential dirty sectors that are $>>$ 1MB in size, so the 1MB network transfer size limitation becomes the dominating factor. For randomly accessed logical volumes, block size has a large impact on I/O requirements, requiring from 12% to 40% at 1 hour. These results indicate that it is worthwhile to configure block sizes and replication intervals for mixed and randomly accessed volumes.

While our work focuses on asynchronous replication to reduce I/O and storage requirements for target systems, an alternative is to consider synchronous replication. Synchronous replication requires a target system to have 100% of the I/O capabilities of the primary system, which would be a horizontal line added to Figure 14 at 100% on the vertical axis. Asynchronous replication can be more efficient than synchronous replication for two reasons: collapsing multiple writes to the same block before replication to reduce transferred data and reordering writes to reduce random I/O. We leave it as future work to explore the impact of each reason.

Rule-of-thumb 7: Target storage must support as much as 20% of the I/O per second capabilities of primary storage when the replication interval is at least one hour.

6 Related Work

Over the years, there have been many studies of storage workloads in various computing environments including aspects of file access and caching [3, 4, 11, 19, 23]. Leung et al. [17] analyzed I/O trace data collected from networked file servers deployed in a data center. Anderson [1] presented new techniques for collecting large, detailed traces. Analysis of high performance computing (i.e. supercomputing) workloads focused on bandwidth, I/O request inter-arrival times, idle time, and access rates [6, 7, 15, 16, 18]. Gulati et al. [9] studied characteristics and consolidation strategies for virtualized systems. Analysis for database workloads [12] has shown qualitatively similar properties to file systems.

Numerous studies have measured disk access properties including block lifetimes, access rates, response time, sequential patterns, and caching [23, 24]. Riska and Riedel [21] analyzed how I/O workloads on disk drives change depending on applications and computing environments, e.g., enterprise servers vs. desktop computers vs. consumer electronics.

Unlike these earlier works, we specifically focus on characterizing the overheads and I/O properties in transferring incremental changes on primary storage to target storage. Specifically, we analyze data changes at the physical (block) level, in part, because creating backups at the physical level is more efficient than doing so at the logical (file) level [13]. Roselli et al. [23] studied block lifetimes but not in the context of data protection. A study a decade ago by Patterson et al. [20] characterized changed data at the block level for a similar goal; see Section 5.1. Wallace et al. [26] described backup workload characteristics, though they intermixed full and incremental workloads.

Snapshots are a common technique to create a point-in-time version of data. WAFL [10], ZFS [5] and BTRFS [22] all natively support snapshots with copy-on-write as means of ensuring data consistency on disk and enabling fast restart after system crash. In these systems, snapshots are first-class objects that can be named and accessed by the end user. In the case of ZFS and BTRFS, snapshots are writable and can be updated independently from the original. In addition, snapshots are taken at the logical level, e.g., the entire file system, directories, and/or individual files. In contrast, a replication snapshot is mainly comprised of blocks written since the last transfer, is not writable, and does not persist; once the transfer is completed, the space allocated for copied-on-write blocks is reclaimed for use by primary storage or later snapshots.

There are several publications on snapshot overheads. Azagury et al. [2] and Shah [25] both report up to 7% degradation in I/O rate due to copy-on-write. We analyzed replication snapshots as a technique to reduce overheads of standard snapshots during replication. Our two versions of replication snapshots can be classified as **write-coalescing batches with atomic update** in a taxonomy for remote mirroring defined by Ji et. al. [14], with the batch size determined by replication intervals. Our asynchronous technique allows for write coalescing to reduce write size and I/O rate on target storage.

Synchronous remote mirroring [14] can also be used for protection of data changes, especially when the change rate is low and/or the geographical distance between primary and target systems is relatively short, e.g., [27, 28]. In this paper, we analyze an asynchronous approach to allow target systems whose I/O performance and storage capacity are characteristically different from primary storage, e.g., purpose-built backup appliances.

7 Discussion and Conclusion

In this paper, we have analyzed I/O traces from over 100,000 logical volumes in customer block-based primary storage systems to understand I/O characteristics and performed a detailed analysis of over 500 traces spanning at least 24 hours to gain a better understanding of incremental change patterns. New insights can help data protection expand from the realm of daily backups to more frequent updates.

Our analysis has uncovered several new findings for both primary and target storage. Overheads on primary storage due to snapshots can require both capacity and I/O to preserve point-in-time copies, though a write buffer decreases I/O requirements. For target storage, storage requirements depend on the write patterns of the host and can vary from 40% for most hosts to 100% for hosts that write sequentially. Replication requires bandwidth, which we have shown grows proportionally with the write-throughput of hosts. We have found that access patterns can change from highly sequential to highly random across different replication intervals, with a large change in data transfer characteristics. Given that the transfer interval is often statically configured by the target system administrator, our observations argue for dynamically changing block sizes and replication intervals at run time based on the host I/O access pattern.

Many findings about data patterns align with previous results: dirty blocks tend to be overwritten again within minutes or hours, the change rate grows less rapidly with longer replication intervals, and volumes tend to be modified in multiples of 4KB or 8KB. From the analysis of over 100,000 traces, we found that there is great diversity in storage requirements in terms of capacity, numbers of writes and reads, as well as average and peak throughput and I/O per second.

Acknowledgments

We would like to thank Fred Douglis, Kadir Ozdemir, Steve Smaldone, Grant Wallace, Ian Wigmore, and our reviewers for their feedback. We also thank Bill Glynn and the EMC VMAX team for providing the traces.

References

[1] E. Anderson. Capture, conversion, and analysis of an intense NFS workload. In *Proc. of the 7th USENIX Conf. on File and Storage Tech.*, 2009.

[2] A. Azagury, M. E. Factor, J. Satran, and W. Micka. Point-in-time copy: Yesterday, today and tomorrow. In *Proc. IEEE/NASA Conf. Mass Storage Systems*, 2002.

[3] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer. Non-volatile memory for fast, reliable file systems. *ACM SIGPLAN Notices*, 27(9):10–22, 1992.

[4] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a distributed file system. In *Proc. of the 13th ACM Symposium on Operating Systems Principles*, October 1991.

[5] J. Bonwick, M. Ahrens, V. Henson, M. Maybee, and M. Shellenbaum. The zettabyte file system. In *Proc. of the 2nd Usenix Conference on File and Storage Technologies*, 2003.

[6] P. Carns, K. Harms, W. Allcock, C. Bacon, S. Lang, R. Latham, and R. Ross. Understanding and improving computational science storage access through continuous characterization. *ACM Trans. on Storage*, 7(3), October 2011.

[7] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley. 24/7 Characterization of Petascale I/O Workloads. In *Proc. of the 1st Works. on Interfaces and Abstractions for Scientific Data Storage*, 2009.

[8] EMC. EMC Symmetrix VMAX. `http://www.emc.com/storage/symmetrix-vmax/symmetrix-vmax.htm`, 2013.

[9] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *Proc. of the 2nd Inter. Workshop on Virtualization Performance: Analysis, Characterization, and Tools*, 2009.

[10] D. Hitz, J. Lau, and M. Malcolm. File system design for an nfs file server appliance. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 235–246, 1994.

[11] W. W. Hsu and A. Smith. The performance impact of I/O optimizations and disk improvements. *IBM Journal of Research and Development*, pages 255–289, March 2004.

[12] W. W. Hsu, A. J. Smith, and H. C. Young. I/O reference behavior of production database workloads and the TPC benchmarks - an analysis at the logical level. *ACM Trans. on Database Systems*, 26:96–143, 2001.

[13] N. C. Hutchinson, S. Manley, M. Federwisch, G. Harris, D. Hitz, S. Kleiman, and S. O'Malley. Logical vs. physical file system backup. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation*, 1999.

[14] M. Ji, A. Veitch, J. Wilkes, et al. Seneca: remote mirroring done write. In *Proc. of the USENIX Annual Technical Conf.*, 2003.

[15] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemeyer. Workload characterization of a leadership class storage cluster. In *Proc. of the 5th Petascale Data Storage Workshop*, 2010.

[16] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/O performance challenges at leadership scale. In *Proc. of Supercomputing*, November 2009.

[17] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller. Measurement and analysis of large-scale network file system workloads. In *Proc. of the USENIX Annual Technical Conf.*, 2008.

[18] E. L. Miller, R. H. Katz, and Y. H. Katz. Analyzing the I/O behavior of supercomputer applications. In *Proc. of the 11th IEEE Symposium on Mass Storage Systems*, 1991.

[19] J. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proc. of the 10th Symposium on Operating System Principles*, 1985.

[20] H. Patterson, S. Manley, M. Federwisch, D. Hitz, S. Kleiman, and S. Owara. SnapMirror: file system based asynchronous mirroring for disaster recovery. In *Proc. of the 1st USENIX Conf. on File and Storage Tech.*, 2002.

[21] A. Riska and E. Riedel. Disk drive level workload characterization. In *Proc. of the USENIX Annual Technical Conf.*, 2006.

[22] O. Rodeh, J. Bacik, and C. Mason. Brtfs: The linux b-tree filesystem. Technical report, IBM Research Report RJ10501 (ALM1207-004), 2012.

[23] D. Roselli, J. Lorch, and T. E. Anderson. A comparison of file system workloads. In *Proc. of the USENIX Annual Technical Conf.*, 2000.

[24] C. Ruemmler and J. Wilkes. Unix disk access patterns. In *Proc. of the Winter USENIX Conf.*, 1993.

[25] B. Shah. *Disk performance of copy-on-write snapshot logical volumes*. PhD thesis, The University Of British Columbia, 2006.

[26] G. Wallace, F. Douglis, H. Qian, P. Shilane, S. Smaldone, M. Chamness, and W. Hsu. Characteristics of backup workloads in production systems. In *Proc. of the 10th USENIX Conf. on File and Storage Tech.*, 2012.

[27] H. Weatherspoon, L. Ganesh, T. Marian, M. Balakrishnan, and K. Birman. Smoke and mirrors: reflecting files at a geographically remote location without loss of performance. In *Proc. of the 7th USENIX Conf. on File and Storage Tech.*, 2009.

[28] M. Zhang, Y. Liu, and Q. Yang. Cost-effective remote mirroring using the iSCSI protocol. In *21st IEEE Conf. on Mass. Storage Systems and Tech.*, 2004.