Can We Really Recover Data If Storage Subsystem Fails?

Weijun Xiao and Qing Yang
Dept. of Electrical and Computer Engineering
University of Rhode Island, Kingston RI 02881
Tel: (401) 874-5880, Fax: (401) 782-6422
Email: {wjxiao,qyang}@ele.uri.edu

Abstract

This paper presents a theoretical and experimental study on the limitations of copy-on-write snapshots and incremental backups in terms of data recoverability. We provide mathematical proofs of our new findings as well as implementation experiments to show how data recovery is done in case of various failures. Based on our study, we propose a new system architecture that will overcome the problems of existing technologies. The new architecture can provide two-way data recovery capability with the same storage overheads and can be implemented fairly easily on existing systems. We show that the new architecture has maximum data recoverability and is practically feasible.

1. Introduction

With explosive growth of networked information services and e-commerce, data protection and recovery have become the top priority of business organizations and government institutions [1,2,3,6]. Since data is the most valuable asset of an organization, any loss or unavailability of data can cause millions of dollars of damage [3,4,5]. Unfortunately, failures do occur such as hardware failures, human errors, software defects, virus attacks, power failures, site failures, and so forth [1,2,3]. In order to protect data from possible failures and to recover data in case of a failure, data protection technology is necessary [6].

Traditionally, data protection has been done using periodical backups. At the end of a business day or the end of a week, data are backed up to tapes. Depending on the importance of data, the frequency of backups varies. The higher the backup frequency, the larger the backup storage is required. In order to reduce the backup volume size, technologies such as copy-on-write (COW) snapshots and incremental backups have been commonly used. Instead of making full backups every time, COW snapshots and incremental backups that only store the changed data are done more frequently in between full backups. For example, one can do daily incremental backups and weekly full backups that are stored at both the production site and the backup site. In this way, great storage savings are possible while keeping data protected.

The way incremental backup works is as follows. Starting from the previous backup point, the storage keeps track of all changed blocks. At the backup time point, a backup volume is formed consisting of all latest changed data blocks. As a result, the incremental backup contains the newest data that have changed since the last backup. COW snapshots work differently from the incremental backup. At the time when a snapshot is created, a small volume is allocated as a snapshot volume with respect to the source volume. Upon the first write to a data block after the snapshot was started, the original data of the block is copied from the source volume to the snapshot volume. After copying, the write operation is performed on the block in the source volume. As a result, the data image at the time of the snapshot is preserved. Write I/Os after the first change to a block is performed as usual, i.e. only the first write to a block copies the original data to the snapshot volume. There have been many variations of COW snapshots in terms of implementation details for performance and efficiency purposes such as pointer remapping [23] and redirect-on-writes [16,17] etc. The main advantage of both incremental backups and COW snapshots is storage savings because only changed data are backed up.

Despite the importance of data protection and recovery, recent study has shown that 67% of backup data cannot be recovered in the real world [6]. Even if data can be recovered, it takes hours and even days to do so [6]. While this fact is well known, there has been no research study on why this is the case. Therefore, it remains unclear and an open question why such high percentage of data recovery failed.

This paper presents a theoretical study on COW snapshot and incremental backup technologies from the point of view of block level storages. Our investigation uncovers the fundamental limitations of the existing data protection technologies and provides a theoretical explanation as to why so many data recoveries (over 67% recoveries) failed using these existing technologies. We show mathematically the data recovery capabilities and limitations of the existing technologies. Based on our theoretical results, we propose a new storage architecture that overcomes the limitations of existing technologies. Instead of storing the original or the new data of a block upon a write operation, we couple the two for data protection purpose using a commutative and invertible function. This new architecture provides more flexible recovery capability than COW and incremental backups with the same storage overheads. We provide mathematical proof of the correctness of the new technology. A prototype system has been built and standard I/O benchmarks and real world I/O workloads are used to test our implementation and to measure the performance of the new architecture as compared to the existing ones. Our experiments establish recoverability and performance of the new data protection technology as well as the existing ones.

The paper is organized as follows. Next section gives the proofs of the capabilities and limitations of the existing technologies. We present our new storage architecture in Section 3. Section 4 describes our experimental settings and implementations. Experimental results and discussions are presented in Section 5. We summarize related work in Section 6 and conclude the paper in Section 7.

2. Capabilities and limitations of current data protection technologies

Consider the two data protection technologies: COW snapshot and incremental backup. COW snapshot keeps the original data upon a write operation whereas incremental backup keeps the freshest data. In order to study the capabilities and limitations of these existing technologies, we formally define several mathematical terms and their relationships with the storage technologies.

Let us assume that the data storage we try to study consists of independent and equally sized data blocks (the specific size of a block is not significant in this discussion). Each of these data blocks is identified by an LBA (logic block address) and contains a specific data value. Let $\boldsymbol{A}$ be the entire set of LBAs of the data storage considered and $\boldsymbol{D}$ represent the set of all possible data values contained in data blocks. A binary relation, $\boldsymbol{R}$, between $\boldsymbol{A}$ and $\boldsymbol{D}$ defines a mapping of addresses to their corresponding data values of the data storage. Since there is exactly one ordered pair in $\boldsymbol{R}$ with each LBA, this binary relation is a function. We refer to this function as *storage data* and use F_t to represent this function (storage data) from $\boldsymbol{A}$ to $\boldsymbol{D}$ at time t. And we use $F_t(a)$ to represent the image or data value of an LBA a. That is, F_t contains a set of ordered pairs such as $\{(a_1,d_1), (a_2,d_2) \ldots\}$ whereas $F_t(a)$ is an image/data value of a such as $F_t(a_1)= d_1$. If A' is a subset of A, i.e. $A' \subseteq A$, then we use F_t /A' to represent the restriction of F_t to A'. That is, $F_t /A' = F_t \cap (A' \times D)$ [7]. Without loss of generality, let us consider three time points as shown in the following diagram.

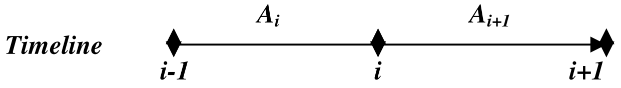

*Figure 1. A three-point timing diagram:**i-1** starting point, **i+1** current point, and **i** recovery point.*

Suppose that time point i-1 represents the original time point when data storage operation starts and time point i+1 represents the current time point. Suppose a failure occurred at some time close to point i+1. We are interested in recovering data to the data as it was at time point i. We use integer numbers to represent time points since all storage events occur at discrete time points with a clear sequential ordering.

Definition 1. Let $A_i \subseteq A$ be a set of LBAs. We define A_i to be a *write set* i if it contains all LBAs whose data value have been overwritten between time point i-1 and time point i.

Looking at the diagram shown in Figure 1, we have A_i containing all LBAs whose data values have been changed by write operations between time point i-1 and time point i while A_{i+1} containing all those between time points i and i+1.

Example 1. If we have $F_i = \{(0,2), (1,5), (2,8)\}$ at time point i and $F_{i+1} = \{(0,4), (1,5), (2,0)\}$ at time point i+1 because of write operations, then we have $A_{i+1} =\{0,2\}$. That is, data values at addresses 0 and 2 have been changed from 2 and 8 to 4 and 0, respectively, whereas the data value of address 1 has not been changed, since time point i.

It is possible that the overwritten value as seen at time i is the same as the original value at time i-1 caused by one or several write operations between time points i-1 and i. We therefore define *substantial write set* that actually changed data values as follows.

Definition 2. Let $A'_i \subseteq A_i$. We define A'_i to be a *substantial write set* i if the data value of every LBA in A'_i has been changed between time points i-1 and i.

It should be noted here that the changed data value is generally not related to the original value because of the nature of write operations at block level storages. That is, $F_{i+1}(a)$ is independent of $F_i(a)$. Furthermore, $F_i(a)$ is independent of $F_i(b)$ for all $b \in A$ and $b \neq a$ as stated in the beginning of this section: data blocks are independent. We believe this assumption is reasonable because block level storages regard each data block as an independent block without any knowledge of file systems and applications above them.

Definition 3: A COW snapshot as seen at time i+1 that was started at time i is defined as F_i/A_{i+1}, where A_{i+1} is *write set* i+1.

As we know, COW snapshot makes a copy of the original data upon the first write to the block. As a result, it keeps a set of original data of all changed blocks since the snapshot started. Consider the storage data in Example 1. Suppose the COW snapshot was started at time point i. At time point i+1, we have the snapshot: {(0,2), (2,8)}, which is F_i/A_{i+1}. That is, A_{i+1} gives all the LBAs that have been written, {0,2}, and their respective images in the snapshot should be the same as they were at time point i, {2,8}.

Lemma 1. If we have storage data at time i+1 and a COW snapshot started at time i, then we can recover data as they were at time i as follows:

$$F_i = (F_{i+1} - F_{i+1}/A_{i+1}) \cup F_i/A_{i+1} , \qquad \textbf{(1)}$$

where "-" and "$\cup$' are difference and union operators of sets, respectively.

Lemma 1 gives the data recovery capability of COW snapshot technology. It is able to recover data to a previous time point provided that the most recent data is available. This data recovery capability is very useful in practice in case of data corruption, virus attack, user errors, software bugs, and so forth. If we know that data was good at a previous time point when snapshot was started, we can go back to that point to recover from failures caused by this type of events.

Although COW snapshot can recover data to a previous time point as stated in Lemma 1, it has limitations. In particular, if the current data (production data) is damaged or lost because of hardware failures, OS failures, outages, or disasters, we cannot recover data to a previous time point even if we have COW snapshots and previous backup data that may be safely stored in a remote backup site. This limitation is formally stated in the following theorem.

Theorem 1. Suppose the storage data at time point $\boldsymbol{i+1}$, $\boldsymbol{F_{i+1}}$, is not available and the substantial write set $\boldsymbol{A'_i}$ is not empty ($\boldsymbol{A'_i} \neq \phi$). COW snapshots cannot recover storage data $\boldsymbol{F_i}$ as they were at time point $\boldsymbol{i}$ if $\boldsymbol{A'_i} \nsubseteq \boldsymbol{A_{i+1}}$.

Proof:

We prove this theorem by contradiction. Let us assume that COW snapshots can recover storage data $\boldsymbol{F_i}$ *as they were at time point* $\boldsymbol{i}$ *without* $\boldsymbol{F_{i+1}}$. *That is, for all* $\alpha \in A$, *we can reconstruct* $\boldsymbol{F_i}(\alpha)$ *from what we have available:*

a) *Data backup made previously:* $\boldsymbol{F_{i-1}}$
b) *COW snapshot as seen at time* $\boldsymbol{i}$ *that was started at time* $\boldsymbol{i-1}$: $\boldsymbol{F_{i-1}/A_i}$, *and*
c) *COW snapshot as seen at time* $\boldsymbol{i+1}$ *that was started at time* $\boldsymbol{i}$: $\boldsymbol{F_i/A_{i+1}}$.

Since different data blocks are independent in our storage system, for every LBA $\boldsymbol{\alpha \in A}$, *the only way to reconstruct its data value,* $\boldsymbol{F_i}(\alpha)$, *is to reconstruct it from* $\boldsymbol{F_{i-1}}(\alpha)$, $\boldsymbol{F_{i-1}/A_i}(\alpha)$, *and/or* $\boldsymbol{F_i/A_{i+1}}(\alpha)$.

Because $\boldsymbol{A'_i} \nsubseteq \boldsymbol{A_{i+1}}$ *and* $\boldsymbol{A'_i} \neq \phi$, *there is an LBA that is in* $\boldsymbol{A'_i}$ *but not in* $\boldsymbol{A_{i+1}}$. *Let* β *be such an LBA such that* $\beta \in \boldsymbol{A'_i}$ *but* $\beta \notin \boldsymbol{A_{i+1}}$. *Now consider the three cases:*

a) *Since* $\beta \in \boldsymbol{A'_i}$, *we have* $\boldsymbol{F_i}(\beta) \neq \boldsymbol{F_{i-1}}(\beta)$ *by* ***Definition 2****.*
b) *Because* $\boldsymbol{F_{i-1}/A_i} \subseteq \boldsymbol{F_{i-1}}$ *and* $\boldsymbol{A'_i} \subseteq \boldsymbol{A_i}$, *we have* $\boldsymbol{F_{i-1}/A_i}(\beta) = \boldsymbol{F_{i-1}}(\beta) \neq \boldsymbol{F_i}(\beta)$
c) *The fact that* $\beta \notin \boldsymbol{A_{i+1}}$ *implies that* $\boldsymbol{F_i/A_{i+1}}(\beta)$ *is undefined because* β *is not in the domain of* $\boldsymbol{F_i/A_{i+1}}$.

Furthermore, $\boldsymbol{F_i}(\beta)$ *is not related in any way to* $\boldsymbol{F_{i-1}}(\beta)$ *because of the nature of write operations at block level storages. As a result, it is impossible to rebuild* $\boldsymbol{F_i}(\beta)$ *from* $\boldsymbol{F_{i-1}}(\beta)$, $\boldsymbol{F_{i-1}/A_i}(\beta)$, *and/or* $\boldsymbol{F_i/A_{i+1}}(\beta)$, *a contradiction to our assumption. Therefore, COW snapshots cannot recover storage data* $\boldsymbol{F_i}$. □

Example 2. Consider one example with 6 blocks in a storage volume. At time point $\boldsymbol{i-1}$, we have {(0, a_0), (1, b_0), (2, c_0), (3, d_0), (4, e_0), (5, f_0)}. From time point $\boldsymbol{i-1}$ to time point $\boldsymbol{i}$, three blocks have been changed to: {(0, a_1), (1, b_1), (3, d_1)}, with the substantial write set being {0, 1, 3}. From time point $\boldsymbol{i}$ to time point $\boldsymbol{i+1}$, two blocks have been changed to: {(3, d_2), (4, e_2)} with the substantial write set being {3, 4}. By Definition 3, we have snapshot $\boldsymbol{F_{i-1}/A_i}$ as {(0, a_0), (1, b_0), (3, d_0)} and snapshot $\boldsymbol{F_i/A_{i+1}}$ as {(3, d_1), (4, e_0)}. When original data $\boldsymbol{F_{i-1}}$ is unavailable, storage data $\boldsymbol{F_i}$ can be reconstructed from COW snapshot $\boldsymbol{F_i/A_{i+1}}$ and $\boldsymbol{F_{i+1}}$ by replacing the changed blocks (3, d_2) and (4, e_2) in $\boldsymbol{F_{i+1}}$ with original data blocks (3, d_1) and (4, e_0) in $\boldsymbol{F_i/A_{i+1}}$, respectively. If fresh data $\boldsymbol{F_{i+1}}$ is damaged, however, $\boldsymbol{F_i}$ cannot be recovered from $\boldsymbol{F_{i-1}}$ and snapshots because substantial write set $\boldsymbol{A'_i}$ is not a subset of write set $\boldsymbol{A_{i+1}}$ as stated in Theorem 1. In this particular case, data blocks (0, a_1) and (1, b_1) cannot be rebuilt from original data $\boldsymbol{F_{i-1}}$ and snapshots in any way.

Definition 4: The incremental backup as seen at time $\boldsymbol{i}$ that was started at time $\boldsymbol{i-1}$ is defined as $\boldsymbol{F_i/A_i}$, where $\boldsymbol{A_i}$ is *write set* $\boldsymbol{i}$.

Incremental backups keep the latest changes in a data storage. Consider Example 1 again, the incremental backup as seen at time point $\boldsymbol{i}$ that was started at time point $\boldsymbol{i-1}$ is {(0, 4), (2, 0)}. In Example 2, the incremental backup as seen at time point $\boldsymbol{i}$ that was started at time point $\boldsymbol{i-1}$ is {(0,a_1),(1,b_1),(3,d_1)}.

Lemma 2. If we have storage data at time point $\boldsymbol{i-1}$ and an incremental backup as seen at time $\boldsymbol{i}$ that was started at time point $\boldsymbol{i-1}$, then we can recover data as they were at time $\boldsymbol{i}$ as follows:

$$\boldsymbol{F_i = (F_{i-1} - F_{i-1}/A_i) \cup F_i/A_i} , \qquad \textbf{(2)}$$

where "**-**" and "$\boldsymbol{\cup}$" are difference and union operators of sets, respectively.

Lemma 2 gives the redo recovery capability of incremental backup technology. It is able to recover data to a recent time point when the original storage data is available. This redo recovery can be used in practice in case of disk failures, volume crash, OS failures, outages, disasters, and so on. If we created a full data backup prior to the incremental backup was started, we can reconstruct the storage data to the latest time point in case of this type of failures.

While incremental backup can recover data as stated in Lemma 2, it also has limitations. Particularly, if the current data gets corrupted because of virus or user errors and it happens that we do not have a prior full backup, we cannot recover data to a good time point using incremental backups and current data that are available. This limitation is formally stated in Theorem 2.

Theorem 2. Suppose the storage data at time point $\boldsymbol{i-1}$, $\boldsymbol{F_{i-1}}$, is not available and substantial write set $\boldsymbol{A'_{i+1}}$ is not empty ($\boldsymbol{A'_{i+1}} \neq \phi$). Incremental backups cannot recover storage data $\boldsymbol{F_i}$ as they were at time point $\boldsymbol{i}$ if $\boldsymbol{A'_{i+1}} \nsubseteq \boldsymbol{A_i}$.

The proof of this theorem is similar to the proof of Theorem 1 and omitted here because of page limitation.

Example 3. Using the same storage scenario as Example 2, we give an example of incremental backups. From Example 2, we have incremental backup $\boldsymbol{F_i/A_i}$ as {(0, a_1), (1, b_1), (3, d_1)} and incremental backup $\boldsymbol{F_{i+1}/A_{i+1}}$ as {(3, d_2), (4, e_2)}. When fresh data $\boldsymbol{F_{i+1}}$ is damaged, storage data $\boldsymbol{F_i}$ can be recovered from $\boldsymbol{F_{i-1}}$ and incremental backup $\boldsymbol{F_i/A_i}$ by overwriting all data blocks in $\boldsymbol{F_i/A_i}$ at the positions of storage data $\boldsymbol{F_{i-1}}$. However, if original data $\boldsymbol{F_{i-1}}$ is unavailable, storage data F_i cannot be rebuilt from F_{i+1} and incremental backups because A'_{i+1} is not a subset of A_i as stated in Theorem 2. Particularly, data block (4, e_0) in F_i cannot be generated by fresh data F_{i+1} and incremental backups in any way.

3. A new architecture for data protection

As we described in Section 2, snapshots cannot redo storage data to a recent time point while incremental backups cannot undo storage data to a previous good point. The reason is that snapshots do not keep the fresh data and incremental backups do not store the original data. To overcome the limitations, we propose a new architecture for data protection. The idea is simple. Instead of storing the original or the new data of a block upon a write operation, we couple the two using a commutative and invertible function. The result of the coupling is stored for data protection purpose. The function should be computationally efficient and should result in the same size data block for the function value. With wide availability of high speed and low cost embedded processors, this can be done easily and efficiently [8]. For example, addition and Exclusive-OR are such functions.

In general, Let us define G_i to be a function at time point i on A_i, the same domain as snapshot F_{i-1}/A_i and incremental backup F_i/A_i. Similarly, we can have G_{i+1} defined on A_{i+1} at time point $i+1$. If snapshot F_i/A_{i+1} can be obtained from G_{i+1} and F_{i+1}, or incremental backup F_i/A_i can be obtained from G_i and F_{i-1}, we can still apply Equation (1) in Lemma 1 for undo recovery, or Equation (2) in Lemma 2 for redo recovery. In other words, G_i can provide two-way data recovery. On the other hand, G_i has the same number of ordered pairs as snapshot F_{i-1}/A_i or incremental backup F_i/A_i because they have the same function domain A_i. That is, G_i needs the same size storage space as F_{i-1}/A_i or F_i/A_i if we assume data values of each LBA in F_{i-1}/A_i , F_i/A_i, and G_i occupy same storage space. Therefore, G_i is our objective function that needs to be designed.

Theorem 3. Let “+” be a commutative binary operator on D and $G_i(\alpha)= F_{i-1}/A_i(\alpha) + F_i/A_i(\alpha)$ for all $\alpha \in A_i$. If there exists an invertible operator “-” on D, such that for any $d_1,d_2 \in D$, $d_1+d_2-d_2=d_1$, then the storage data at time i, F_i, can be recovered from F_{i+1} and G_{i+1} by an undo process when F_{i-1} is unavailable, or from F_{i-1} and G_i by a redo process when fresh data F_{i+1} is damaged or lost.

Proof: *We prove this theorem in two steps corresponding to two cases.*

a) *Original data F_{i-1} is unavailable.*

First, let us consider function G_{i+1} at time point $i+1$: $G_{i+1}(\beta) = F_i/A_{i+1}(\beta) + F_{i+1}/A_{i+1}(\beta)$ for all $\beta \in A_{i+1}$. From this equation, we know $F_i/A_{i+1}(\beta) = G_{i+1}(\beta) - F_{i+1}/A_{i+1}(\beta)$ by applying invertible operator “-” to $F_{i+1}/A_{i+1}(\beta)$ on both sides of the equation. Furthermore, $F_{i+1}/A_{i+1} \subseteq F_{i+1}$ implies $F_{i+1}/A_{i+1}(\beta)=F_{i+1}(\beta)$. Replacing $F_{i+1}/A_{i+1}(\beta)$ with $F_{i+1}(\beta)$ in above equation, we have $F_i/A_{i+1}(\beta)=G_{i+1}(\beta) - F_{i+1}(\beta)$. In other words, snapshot F_i/A_{i+1} started at time point i can be obtained from fresh data F_{i+1} and G_{i+1}. By applying Equation (1) in Lemma 1, storage data F_i can be recovered from F_{i+1} and G_{i+1}.

b) *Fresh data is damaged or lost.*

Consider function G_i at time point i: $G_i(\alpha)= F_{i-1}/A_i(\alpha) + F_i/A_i(\alpha)$ for all $\alpha \in A_i$. Since operator “+” is commutative, we have $G_i(\alpha)= F_i/A_i(\alpha) + F_{i-1}/A_i(\alpha)$. Applying the inverse operation to above equation, we obtain $F_i/A_i(\alpha)=G_i(\alpha) - F_{i-1}/A_i(\alpha)$. Because $F_{i-1}/A_i \subseteq F_{i-1}$, we have $F_{i-1}/A_i(\alpha)=F_{i-1}(\alpha)$. Replacing $F_{i-1}/A_i(\alpha)$ with $F_{i-1}(\alpha)$ in above equation, we have $F_i/A_i(\alpha)=G_i(\alpha) - F_{i-1}(\alpha)$. This equation indicates that incremental backup F_i/A_i can be obtained from original data F_{i-1} and G_i. By applying Equation (2) in Lemma 2, storage data F_i can be reconstructed from F_{i-1} and G_i. □

Theorem 3 indicates that G_i can provide two-way data recovery with the same amount of storage space overhead as COW snapshot and incremental backups. As shown in Theorem 3, any commutative binary operator with an invertible operator can be used to define function G_i. For example, simple addition, Exclusive-OR, or inverse Exclusive-OR can be chosen for G_i. G_i trades off high-speed computation for storage space over the approach of keeping both versions of data. We can leverage powerful computation capability of modern computer systems to save storage space. Large storage space not only is costly but also takes more time to recover data, which is undesirable.

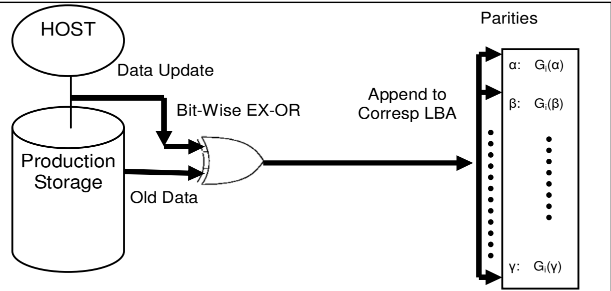

Figure 2. Overview of CUP design

Example 4. We give an example of function G_i by using Exclusive-OR operation. Suppose $G_i = F_{i-1}/A_i \oplus F_i/A_i$, where $\oplus$ is logical Exclusive-OR operator. By computing parities between the original data and the fresh data, we store parities at time i and $i+1$ for recovery. We therefore call this method ***CUP***: ***C***oupling ***U***pdates by ***P***arties. Obviously, CUP can recover storage data in two ways from parities. Figure 2 shows the overall structure of CUP design. Instead of storing either the newly updated data block or the old data block upon an update from the host computer, we couple both using an Exclusive-OR function.

4. Experimental settings and implementations

To verify the data recoverability and enable quantitative performance evaluation for the three data protection technologies: COW snapshot, incremental backup, and CUP, we have designed and implemented these three data protection technologies embedded in an iSCSI target. Using our experimental system, we install our prototype software on a PC serving as a storage server, as shown in Figure 3. Two PCs are interconnected using Intel's NetStructure 10/100 /1000Mbps 470T switch. One of the PCs acts as a client running benchmarks with iSCSI initiator installed and the other acts as the storage server with our iSCSI target installed. On top of the iSCSI target and the data protection module, we set up Postgres Database 8.1.4. We chose a database benchmark TPC-C [9,9] and two File system benchmarks, PostMark [10] on Linux Ext3 and IoMeter [11] on Windows NTFS. For TPC-C benchmark, we used the implementation from TPCC-UVA [12]. Five warehouses with 50 users were built on a Postgres database taking 2GB storage space. For PostMark, we chose a workload that performs 200,000 transactions on 200,000 files. *Read* and *Write* buffer sizes were set to 4KB. We ran the IoMeter on NTFS with 4KB block size for the workload of 67% random *writes* and 33% random *reads*.

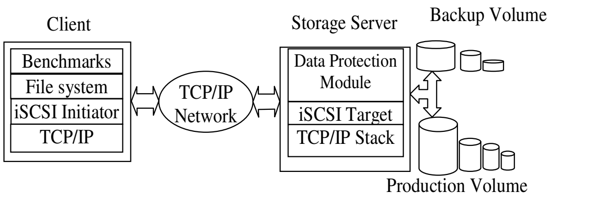

Figure 3. Software stack of the experimental system

5. Experimental results and discussions

5.1. Recoverability

Based on our design and implementation of the three data protection technologies, we carried out a recovery experiment to verify the capability and limitation of COW snapshots. This experiment simulated an editing process of our paper using Microsoft Word 2007. We picked up three time points as $\boldsymbol{i-1}$, $\boldsymbol{i}$, and $\boldsymbol{i+1}$ with 2 minutes interval between two adjacent time points and enabled COW snapshot for data protection. At the beginning of time point $\boldsymbol{i-1}$, we had a word document file that had only a title and an abstract for the paper. The size of the file was 12KB. From time points $\boldsymbol{i-1}$ to $\boldsymbol{i}$, we added new text to the paper. The size of the file became 16KB. Later on we accidentally deleted some text and only left the title. The size of the file shrank to 11KB. The accident time was between time points $\boldsymbol{i}$ and $\boldsymbol{i+1}$. At the storage server side, we collected all LBA traces for verification analysis. In this experiment, two COW snapshots were made one started at time point $\boldsymbol{i-1}$ and the other started at time point $\boldsymbol{i}$. Our first recovery attempt was to do an undo recovery by writing the snapshot started at time point $\boldsymbol{i}$ to the fresh data at time point $\boldsymbol{i+1}$. As a result of this attempt, we were able to undo storage data to time point $\boldsymbol{i}$ and opened the word file. This confirms the recoverability of COW snapshots using the undo process.

Our second recovery attempt was to do a redo recovery assuming that the fresh data is lost. After we destroyed the fresh data at time point $\boldsymbol{i+1}$, we tried to recover data to time point $\boldsymbol{i}$ in three possible cases using only the original data at time point $\boldsymbol{i-1}$ and two snapshots started at time points $\boldsymbol{i-1}$ and $\boldsymbol{i}$, respectively. First, we overwrote the snapshot started at time point $\boldsymbol{i-1}$ to storage data at time point $\boldsymbol{i-1}$. The word file was opened but with the contents same as the one at time point $\boldsymbol{i-1}$ because snapshot started at time point $\boldsymbol{i-1}$ had the same data values as original storage data for changed blocks between time points $\boldsymbol{i-1}$ and $\boldsymbol{i}$. The newly typed text from time $\boldsymbol{i}$ $\boldsymbol{-1}$ to $\boldsymbol{i}$ was lost and the size of the file was still 12KB. Secondly, we overwrote the snapshot started at time point $\boldsymbol{i}$ to storage data at time point $\boldsymbol{i-1}$. The file size became 16KB, but the word file could not be opened because data was corrupted. We observed the same results for the third case where we overwrote two snapshots to storage data at time point $\boldsymbol{i-1}$. Therefore, we failed to recover data to time point $\boldsymbol{i}$ for all three cases. By analyzing LBA traces, we found that both substantial write set $\boldsymbol{A'_i}$ and write set $\boldsymbol{A_{i+1}}$ contained 35 LBAs with 5 LBAs being different. That is, $\boldsymbol{A'_i} \not\subseteq \boldsymbol{A_{i+1}}$. As stated in theorem 1, data cannot be recovered to time point $\boldsymbol{i}$ by COW snapshots. This conclusion is consistent with our recovery experiment.

Having tested the capability and limitation of COW snapshots, we carried out a similar recovery experiment to verify two-way recovery capability of CUP. By using the same storage operations as our first recovery experiment discussed above, we stored parities at time points $\boldsymbol{i}$ and $\boldsymbol{i+1}$ instead of COW snapshots. When original data $\boldsymbol{F_{i-1}}$ was deleted, we took parities at time point $\boldsymbol{i+1}$ and fresh data $\boldsymbol{F_{i+1}}$ to compute snapshot $\boldsymbol{F_i/A_{i+1}}$ back. We then used the snapshot together with the fresh data to recover storage data $\boldsymbol{F_i}$ using the undo process. This recovery process was done successfully and the word file contains the data at time point $\boldsymbol{i}$. On the other hand, when we destroyed the fresh data with only the original data at time point $\boldsymbol{i-1}$ and parities being available, we used parities at time point $\boldsymbol{i}$ and original data $\boldsymbol{F_{i-1}}$ to generate incremental backup $\boldsymbol{F_i/A_i}$. We then tried to recover storage data $\boldsymbol{F_i}$ using the redo process. We were able to recover data and the word file was recovered correctly as at time point $\boldsymbol{i}$. Therefore, CUP can recover data in two directions. This fact is consistent with our theoretical proof of Theorem 3.

5.2. Performance evaluation

CUP architecture provides additional recovery capability over COW snapshots and incremental backups. Specifically, it is capable of recovering data in two directions, redo and undo. One immediate question is whether such additional capability comes at high cost. In order to quantitatively evaluate how CUP performs in comparison with COW snapshots and incremental backups, we carried out two experiments to measure and compare the performances of the three data protection technologies.

Using the performance of incremental backup technology as a baseline reference, we define performance penalty of CUP as:

$$Penalty_{cup} = \frac{Thrput\ of\ Backup - Thrput\ of\ CUP}{Thrput\ of\ Backup} \quad \textbf{(3)},$$

and performance penalty of COW snapshots as:

$$Penalty_{cow} = \frac{Thrput\ of\ Backup - Thrput\ of\ COW}{Thrput\ of\ Backup} \quad \textbf{(4)}.$$

Our first experiment is to compare the performances of the three data protection technologies assuming the data protection interval to be 5 minutes. That is, the storage system will take incremental backup, COW snapshot, or CUP at every 5 minutes so that in case of failures one can recover data to 5 minutes ago. We ran the three benchmarks described in the previous section on our experimental system. TPC-C benchmark was run on Postgres database with each of the three different data protection technologies enabled. We measured tpmC, the number of transactions finished per minute, as performance results. For the two file system benchmarks, we measured IOps (I/O operations per second) for IoMeter and transaction rate (files per second) for PostMark as performance results, respectively. After measuring all performance results directly from the experiment, we calculated the performance penalties as defined in Equations (3) and (4) above.

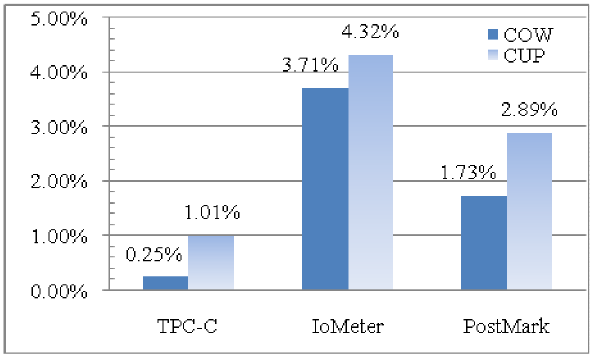

Figure 4. Performance penalty comparison (data protection interval=5 minutes)

Figure 4 shows the results in terms of performance penalty of CUP and COW snapshots for the three benchmarks when data protection interval is five minutes. As shown in Figure 4, both CUP and COW snapshots have lower performance than incremental backups. The penalty ranges from a fraction of percentage up to 4.32%. The reason is that incremental backups do not need to read the original data from the production storage upon the first write to a block while COW snapshots copy it to the snapshot volume and CUP needs it for parity computation. Furthermore, it is also shown in Figure 4 that CUP has slightly lower performance than COW snapshots. The difference of the two goes up to 1.16% because CUP needs additional Exclusive-OR computations.

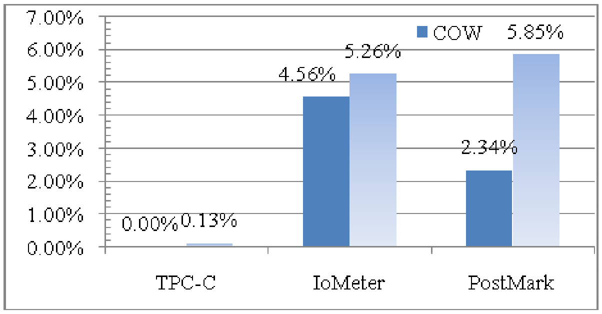

Figure 5. Performance penalty comparison (data protection interval=2 minute)

In the second experiment, we changed data protection interval from five minutes to two minutes. Again, we ran the three benchmarks with the same parameter settings as the first experiment to measure the performance results of the three data protection technologies. Figure 5 shows the results for the three benchmarks when data protection interval is two minutes. As shown in Figure 5, both CUP and COW snapshots have lower performance than incremental backups with maximal penalty of 5.26%. CUP has slightly lower performance than COW snapshots. The performance penalty of CUP goes as high as 2.51% compared to COW snapshots. One exception is that COW snapshots have the same performance as incremental backups for TPC-C benchmark. One possible reason for the exception is that the frequency of write requests when running TPC-C benchmark is so low that the additional read overhead of COW snapshots is unnoticeable.

Our experiments clearly demonstrated that CUP has comparable production performance as COW snapshots and incremental backups. The maximum performance penalty is less than 6% in all cases considered. This performance penalty comes from the additional computation overhead and data copying when Exclusive-OR function is performed to obtain parities. It is important to note that our evaluation here is very conservative with very high backup frequencies: 2 and 5 minutes data protection intervals as opposed to hourly or daily backups commonly done in practice. There are many possible ways to minimize the performance penalty with design optimizations. For example, effective caching techniques can be used to hide the latency of data copying. Furthermore, embedded systems or FPGA hardware can be used to carry out the Exclusive-OR computations that are done in parallel to production storage operations [13]. These design optimizations are possible topics of our future research.

6. Related work

Backup and snapshot technologies have been widely used in storage industry for data protection and recovery. A good summary of various backup techniques can be found in [14] and a survey for snapshot can be found in [15]. Basically, there are two types of snapshot technologies: copy-on-write and redirect-on-write [16,17]. In the literature, there are many systems that use copy-on-write to create snapshot for data backup. Plan 9 makes daily online backups by creating snapshots of the file system [18]. Petal creates a virtual disk backup using tar command through snapshots [19]. Frangipani [20] is a distributed file system built on top of Petal virtual disks that use the Petal snapshot facility to perform file system backups. VSS provides a mechanism to create consistent point-in-time copies for Windows Systems [21]. Spiralog provides online backup of a log-structured file system (LFS) [22]. All these systems can effectively recover files or volumes to an earlier version with the fresh data and snapshots.

NetApp's WAFL (Write Anywhere File Layout) generates snapshots by using redirect-on-write and pointer mapping techniques [16,23]. Both the original data and the latest data are kept in the system and all data including the original data and the latest data that are managed at the same storage space. Recent work in Thresher [24] provides a new storage management system based on copy-on-write technique to discriminate among snapshots effectively, thereby making valuable snapshots accessible online and less valuable snapshots discarded or moved offline.

Besides data backups, data can also be protected using file versioning that keeps a history of updates to files. For example, Peterson and Burns [25] designed a versioning file system named Ext3cow that uses snapshot functionality. There are many versioning file systems such as Tops-20 [26], VMS [27], Elephant [28], and CVFS [29] that also make use of copy-on-write snapshot.

Recently, continuous data protection has emerged to continually capture all changes, thus storage data can be potentially recovered to any point in time for minimizing data loss in case of errors or outages [30,31,32]. Laden et al proposed four alternative architectures for CDP in a storage controller, and compared them analytically with respect to both write performance and space usage overhead [31]. Zhu and Chiueh proposed a user-level CDP architecture that is both efficient and portable [32]. They implemented four variants of this CDP architecture for NFS servers and compared their performance characteristics. Lu et al presented an iSCSI storage system named Mariner to provide comprehensive and continuous data protection on commodity ATA disk and Gigabit Ethernet technologies [33]. Flouris and Bilas presented Clotho, a versioning system at the block level that supports creating an unlimited amount of snapshots [34]. Different from these works for CDP architectures, our study concentrates on data recoverability of block level storages and what kind of data needs to be stored for recovery purpose.

Although extensive research has been reported in the literature and various data protection products have been released in the industry, few provides formal model of storage technologies and recovery capability except for the work in [35]. Sivanthanu et al presented a logical framework for modeling the interaction of a file system with the storage system [35]. This logical framework can substantially simplify and systematize the process of verifying file system correctness such as journaling, consistent undelete, and so on. Our study focuses on providing a theoretic foundation for data protection technologies at the block level. We formally modeled existing data protection technologies and proposed a new data protection method to overcome current limitations. Based on our theoretical and experimental work, one can re-evaluate current data protection technologies and develop new data protection technologies.

It should be mentioned that we assume data blocks are independent and the changed data value is not related to the original value in our study. There are research works in the literature trying to discover block correlations or block value similarities for storage system optimization [36,37,38]. However, these block correlations and value dependencies are probabilistic. Our interest here is guaranteed data recoverability in case of various failures.

7. Conclusions

In this paper, we have presented a theoretical study on COW snapshots and incremental backups. Our theoretical work has uncovered the fundamental limitations of existing data protection technologies and explained why in some situations storage data cannot be recovered by using these existing technologies. We have provided mathematical proofs for the data recovery capabilities and limitations of the existing technologies. Based on our theoretical results, we have proposed a new architecture for data protection to overcome the limitations and given a practical example named CUP for the new technology. Instead of storing either the old data or the newly updated data, CUP couples the two for recovery purpose with the same amount of storage space as COW snapshots and incremental backups. In order to show the data recoverability and evaluate the performance of the new technology, we have implemented the three data protection technologies: COW snapshots, incremental backups, and CUP. Experimental results show that CUP can recover data either from an old backup or from fresh production data and has comparable production performance as COW snapshots and incremental backups.

Acknowledgments

This research is sponsored in part by National Science Foundation under grant CCR-0312613 and SGER Grant 0610538.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We thank Jin Ren for his participation and discussions in the design and implementation of experimental system. This work is also supported in part by National Science Foundation of China under grant number NSFC-60736013.

References

[1] D. A. Patterson et al, "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies," *UC Berkeley Computer Science Technical Report UCB//CSD-02-1175,* March 15, 2002.

[2] M. Ji, A. Veitch, and J. Wilkes, “Seneca: remote mirroring done write,” In *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, TX, 2003, pp. 253-268.

[3] D. M. Smith, “The cost of lost data,” *Journal of Contemporary Business Practice*, Vol. 6, No. 3, 2003.

[4] K. Keeton, C. Santos, D. Beyer, J. Chase, J. Wilkes, “Designing for disasters,” In *Proc. of 3rd Conference on File and Storage Technologies*, San Francisco, CA, 2004.

[5] D. A. Patterson, "A New Focus for a New Century: Availability and Maintainability >> Performance," In *FAST Keynote*, January 2002, http://www.cs.berkeley.edu/~patterson/talks/keynote.html.

[6] The 451 Group, ”Total Recall: Challenges and Opportunities for the Data Protection Industry,” May 2006, http://www.the451 group.com/reports/executive_summary.php?id=218.

[7] J. P. Tremblay and R. Manohar, “Discrete mathematical structures with applications to computer science,” New York : McGraw-Hill,1975

[8] L. Gwennap and J. Byrne, “A Guide to High-Speed Embedded Processors,” the third edition, The Linley Group, 2006.

[9] Transaction Processing Performance Council, "TPC Benchmark TM C Standard Specification," 2005, http://www.tpc.org/tpcc.

[10] J. Katcher, "PostMark: A new file system benchmark," Network Appliance, Tech. Rep. 3022, 1997.

[11] Intel Corp.,"IoMeter: Performance Analysis Tool," 2003, http://www.iometer.org/.

[12] J. Piernas, T. Cortes and J. M. García, "TPCC- UVA: A free, open-source implementation of the TPC-C Benchmark," 2005, http://www.infor.uva.es /~diego/tpcc-uva.html.

[13] S. Fuller and J. Fakiris, “How AMCC's PowerPC440SP is addressing today's storage solutions,” 2004, http://www.embedded-computing .com/articles/fuller_and_fakiris/, 2004.

[14] A.L. Chervenak, V. Vellanki, and Z. Kurmas, "Protecting file systems: A survey of backup techniques," In *Proc. of Joint NASA and IEEE Mass Storage Conference*, College Park, MD, March 1998.

[15] G. Duzy, "Match snaps to apps," *Storage*, Special Issue on Managing the information that drives the enterprise, pp. 46-52, Sept. 2005.

[16] H. Simitci, “Storage Network Performance Analysis,” Wiley Publishing, Inc., 2003.

[17] W. Xiao, Y. Liu, Q. Yang, J. Ren, and C Xie, “Implementation and Performance Evaluation of Two Snapshot Methods on iSCSI Target Storages,” In *Proc. Of NASA/IEEE Conference on Mass Storage Systems and Technologies*, May, 2006.

[18] R. Green, A. Baird, and C. Davies, "Designing a Fast, On-line Backup System for a Log-structured File System," *Digital Technical Journal*, Oct. 1996.

[19] E. K. Lee and C. A. Thekkath, "Petal: Distributed virtual disks," In *Proc. of the 7th International Conference on Architecture Support for Programming Languages an Operating Systems (ASPLOS-7)*, Cambridge, MA, 1996.

[20] C. A. Thekkath, T. Mann, and E. K. Lee, “Frangipani: A Scalable Distributed File System,” In *ACM SOSP-16*, 1997.

[21] A. Sankaran, K. Guinn, and D. Nguyen, "Volume Shadow Copy Service," March 2004, http://www. microsoft.com.

[22] M. Rosenblum and J. Ousterhout, "Log-Structured File System," In *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, June 1991, pp. 1-15.

[23] D. Hitz, J. Lau, and M. Malcolm, “File system design for an NFS file server appliance,” In *Proc. of the USENIX Winter Technical Conference, San Francisco*, CA, 1994, pp. 235-245.

[24] L. Shrira and H. Xu, “Thresher: An Efficient Storage Manager for Copy-on-write Snapshots,” In *Proceedings of 2006 USENIX Annual Technical Conference*, Boston, MA, 2006

[25] Z. Peterson and R. C. Burns, “Ext3cow: A Time-Shifting File System for Regulatory Compliance”, *ACM Transactions on Storage*, Vol.1, No.2, pp. 190-212, 2005.

[26] L. Moses, “An introductory guide to TOPS-20,” *Tech. Report TM-82-22*, USC/Information Sciences Institutes, 1982.

[27] K. McCoy, “VMS File System Internals,” Digital Press, 1990.

[28] D. S. Santry, M.J. Feeley, N.C. Hutchinson, A.C. Veitch, R.W. Carton, and J. Ofir, “Deciding when to forget in the Elephant file system,” In *Proc. of 17th ACM Symposium on Operating System Principles*, Charleston, SC, Dec. 1999, pp. 110-123.

[29] C.A.N. Soules, G. R. Goodson, J. D. Strunk, and G.R. Ganger, “Metadata efficieny in versioning file systems,” In *Proc. of the 2nd USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2003, pp. 43-58.

[30] Qing Yang, Weijun Xiao, and Jin Ren，"TRAP-Array: A Disk Array Architecture Providing Timely Recovery to Any Point-in -time" in *Proc. Of The 33rd Annual International Symposium on Computer Architecture, (ISCA'06),* pp.289-300, June 2006.

[31] G. Laden, P. Ta-shma, E. Yaffe, and M. Factor, “Architectures for Controller Based CDP”, In *Proc. of the 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, Feb. 2007.

[32] N. Zhu and T. Chiueh, "Portable and Efficient Continuous Data Protection for Network File Servers," In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 07)*, Edinburgh, UK, June 2007.

[33] M. Lu, S. Lin, and T. Chiueh, " Efficient Logging and Replication Techniques for Comprehensive Data Protection, " In *Proceedings of the 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)*, San Diego, California, Sept. 2007.

[34] M. D. Flouris and A. Bilas, "Clotho: Transparent Data Versioning at the Block I/O Level," In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*,College Park, Md. 315--328.

[35] M. Sivathanu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and S. Jha, “A logic of file systems,” In *Proceedings of the 4rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, Dec. 2005.

[36] Z. Li, Z. Chen, S. Srinivasan, and Y. Zhou, “C-Miner: Mining Block Correlations in Storage,” In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2004

[37] C. B. Morrey III and D. Grunwald, ”Peabody: The Time Travelling Disk,” In *Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies(MSST2003)*, San Diego, CA, Apr. 2003.

[38] A. C. Arpaci-Dusseau et al, "Semantically-Smart Disk Systems: Past, Present, and Future," *Sigmetrics Performance Evaluation Review*, Vol.33, No.4, 2006.