ST-CDP: Snapshots in TRAP for Continuous Data Protection

Jing Yang, Qiang Cao, *Member*, *IEEE*, Xu Li,
Changsheng Xie, *Member*, *IEEE*, and Qing Yang, *Fellow*, *IEEE*

Abstract—Continuous Data Protection (CDP) has become increasingly important as digitization continues. This paper presents a new architecture and an implementation of CDP in Linux kernel. The new architecture takes advantages of both traditional snapshot technology and recent Timely Recovery to Any Point-in-time (TRAP) architecture [41]. The idea is to periodically insert snapshots within the parity logs of changed data blocks in order to ensure fast and reliable data recovery in case of failures. A mathematical model is developed as a guide to designers to determine when and how to insert snapshots to optimize performance in terms of space usage and recovery time. Based on the mathematical model, we have designed and implemented a CDP module in the Linux system. Our implementation is at block level as a device driver that is capable of recovering data to any point-in-time in case of various failures. Extensive experiments have been carried out to show that the implementation is fairly robust and numerical results demonstrate that the implementation is efficient.

Index Terms—Data storage, data protection and recovery, data backup, continuous data protection.

1 Introduction

With the rapid advances in networked information services, data protection and recovery have become top priority in many businesses and government organizations [25], [33]. Traditionally, periodic snapshots and backups have been used to protect data, which has been fundamentally the same as it was 30 years ago [2], [5], despite the rapid advances in computer technology witnessed in the past decades. It is well known that backup remains a costly and highly intrusive batch operation that is prone to errors and consumes an exorbitant amount of time and resources. In addition, data changed between two consecutive backups are vulnerable to data loss giving rise to high Recovery Point Objective (RPO) [11], the maximum acceptable age of data at the time of outage. For example, if daily backup were used as a data protection and recovery plan, the worst case scenario would be that the recovered data would be 24 hours old if an outage happened just before the next backup.

In an ideal situation, data should be recoverable to the most recent data image as it was right before a failure event. This has spurred extensive research on Continuous Data Protection (CDP) [6], [13], [16], [18], [24], [37], [39], [41], [43]. In CDP storage, an I/O write operation on a data block does not destroy the old value of the data block. Instead, the old value is kept in a backup storage. As a result, the backup storage keeps a log (CDP log) of changed data together with timestamps indicating when data were changed for each data block. Since all data changes are kept in the CDP logs, one can recover data to any point-in-time in case of an outage or a failure event. A recovery program traces back the CDP logs to find the desired time point to restore the data image as it was at the time. In order to keep all data changes, CDP requires a large amount of storage space, which may be prohibitively costly and has thus far prevented it from being widely adopted.

To reduce the amount of space required in CDP storage, researchers have tried to find ways to store CDP logs with minimal space possible. Morrey III and Grunwald [18] presented an approach to avoiding duplicates by maintaining 128 bit content summary hash about the contents of individual sectors. Flouris and Bilas [6] have observed storage space savings by binary differential compression to store only the delta of data changes since the last version. Timely Recovery to Any Point-in-time (TRAP) architecture [41] keeps a log of XOR's (Exclusive OR) of successive data changes on a data block. Because of the strong data locality, TRAP provides up to two orders of magnitude space savings compared to plain CDP data logs. TRAP recovers data by finding the desired recovery time (RTO) point in each parity log and computing XOR from the found point to either the beginning or the end of the parity log. Depending on the length of each parity log, such XOR computation may take a long time, resulting in a large RTO (Recovery Time Objective, time it takes to recover data after a failure event).

It is clear from the above discussions that periodic backup/snapshot provides us with limited recovery points depending on the frequency of snapshots performed. But it does not need to perform XOR computations on parity logs that may potentially be very long. TRAP architecture, on the other hand, provides CDP with unlimited recovery points

- *J. Yang, Q. Cao, X. Li, and C. Xie are with the Data Storage Division, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, F304, Wuhan, Hubei 430074, P.R. China. E-mail: {jingyangcn, lixu.hust}@gmail.com, caoqiang@hust.edu.cn, cs_xie@mail.hust.edu.cn.*
- *Q. Yang is with the Department of Electrical and Computer Engineering, University of Rhode Island, Kelley Annex, Room A220, Kingston, RI 02881. E-mail: qyang@ele.uri.edu.*

Manuscript received 20 Oct. 2008; revised 21 Mar. 2011; accepted 13 Apr. 2011; published online 5 Aug. 2011.
Recommended for acceptance by M. Eltoweissy.
For information on obtaining reprints of this article, please send E-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-10-0508.
Digital Object Identifier no. 10.1109/TC.2011.150.

while it requires parity log retrieval and possibly long parity computations. We are therefore faced with two data protection and recovery techniques with a trade-off between RPO and RTO. Our objective here is to design an optimal approach to data protection and recovery by taking advantages of both techniques.

The idea is to break down the parity log into sublogs [14]. Between any two subsequent sublogs, we insert a snapshot data image. The length of the sublog is a configurable parameter determined by the system administrator or storage manager. We call our design Snapshot in TRAP CDP (ST-CDP). Adding snapshots between parity logs has several practical advantages. First of all, it limits the maximum recovery time. Second, the configurable sublog sizes allow a system administrator to organize the parity logs in different data structures and to optimize space usage and retrieval times. Third, this organization increases significantly the reliability and recoverability of the TRAP architecture. This is because a parity log may become completely useless if there is any single bit error in the log. The longer the parity log is, the higher the probability of log failure. Breaking up the parity logs into sublogs and adding snapshot in between reduces the probability of such failures and increases data recoverability.

Some of the most important design issues with ST-CDP are how often we should insert snapshots into the parity logs and how long each subparity log should be. In order to provide quantitative guidance to make such design decisions, we have developed a mathematical model as an analytical tool for optimizing space usage and recovery time for each individual block in our previous paper [14]. In this paper, we enhance the model to optimize for the entire volume of a storage and give a new set of equations to guide different users who may weigh recovery time and space costs differently. The next important design issue is how to organize parity logs and snapshot data. Efficient data structures are important to online performance, space usage, and recovery speed. Based on [14], we have studied this issue in depth and proposed a new and optimal way of organizing the parity logs with periodic snapshots inserted in the logs.

Based on our mathematical model and considerations of practical design issues, we have designed and implemented a prototype ST-CDP in Linux kernel. Our implementation is done as a block level device driver and represents substantial improvement in terms of robustness and functionality over our previous work [14]. Specifically, it is implemented at the same layer as Linux software RAID, MD [15]. Standard benchmarks such as TPC-C, IOMeter, PostMark, and microbenchmarks are used to drive our prototype for the purpose of testing and evaluation. The new prototype allows us to carry out comprehensive experiments to evaluate space efficiency, performance, and recovery time of ST-CDP as compared to traditional systems. Our measurement results showed that our implementation provides good space efficiency, high performance, and fast data recovery.

The paper is organized as follows: We discuss related work in the next section. Section 3 presents the ST-CDP architecture. We will give a brief overview of TRAP architecture for the purpose of completeness followed by our new ST-CDP design. Section 4 presents the mathematical model that is used as a guide for optimal implementation of the ST-CDP architecture. The detailed design and implementation of ST-CDP as a device driver in Linux kernel are discussed in Section 5 followed by our experimental settings in Section 6. Section 7 gives numerical results and discussions. We conclude our paper in Section 8.

2 Related Work

Depending on the different values of RPO and RTO, there exist different storage architectures capable of recovering data upon an outage. We summarize existing related works in three different categories based on different RPOs.

2.1 Snapshot or Incremental Backup

Data protection and recovery have traditionally been done using periodic backups and snapshots [2], [5], [40]. Typically, backups are done nightly when data storage is not being used since the process is time consuming and degrades application performance. During the backup process, user data are transferred to a tape, a virtual tape, or a disk for disk-to-disk backup [4], [5]. To save backup storage, most organizations perform full backups weekly or monthly with daily incremental backups in between. Data compression is often used to reduce backup storage space [2]. ST-CDP differs from existing backup and snapshot techniques by increasing the data recovery granularity using sublogs of parities resulting from data changes.

2.2 File Versioning

Besides periodic data backups, data can also be protected at the file system level using file versioning that records a history of changes to files. Versioning was implemented by some early file systems such as Cedar File System [7], 3DFS [12], and CVS [1] to list a few. Typically, users need to create versions manually in these systems. There are also copy-on-write versioning systems exemplified (COW) by Tops-20 [20] and VMS [17] that have automatic versions for some file operations. Elephant [31] transparently creates a new version of a file on the first write to an open file. CVFS [34] versions each individual write or small metadata using highly efficient data structures. OceanStore [28] uses versioning not only for data recovery but also for simplifying many issues with caching and replications. The LBFS [22] file system exploits similarities between files and versions of the same files to save network bandwidth for a file system on low-bandwidth networks. Peterson and Burns have implemented the ext3cow file system that brings snapshot and file versioning to the open-source community [26]. Muniswamy-Reddy et al. [21] presented a lightweight user-oriented versioning file system called Versionfs that supports various storage policies configured by users. ST-CDP provides block level CDP as opposed to file system level as done by file versions that are file system dependent.

2.3 Traditional CDP

To provide timely recovery to any point-in-time at the block device level, one can keep a log of changed data for each data block in a time sequence [5] referred to as CDP. 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 [13]. Zhu and Chiueh proposed a user-level CDP architecture that is both efficient and portable [43]. 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 CDP on commodity ATA disk and Gigabit Ethernet technologies [16]. 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.

The main drawback of the CDP storage is the huge amount of storage space required, which has thus far prevented it from being widely adopted. Flouris and Bilas [6] proposed a storage architecture named Clotho providing transparent data versioning at the block level. Clotho coalesces the updates that take place within a period of time by creating new versions for them. This versioning happens at discrete time points that are not necessarily continuous as is done in CDP. Morrey III and Grunwald [18], [19] observed that for some workloads, a large fraction of disk sectors to be written contain identical content to previously written sectors within or across volumes. By maintaining information (128 bit content summary hash) about the contents of individual sectors, duplicate writes are avoided. Zhu, Li, and Patterson [42] proposed an efficient storage architecture that identifies previously stored data segments to conserve storage space. These data reduction techniques generally require a search in the storage for an identical data block before a write is performed. ST-CDP does not need to search storage for the purpose of deduplication. Rather, it exploits the data locality property to insert logs of parities between snapshots.

It should be noted that keeping a log of changed data has been studied and used in other contexts other than data recovery. For example, log-structured file system has been researched extensively for improving disk write performance [29], [32]. There are variations of such log-structured file system such the DCD [8] proposed by Hu and Yang, and the Vagabond [23] proposed by Norvag and Bratbergsengen for optimizing disk write performance. Norvag and Bratbergsengen also noticed the space savings of storing a delta object in buffer space when an object is changed, which suggests data locality that exists in data write operations. The difference between ST-CDP and the existing log structured file system is that ST-CDP aims at high-data recoverability at a low cost as opposed to purely improving storage performance.

3 ST-CDP Architecture

In this section, we discuss the overall structure of ST-CDP system. For the purpose of completeness, we first summarize the original TRAP architecture followed by the complete ST-CDP architecture.

3.1 Brief Review of TRAP Architecture

As presented in [41], TRAP keeps a log of parities as a result of each write on a block. Fig. 1 shows the basic design of TRAP. Suppose that at time t_k, the host writes into a data block with logic block address A_i that belongs to

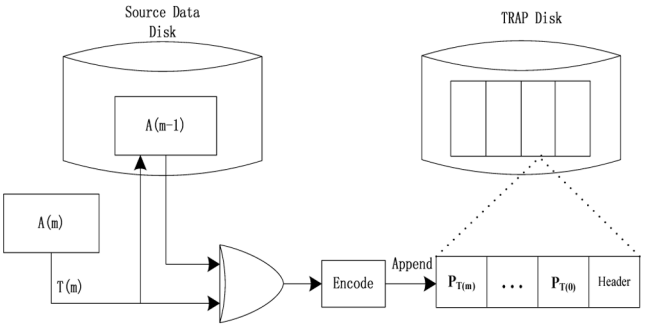


Fig. 1. Data logging method of TRAP design.

a data stripe $(A_1, A_2, \ldots, A_i, \ldots, A_n)$. The RAID controller performs the following operation to update its parity disk:

$$P_{T(k)} = A_i(k) \oplus A_i(k-1) \oplus P_{T(k-1)}, \quad (1)$$

where $T(K)$ is the time stamp, $P_{T(k)}$ is the new parity for the corresponding stripe, $A_i(k)$ is the new data for data block A_i, $A_i(k-1)$ is the old data of data block A_i, and $P_{T(k-1)}$ is the old parity of the stripe. Leveraging this computation, TRAP appends the first part of the above equation, i.e., $P'_{T(k)} = A_i(k) \oplus A_i(k-1)$, to the parity log stored in the TRAP disk after a simple encoding box, as shown in Fig. 1.

Now consider the parity log corresponding to a data block, A_i, after a series of write operations. The log contains $(P'_{T(k)}, P'_{T(k-1)\ldots\ldots}, P'_{T(2)}, P'_{T(1)})$ with time stamps $T(k), T(k-1), \ldots\ldots, T(2)$, and $T(1)$ associated with the parities. Suppose that an outage occurred at time t_1, and we would like to recover data to the image as it was at time $t_0 (t_0 \leq t_1)$. To do such a recovery, for each data block A_i, we first find the largest $T(r)$ in the corresponding parity log such that $T(r) \leq t_0$. We then perform the following computation:

$$A_i(r) = P'_{T(r)} \oplus P'_{T(r-1)} \oplus \cdots \oplus P'_{T(1)} \oplus A_i(0), \quad (2)$$

where $A_i(r)$ denotes the data image of A_i at time $T(r)$ and $A_i(0)$ denotes the data image of A_i at time $T(0)$. Note

$$P'_{T(l)} \oplus A_i(l-1) = A_i(l) \oplus A_i(l-1) \oplus A_i(l-1) = A_i(l)$$

for all $l = 1, 2, \ldots, r$. Therefore, (2) gives $A_i(r)$ correctly assuming that the original data image, $A_i(0)$, exists in a full backup storage.

The above process represents a typical redo recovery process upon an outage that results in data loss or data damage while earlier data are available in a full backup or a mirror storage. An undo process is also possible with the parity log if the newest data is available by doing the following computation instead of (2):

$$A_i(r) = A_i(k) \oplus P'_{T(k)} \oplus P'_{T(k-1)} \oplus \cdots \oplus P'_{T(r+1)},$$

where $A_i(k)$ represents the latest or newest data of block A_i.

3.2 Inserting Snapshots in Parity Logs

The TRAP architecture discussed above provides a CDP function by means of parity logs resulting from block write operations. Since every change is kept in the log, one can go back to any point-in-time. The traditional snapshot/backup,

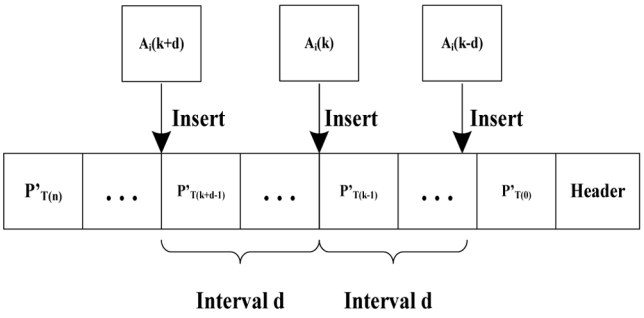


Fig. 2. The overall data structure of ST-CDP backup data.

on the other hand, provides periodic data images of block level storage. When data recovery is necessary, these two data protection techniques work quite differently. TRAP needs to retrieve the parity log for each data block and perform the parity computation to recover the data block corresponding to the recovery time point. As the parity log gets longer, so does the recovery time because of longer parity computations. Snapshot, on the other hand, just needs to restore the corresponding data blocks corresponding to the recovery time point, though the number of possible recovery points is limited by the frequency of snapshots performed.

ST-CDP takes a hybrid approach by inserting snapshots periodically into the parity logs. Snapshot is a common technique used in data storages for data protection and recovery. There are two types of snapshots: clone/full copy snapshot and copy-on-write differential snapshot. A clone is a full copy of the original data on a volume at a point-in-time that can be created through either software or hardware mirroring. Copy-On-Write differential snapshot makes a copy of the original data before it is overwritten when a change to the original volume occurs. The block about to be modified is read and then written to a "differences area," which preserves a copy of the data block before it is overwritten with the change. Using the blocks in the differences area and unchanged blocks in the original volume, a volume can be logically constructed that represents the snapshot copy at the point-in-time in which it was created. Fig. 2 shows the new parity logging structure. As we insert snapshots in the parity logs, sublogs are formed that are separated by periodic snapshots. At recovery time, only one sublog that contains the recovery time point is needed. The recovery time for each data block is limited by the half of the sublog length because parity computation can be done both ways, redo and undo, as shown in [41]. To minimize log retrieval time, one can also organize all sublogs in an efficient data structure as will be discussed shortly.

There are two important design issues that need to be considered carefully to implement the ST-CDP architecture. These issues have to do with whether we keep CDP data and insert snapshots in parity logs based on LBA address of data blocks (Space) or based on time of updates (Time). Each has its pros and cons.

Let us first consider that we keep our CDP data and insert snapshots based on space. That is, we keep a parity log for each distinct LBA and insert a snapshot data block after a certain number of write operations has been done on the block. In this way, we can ensure that all parity sublogs are limited to a predetermined length and hence control the recovery time. In addition, the recovery algorithm is relatively simple since each block has its corresponding CDP data. There are several disadvantages of space-based design. The first is its performance impact on applications. We need to keep track of CDP data corresponding to each data block, which takes not only more I/O operations on each write but also more metadata to keep track of CDP data of each block. For example, to determine when to insert snapshots, a counter is needed to keep track of the number of writes for each block. Furthermore, if write operations are not uniformly distributed across data blocks, we may end up with long parity logs and more snapshots for some blocks and short parity logs and fewer snapshots for others.

Now consider the time option, which keeps CDP data purely based on time of updates. That is, we do not distinguish different blocks but insert snapshots based on time when the total amount of write I/Os reaches a predetermined value. In this way, we do not need to keep track of write counts on each data block but only the aggregated total write I/Os. All CDP logs are contained in only one file, which makes management easier. Snapshots are done on the entire volume as opposed to individual data blocks. At recovery time, there is no need to go through every LBA to retrieve history data but only the aggregated CDP log. Furthermore, implementation is relatively simple since TRAP and snapshot can be done separately. The shortcoming of this approach is uncontrollable length of parity logs when write I/Os are unevenly distributed among data blocks. Some parity logs may be much longer than other parity logs depending on the write frequency to individual LBAs.

Clearly, both approaches discussed above have advantages and disadvantages. To make a wise design decision, a quantitative evaluation is necessary. In the next section, we will develop an optimization model to guide our design.

4 An Analytical Model

In this section, we will give a simple analytical model to optimize our ST-CDP design. Our objective is to determine when and how to insert snapshots in parity logs so that the design is space and recovery time optimal.

Let d be the length of each sublog in terms of the number of parity blocks enclosed by two subsequent snapshots. We would like to determine what d value one should choose for optimal implantation of ST-CDP on Linux operating system. In order to provide a quantitative guidance on how to choose d, let us define a set of symbols as shown in Table 1.

If we do not break up parity logs, the recovery time of each data block is given by

$$(T_{dec} + T_{xor} + S_{log}/IO_{rate}) * W_{avg} * T_{span}. \quad (3)$$

Now consider our ST-CDP design with sublogs of d parity blocks. As mentioned previously, the recovery time for each data block is limited by the half of the sublog length, d, because parity computation can be done both ways, redo and undo. If we assume that the recovery time point is uniformly distributed among d points within a

TABLE 1
Definition of Symbols Used in Analysis

Symbols	Definition
$\boldsymbol{d}$	Sub-log (parity log) Length: the number of parity blocks in each sub-log
$\boldsymbol{IO_{rate}}$	IO throughput of the disk storage
$\boldsymbol{S_{blk}}$	Data block size
$\boldsymbol{S_{log}}$	Size of a compressed parity block
$\boldsymbol{C}$	Compression Ratio: C= S_{blk} / S_{log}
$\boldsymbol{T_{dec}}$	Decoding time
$\boldsymbol{T_{xor}}$	XOR operation time
$\boldsymbol{T_{spn}}$	RPO: time span between current time and recovery time
$\boldsymbol{W_{avg}}$	Average number of write operations per time unit
$\boldsymbol{W_{total}}$	Total number of write operations a day
$\boldsymbol{\beta}$	Importance factor of time cost perceived by user
$\boldsymbol{\gamma}$	Importance factor of space cost perceived by user
M_0	Optimal number of total writes between two subsequent snapshots
M_{max}	Maximum number of total writes allowed between two subsequent snapshots
M	Counter keeping the number of total writes
m	Counter keeping the number of distinct data blocks changed
N	Average number of blocks in a snapshot

sublog if the RPO falls into that log, the expected parity blocks needed to do XOR computations for the data recovery is given by

$$E(d) = \frac{1}{2}\sum_{k=0}^{\lceil \frac{d}{2} \rceil} k * P(x=k) + \frac{1}{2}\sum_{k=1+\lceil \frac{d}{2} \rceil}^{d} (d-k) * P(x=k) \approx \sum_{k=0}^{\lceil \frac{d}{2} \rceil} \frac{2k}{d} \approx \left\lceil \frac{d+1}{4} \right\rceil. \quad (4)$$

The recovery time of each data block in the case of ST-CDP is given by

$$\boldsymbol{T}(\boldsymbol{d}) = (\boldsymbol{T_{dec}} + \boldsymbol{T_{xor}} + \boldsymbol{S_{log}}/\boldsymbol{IO_{rate}}) * \boldsymbol{E}(d) + \boldsymbol{S_{blk}}/\boldsymbol{IO_{rate}}. \quad (5)$$

The first half of the above equation gives the parity computation time and the second half gives the data copy time. It is interesting to note that this recovery time is independent of RPO but dependent on $\boldsymbol{d}$ value. The recovery time increases as $\boldsymbol{d}$ increases.

Let us now consider the additional storage space needed to store the parity logs and snapshot data while running ST-CDP. We would like to examine the average storage increase per time unit while running the ST-CDP, which is given by

$$\boldsymbol{S}(\boldsymbol{d}) = \boldsymbol{S_{log}} * \boldsymbol{W_{avg}} + \boldsymbol{S_{blk}} * \boldsymbol{W_{avg}}/\boldsymbol{d}, \quad (6)$$

where the first term gives the space for parity log and the second term gives the snapshot space. Since the number of snapshots inserted is inversely proportional to $\boldsymbol{d}$, the space usage of snapshots is also inversely proportional to $\boldsymbol{d}$.

Ideally, we would like to use as little storage space as possible and recover data as quickly as possible to reduce the overall cost. We will use these two factors to determine how good a data protection technology is. We therefore use the product of these two cost factors as the compound cost of ST-CDP. Let

$$\begin{aligned} \boldsymbol{F}(\boldsymbol{d}) &= \boldsymbol{T}(\boldsymbol{d}) * \boldsymbol{S}(\boldsymbol{d}) \\ &= [(\boldsymbol{T_{dec}} + \boldsymbol{T_{xor}} + \boldsymbol{S_{log}}/\boldsymbol{IO_{rate}}) * \boldsymbol{E}(\boldsymbol{d}) + \boldsymbol{S_{blk}}/\boldsymbol{IO_{rate}}] * \\ &\quad (S_{log} * \boldsymbol{W_{avg}} + \boldsymbol{S_{blk}} * \boldsymbol{W_{avg}}/\boldsymbol{d}) \\ &= (c_1 * \boldsymbol{E}(d) + \boldsymbol{c_2}) * (\boldsymbol{c_3} + \boldsymbol{c_4}/\boldsymbol{d}) \\ &= (c_1 * \boldsymbol{d}/4 + \boldsymbol{c_1}/4 + \boldsymbol{c_2})(\boldsymbol{c_3} + \boldsymbol{c_4}/d), \end{aligned} \quad (7)$$

where $\boldsymbol{c_1} = (\boldsymbol{T_{dec}} + \boldsymbol{T_{xor}} + \boldsymbol{S_{log}}/\boldsymbol{IO_{rate}}), \boldsymbol{c_2} = \boldsymbol{S_{blk}}/\boldsymbol{IO_{rate}}, \boldsymbol{c_3} = \boldsymbol{S_{log}} * \boldsymbol{W_{avg}}$, and $\boldsymbol{c_4} = \boldsymbol{S_{blk}} * \boldsymbol{W_{avg}}$. These are constants independent of $\boldsymbol{d}$.

Now, let us consider the derivative of $F(d)$ and set it to 0. We have

$$\boldsymbol{F}'(\boldsymbol{d}) = \boldsymbol{0} \rightarrow \boldsymbol{d_0} = \sqrt{\frac{(c_1 + 4c_2)c_4}{c_1 c_3}}. \quad (8)$$

Since the second derivative

$$\boldsymbol{F}''(\boldsymbol{d}) = (\boldsymbol{c_1} + 4\boldsymbol{c}2)\boldsymbol{C_4} * \boldsymbol{d}^{-3}/\boldsymbol{2}, F''(\boldsymbol{d} = \boldsymbol{d_0}) = \boldsymbol{c_1 c_3}/2\boldsymbol{d_0} > \boldsymbol{0},$$

the minimum value of $\boldsymbol{F}(d)$ exists when $\boldsymbol{d} = \boldsymbol{d_0}$. We will choose $\boldsymbol{d}$ to be the integer closest to $\boldsymbol{d_0}$ as our optimal sublog size.

While (8) gives the optimal $\boldsymbol{d}$ value, there are two important considerations in practical system designs as discussed below.

First of all, (8) gives the optimal subparity log sizes for each individual block. In a large storage system with millions of data blocks, keeping track of $\boldsymbol{d_0}$ for each data block may be excessively costly that may adversely impact application performance. In order to simplify system design, we introduce two additional parameters, $\boldsymbol{M}$ and $\boldsymbol{m}$. $\boldsymbol{M}$ is a counter that keeps track of total number of write operations performed in the storage while $\boldsymbol{m}$ keeps track of the number of distinct data blocks that have been written so far. $\boldsymbol{M}$ differs from $\boldsymbol{m}$ in the sense that when a specific data block is written $\boldsymbol{n}$ times, $\boldsymbol{M}$ is increased by $\boldsymbol{n}$ while $\boldsymbol{m}$ is incremented by one. We would like to consider the entire storage system and determine what would be the optimal value of $\boldsymbol{M}$ to make snapshot.

The second consideration is that (8) assumes that both recovery time and space costs are equally important. In practice, different organizations have different priorities and may weigh recovery time and space cost differently. For example, the New York Stock Exchange would be willing to sacrifice a lot of space to improve recovery time while small businesses might prefer a lower cost solution that would take some extra time to recover lost data. To guide different users in choosing optimal $\boldsymbol{d}$, let us define the perceived importance factors of recovery time and storage space to be β and γ, respectively. These importance factors can be considered as the cost per second of RTO and the cost per Gbyte of extra storage needed to implement ST-CDP. We would like to take into account these two importance factors in our optimization process.

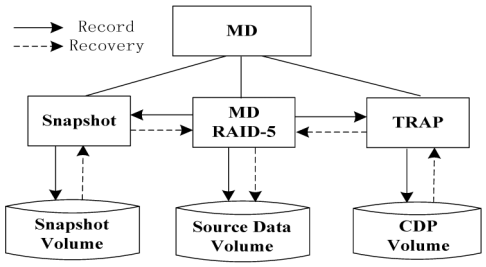


Fig. 3. The block diagram of the ST-CDP implementation in the Linux kernel.

Suppose that there are W_{total} write IOs during a day of operations. Taking into account all blocks in the storage system and the two importance factors, the total cost of the ST-CDP can be approximately expressed as

$$\begin{aligned}\mathbf{F}_{\text{cost}}(\mathbf{M}) = \beta\Bigg(\Bigg(\mathbf{T_{dec}} + \mathbf{T_{xor}} + \frac{\mathbf{S}_{\log}}{\mathbf{IO_{rate}}}\Bigg) * \mathbf{E}(\mathbf{M}) \\ + \frac{\mathbf{S_{blk}} * \mathbf{E}\left(\frac{\mathbf{W_{total}}}{\mathbf{M}}\right) * \mathbf{N}}{\mathbf{IO_{rate}}}\Bigg) \\ + \gamma\Bigg(\mathbf{S}_{\log} * \mathbf{W_{total}} + \mathbf{S_{blk}} * \frac{\mathbf{W_{total}}}{\mathbf{M}} * \mathbf{N}\Bigg),\end{aligned} \tag{9}$$

where $\mathbf{E}(\mathbf{M}) = \frac{\mathbf{M}+1}{4}$(see (4)), $\mathbf{E}(\frac{\mathbf{W_{total}}}{\mathbf{M}}) = \frac{1}{2} * \frac{\mathbf{W_{total}}}{\mathbf{M}}$. Let $\mathbf{e_1} = \mathbf{T_{dec}} + \mathbf{T_{xor}} + \frac{\mathbf{S}_{\log}}{\mathbf{IO_{rate}}}$, $\mathbf{e_2} = \frac{\mathbf{S_{blk}} * \mathbf{N}}{\mathbf{IO_{rate}}}$, $\mathbf{e_3} = \mathbf{S}_{\log 2} * \mathbf{W_{total}}$, and $\mathbf{e_4} = \mathbf{S_{blk}} * \mathbf{W_{total}} * \mathbf{N}$, we have

$$\mathbf{F}_{\text{cost}}(\mathbf{M}) = \beta\Bigg(\mathbf{e_1} * \frac{\mathbf{M}+\mathbf{1}}{\mathbf{4}} + \mathbf{e_2} * \frac{\mathbf{W_{total}}}{\mathbf{2M}}\Bigg) + \gamma\Big(\mathbf{e_3} + \frac{\mathbf{e_4}}{\mathbf{M}}\Big). \tag{10}$$

Deriving the derivative of $\mathbf{F}_{\text{cost}}(\mathbf{M})$ and set it to 0, we have

$$\mathbf{F}'_{\text{cost}}(\mathbf{M}) = \beta\Bigg(\frac{\mathbf{e}_1}{4} - \frac{\mathbf{e_2} * \mathbf{W}_{\text{total}}}{2}\mathbf{M}^{-2}\Bigg) - \gamma\mathbf{e_4}\mathbf{M}^{-2}, \tag{11}$$

$$\mathbf{F}'_{\text{cost}}(\mathbf{M}) = \mathbf{0} \rightarrow \mathbf{M_0} = \sqrt{\frac{\mathbf{2\beta e_2 W_{total}} + \mathbf{4\gamma e_4}}{\beta \mathbf{e_1}}}. \tag{12}$$

Because

$$\mathbf{F}''_{\text{cost}}(\mathbf{M}) = \beta * \mathbf{e_2} * \mathbf{W_{total}} * \mathbf{M^{-3}} + \mathbf{2\gamma e_4 M^{-3}} > \mathbf{0}, \tag{13}$$

optimal value of M, M_0, exists and is given by (12). Equation (12) will be used in ST-CDP system to determine when to make a system wide snapshot. After each snapshot operation, both M and m are reset to zero and start counting over again.

Given the limited system resources available, we also introduce a maximal allowable number of write operations before making a snapshot, M_{max}, to limit the usage of system RAM for SP-CDP. This maximum allowable writes can be determined based on the system resources available. For example, at recovery time, we need a large amount of system RAM to uncompress parity logs and compute XORs. Suppose the system RAM is 1 GB and the compression ratio is 5. If 60 percent of the RAM were allocated for recovery process, we could set M_{max} to be 100,000 if each uncompressed data block is 5 KB. A storage administrator can determine the best value of M_{max} based on the hardware resources available, work load characteristics, and RTO requirement. When M reaches either M_{max} or M_0, a snapshot operation is initiated followed by parity logging operations.

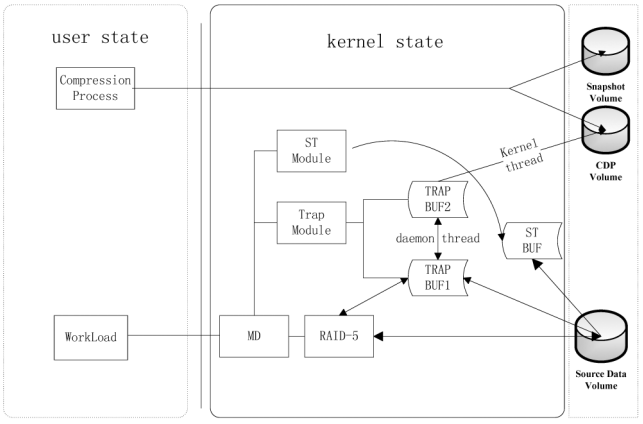


Fig. 4. Block diagram of the detailed implementation of ST-CDP with respect to existing kernel modules.

5 Prototype Implementation

5.1 Basic Architecture of ST-CDP

Based on the design and analysis presented in the previous sections, we implemented our ST-CDP as an added kernel module on RAID-5 in Linux kernel. As the block level is typically the greatest common denominator of heterogeneous enterprise applications, our ST-CDP is developed as a stand-alone block device driver independent of higher level systems and applications. Fig. 3 shows the location where our ST-CDP is implemented in the Linux kernel. It is implemented at the same level as the Linux software RAID, MD (Multiple Device). Two separate volumes, snapshot volume and CDP volume, are under the direct control of snapshot module and TRAP module, respectively, as shown in Fig. 3.

Fig. 4 shows a more detailed layout of the design of our ST-CDP. It has two major functional modules, ST-CDP logging module and recovery module.

The ST-CDP logging module keeps periodic COW-snapshots (copy-on-write differential copy) as well as a journal of parities between two consecutive snapshots. It bypasses all read requests and intercepts all write requests. The normal IO write operations and the operation of ST-CDP are done in parallel. There are two major parts in the ST-CDP logging module. The first part carries out COW snapshot operations when triggered, referred to as ST module. The other part, referred to as the TRAP module, does the parity computations and logging, as shown in Fig. 4.

The recovery module of the ST-CDP is a program that runs only after normal write I/Os stop. When data recovery needs to be done, the recovery module starts by retrieving snapshot data and parity logs. The recovery process is shown in Fig. 5.

At first, the recovery module will search the metadata of snapshots and recover to the nearest version of the snapshot. And then the recovery module continues to recover to the right point as designated by a user. Based on the designated

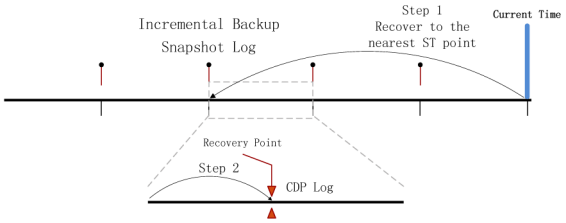


Fig. 5. Timing diagram showing recovery process.

RPO, it searches the parity logs for each data block to find the sublog that contains the desired RPO. Once such a sublog is found, the recovery program searches for a parity block that has the timestamp matching or being the closest to the RPO. Exclusive-OR operations are then performed to recover the right data block. After all changed data blocks are recovered, the data will be written to the source volume and the recovery process is done. It is also possible that the RPO matches one of the snapshots in the CDP volume. In this case, no parity computation is necessary. The recovery program just copies the snapshot data to the source volume.

5.2 Implementation Details and Data Structures

5.2.1 ST-CDP Logging Module

As mentioned previously, the first part of ST-CDP logging module is the snapshot, ST module. There are two types of snapshots for block level data storages. One is a full snapshot (cloning) and the other is copy-on-write (COW differential copy) [30]. Our prototype ST module uses the second type of snapshot, COW snapshot, which makes a copy of the original block upon a write operation and the copy is stored in the snapshot volume. Together with the original data, the COW differential copy in the snapshot volume forms the point-in-time snapshot of the entire volume. The exact point-in-time is determined dynamically at runtime based on the chosen value of M discussed in the last section as well as predefined time limit.

When MD is started in the Linux kernel, ST-CDP will start a ST-CDP Kernel daemon thread that intercepts all write requests except for resync and recovery. It does this in the core function of RAID-5-handle_stripe5(). After it catches a write request (write bio), two actions will be performed. The first action is to mark a time stamp on the bio and the second action is to generate a read bio on the same block address and send it to disk before write. Since the write requests which have been sent to disk cannot

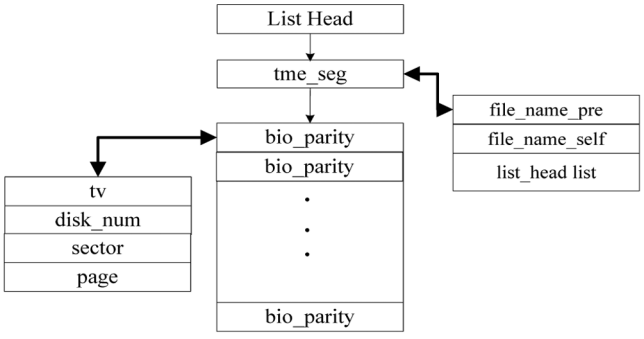


Fig. 6. The kernel list buffer structure.

TABLE 2
Data Structure of **Bio_Parity**

data structure	meaning	member
List Head	Head of the list buffer	atomic_t v; /*list counter*/ spinlock_t lock; /*buffer lock*/
time_seg	Some information of the buffer	char file_name_pre[18]; /*pre-file name*/ char file_name_self[18]; /*file name self */ struct list_head list; //list head point
bio_parity	Useful data of each I/O operation.	struct timeval tv; /*time stamp */ int disk_num; /*index num of disk */ sector_t sector; /*index of sector */ void * page; /*data after xor*/ struct list_head list; /*list point*/

return in the same order, the 4-tuple information entry, (sector, disk number, time stamp, data to be written), on the bio will be stored in a data structure named mdtrap_bios_need_save in the kernel list named list_will_save. After the read bio returns to its callback function (interrupt context), it will be removed from the list_will_save and stored in a buffer. When the buffer space reaches a threshold, the Kernel daemon thread will be woken up to start a new Kernel thread which takes the buffer to ST-TRAP record module (process context).

In the ST-TRAP record module, the nodes in the buffer will be sent to mdtrap_save() function one by one. The mdtrap_save() function copies the information and read data (from read bio) of each node to a hash table node called st_hlist_node. When inserting st_hlist_node to the hash table, the counter M mentioned above will add 1. If there is no collision in the hash table (which means a new block access), m is incremented. When M reaches M_{max} or M_0, the snapshot module of ST-CDP starts to store COW snapshot to the snapshot volume with exactly one value for each LBA representing the COW differential snapshot. At the same time the information of the snapshot version is stored in the Snapshot Metadata.

After the snapshot process, XOR will be performed between the data of write bio and read bio that was read before write. The XOR results are stored in a new data structure called bio_parity. It also contains the necessary information of bio. It will be inserted in a kernel list buffer in time order as shown in Fig. 6. The data structure of bio_parity is shown in Table 2. When the buffer above reaches a certain value, TRAP module will store it to the CDP Volume. It is done by starting a new kernel thread. Similar to the snapshot module, metadata of the CDP buffer is stored in the CDP metadata.

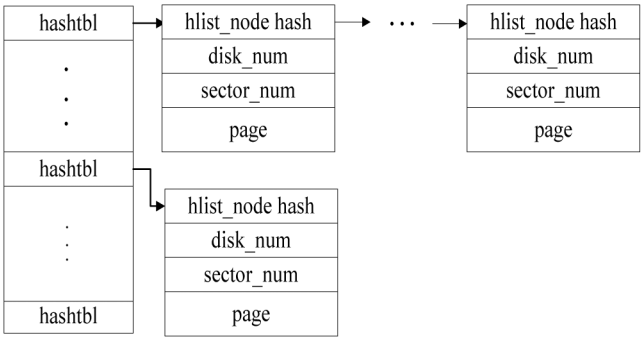


Fig. 7. Structure of hash tables.

5.2.2 ST-CDP Recovery Module

In order to ensure consistency of data, the normal write process of RAID-5 must be stopped and all the buffer data must be flushed to disk. To start a recovery process, the user inputs a recovery time point, RPO. The metadata of snapshots and TRAP will be processed according to the time point to determine the total amount of snapshots and parity log data that will be processed for recovery. Then, all the necessary information will be sent to ST-CDP in the Linux kernel.

The recovery module will first recover to the nearest snapshot version from the data in the snapshot volume. Next, the recovery module will recover to the exact RPO point by tracing the parity logs in the CDP volume.

During this recovery process, a hash table is used to reduce disk I/O operations. When recovering to the nearest snapshot version, the same bio (at the same sector in the same disk) data will be replaced. When recovering to the exact RPO point through the nearest snapshot version, the same bio (at the same sector in the same disk) data will be replaced after XOR computations. The data structures are shown in Fig. 7 and Table 2. During the recovery process, if any of the two hash tables reaches the upper limit of memory usage, it will be flushed to the RAID-5. After that, it continues the recovery process until it has recovered to the right RPO point.

6 Experimental Settings

For the purpose of testing our ST-CDP implementation and performance evaluation, we have carried out extensive experiments. Fig. 8 shows the high-level block diagram of our experimental settings. To allow multiple clients and multiple storage servers in a networked environment, we implemented the lower level storage device using iSCSI protocol as shown in Fig. 8. Our ST-CDP module runs on the storage server at the block device level of the Linux operating system. The client machine has file system, database, and application benchmarks installed. The details of the hardware and software environment in our experiments are shown in Table 3.

Right workloads are important for performance studies [8]. In order to have an accurate evaluation, we use real world I/O workloads and standard benchmarks. The first benchmark, TPC-C, is a well-known benchmark used to model the operational end of businesses where real-time transactions are processed [3]. TPC-C simulates the execution of a set of distributed and online transactions (OLTP)

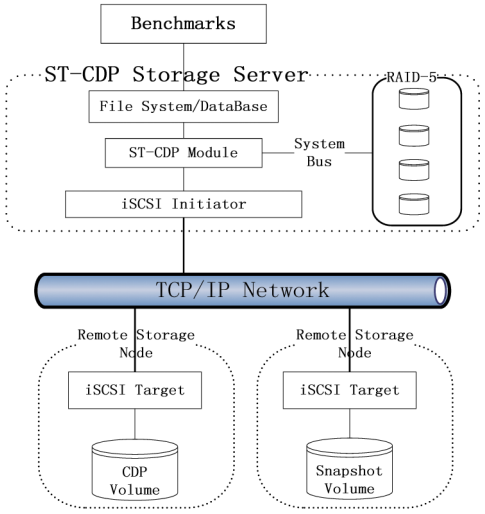


Fig. 8. Experimental setup.

for a period of two to eight hours. It is set in the context of a wholesale supplier operating on a number of warehouses and their associated sales districts. TPC-C incorporates fivetypes of transactions with different complexities for online and deferred execution on a database systems. These transactions perform the basic operations on databases such as inserts, deletes, and updates. From a data storage point of view, these transactions will generate reads and writes that will change data blocks on disks. For Postgres Database, we use the implementation from TPCC-UVA [27]. Eight warehouses with 25 users are built on Postgres database. Details regarding TPC-C workloads specification can be found in [3].

Besides benchmarks running on databases, we have also run two file system benchmarks IoMeter and PostMark. IoMeter is a flexible and configurable benchmark tool that is also widely used in industries and the research community[9]. It can be used to measure the performance of a mounted file system or a block device. We run the IoMeter on NTFS with 4 KB block size for two types of workloads: 100 percent random writes, and 30 percent writes and 70 percent reads. PostMark is another widely used file system benchmark tool written by Network Appliance, Inc. [10]. It measures performance in terms of transaction rates in an ephemeral small-file environment by creating a large pool of continually changing files. Once the pool has been

TABLE 3
Hardware and Software Configurations in the Experiments

	Remote Storage Nodes	ST-CDP Storage Server
CPU	Inter Celeron-D 2.80GHz	Inter Celeron-D 2.80GHz
RAM	DDR2 533, 512MB	DDR2 533, 2GB
Disk	SATA 300GB	SATA 300GB * 5
OS	Gentoo Linux (kernel 2.6.22)	Gentoo Linux (kernel 2.6.22)
Switch	Cisco 3750-E Gb	Cisco 3750-E Gb

TABLE 4
Benchmarks Used in Performance Evaluations

Benchmark	Brief description
tar	Run 8 times randomly on ext3
gcc	Compile ST-CDP source code on ext3
zip	Compress 300MB misc data on ext3
cp/rm/mv	Execute basic file operations (cp, rm and mv) on ext3
PostMark	Run PostMark, 10,000 files, 20,000 transactions
IoMeter	Run IoMeter 10 minutes， 30% write,100% random, 4KB, 8kB, 16KB, 32KB, 64KB, 2 minutes for each case

created, a specified number of transactions occur. Each transaction consists of a pair of smaller transactions, i.e., Create file/Delete file and Read file/Append file. Each transaction's type and files it affects are chosen randomly. The read and write block size can be tuned. In our experiments, we set PostMark workload to include 10,000 files and to perform 20,000 transactions. Read and Write block sizes are set to 4 KB.

SPC-1 [35] is a synthetic storage subsystem performance benchmark of Storage Performance Council. It works by setting the storage subsystem to an I/O workload designed to mimic realistic workloads that are collected in typical business critical applications such as OLTP systems and mail server applications. SPC-1 has gained some industry acceptance and storage vendors such as Sun, IBM, HP, Dell, LSI-Logic, Fujitsu, StorageTek, and 3PARData among others have submitted results for their storage controllers [36]. SPC Financial is a well-known block level disk I/O trace taken from online transaction processing applications running at large financial institutions. We use the first 1 million lines of SPC Financial1 [38] for our performance evaluation and comparison. It has very high-write ratio (above 70 percent). We chose different numbers of concurrent processes and different block sizes in our experimental evaluations.

We have also used some microbenchmarks reflecting users' experiences. All the microbenchmarks are the shell commands that are often used by users such as gcc, tar, cp, mv, and so on as shown in Table 4.

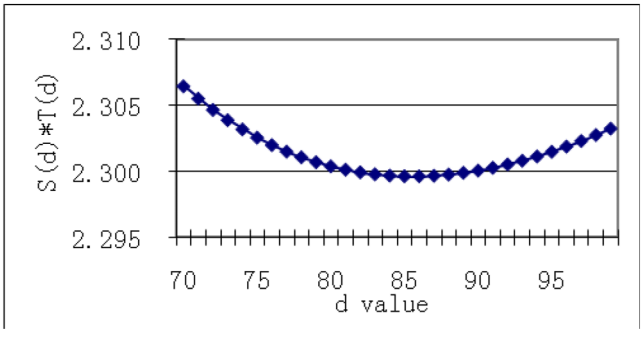


Fig. 9. ST-CDP cost versus sublog size, d, for block size of 4 KB. The optimal value of d is 77.

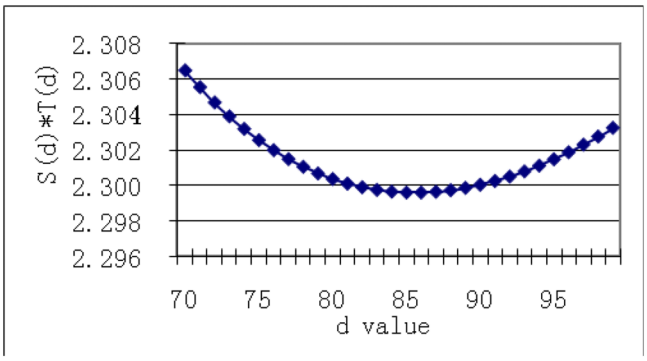


Fig. 10. ST-CDP cost versus sublog size, d, for block size of 16 KB. The optimal value of d is 86.

7 Numerical Results and Discussions

In this section, we present our measurement results in terms of space usage, recovery time, and runtime performance impact of ST-CDP. We compare the performance results of three data protection techniques: namely native TRAP with no snapshots, ST-CDP, and pure CDP that keeps all changes of a data block. The CDP we evaluate here is copy-on-write CDP, COW for short, as opposed to redirect-on-write [40].

Before starting our experiments, we first use our analytical model presented in Section 4 to select appropriate values of d for optimal performance. For this purpose, we calculate the compound function of storage space, $S(d)$, and recovery time, $T(d)$, for three different block sizes as shown in Fig. 9, Fig. 10, and Fig. 11. It can be seen from these figures that optimal d values do exist. The optimal value for block size of 4 KB happens at $d = 77$. And similarly, the optimal values of block sizes of 16 and 64 KB happen at $d = 86$ and $d = 94$, respectively. It is interesting to note that as block size increases, the d value increases. This phenomenon can be attributed to the fact that large block sizes result in more storage space requirement for storing snapshots. Bigger snapshots need more space to store and more time to retrieve during data recovery time. The parity logs between two snapshots do not increase space requirement as fast because of content locality [41]. As far as recovery time is concerned, one may argue that larger block size and d value will increase the parity computation time during the recovery process. However, such additional computation time is comparable to or less than IO time of retrieving large blocks from snapshots. Based on the analysis, we set the values of d in our

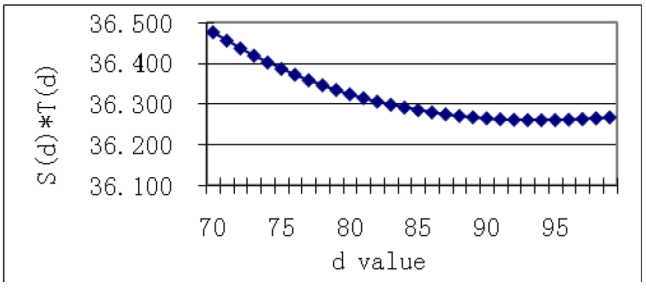


Fig. 11. ST-CDP cost versus sublog size, d, for block size of 64 KB. The optimal value of d is 94.

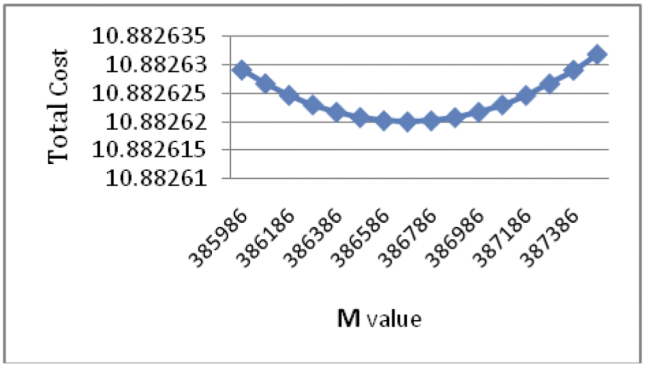
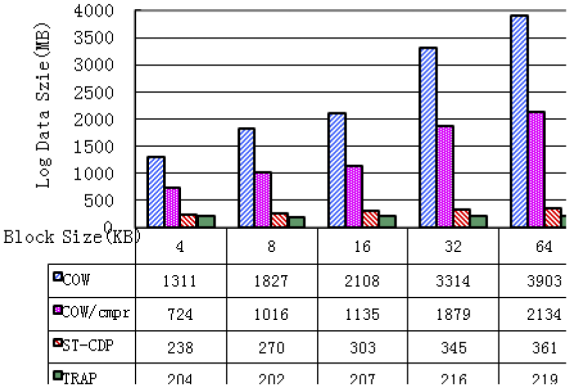


Fig. 12. Total cost versus sublog size, M, for block size being 4 KB and it is assumed that the time cost is \$1K per hour and space cost is \$10 per gigabyte. The optimal value of M is 227,629.



Block Size(KB)	4	8	16	32	64
COW	1311	1827	2108	3314	3903
COW/cmpr	724	1016	1135	1879	2134
ST-CDP	238	270	303	345	361
TRAP	204	202	207	216	219

Fig. 14. CDP space overhead comparison while running TPC-C benchmark on Postgres database.

experiments to be 71, 79, 85, 91, and 94 corresponding to block sizes of 4, 8, 16, 32, and 64 KB, respectively.

As discussed in the previous section, different organizations may choose snapshot frequencies differently depending on their priority and relative importance of recovery time and space cost. Furthermore, it may be practically convenient to use the optimal M value (12) to determine snapshot times. Using (10), we plotted ST-CDP cost as a function of M considering different cost factors as shown in Fig. 12 and Fig. 13. Fig. 12 assumes a lower down time cost appropriate for small to medium size businesses and Fig. 13 assumes a higher downtime cost appropriate for high-end systems. From these two figures, we can see that if an organization can afford to spend some time to do data recovery in case of a failure, one can do snapshot less frequently with large M value (Fig. 12). On the other hand, if downtime cannot be tolerated meaning short RTO is essential to the business, one can choose relatively small M value implying that more and frequent snapshots should be done to speedup recovery time. This is consistent with our early analysis and expectations.

Based on the d values discussed above, our first experiment is to measure the additional space usage of the three data protection techniques. Fig. 14 shows the measured results when we ran TPC-C benchmark on Postgress database. We plotted the space usage of the three data protection technologies for different block sizes ranging from 4 KB through 64 KB. For COW CDP, we measured the space usage of both uncompressed and compressed CDP logs. It can be seen from this figure that COW CDP takes most space because it keeps the original data blocks of all changed data. Even data compression is applied to the CDP logs, the space usage is still fairly large. Native TRAP takes the least amount of space because of locality property of write operations as evidenced in [41]. The space usage of ST-CDP is somewhere in between native TRAP and compressed COW CDP because it stores both parity logs and small amount of snapshots between sublogs. We choose the optimal value of d for each block size, resulting in 4, 5, 6, 9, and 10 snapshots for block sizes of 4, 8, 16, 32, and 64 KB, respectively, for each benchmark run. Subparity logs are kept between these snapshots. The space overhead of ST-CDP is closer to that of TRAP than that of COW CDP. Our observation is that ST-CDP provides CDP with substantially less storage overhead than continuous real-time snapshots.

The space overheads of the CDP solutions for IOMeter, Postmark, and microbenchmarks are shown in Figs. 15 and 16 for block size of 8 KB. Because the micro benchmarks (tar, gcc, and zip) took between 25 seconds and 2.2 minutes to run, only one snapshot was taken for each benchmark run. For cp&rm&mv, PostMark, and IoMeter benchmarks, three, two, and six snapshots were taken, respectively, during the benchmark runs. It is observed that ST-CDP uses the amount of space for storing CDP data very close to the amount of space that TRAP uses for all the benchmarks shown in these two figures. This observation is further validated by

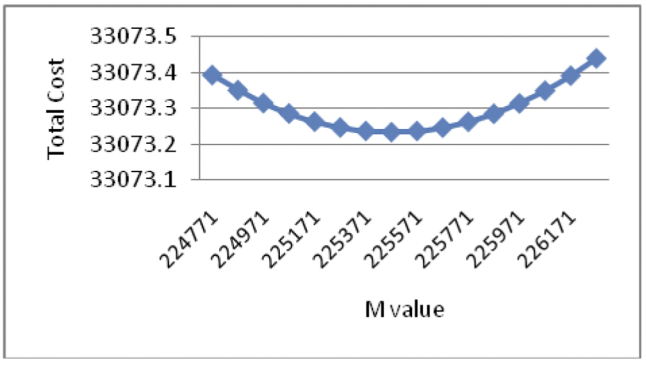


Fig. 13. Total cost versus sublog size, M, for block size being 4 KB and it is assumed that the time cost is \$10 million per hour and space cost is \$10 per gigabyte. The optimal value of M is 225,471.

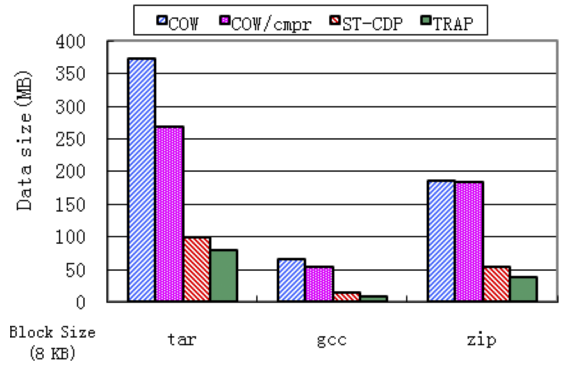


Fig. 15. CDP space overhead comparison while running micro-benchmarks for block size 8 KB.

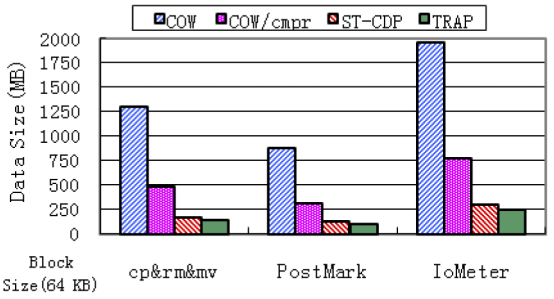


Fig. 16. CDP space overhead comparison while running microbenchmark, PostMark, and IOMeter for block size 8 KB.

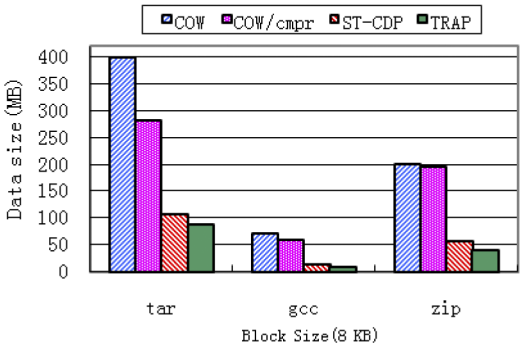


Fig. 17. CDP space overhead comparison while running microbenchmarks for block size 64 KB.

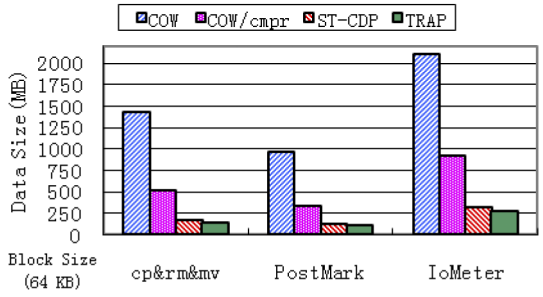


Fig. 18. CDP space overhead comparison while running microbenchmark, PostMark, and IOMeter for block size 64 KB.

changing the block size as shown in Figs. 17 and 18 that show the similar results for block size of 64 KB. Note that while one snapshot was taken for each of tar, gcc, and zip, five, four, and eight snapshots were taken for cp&rm&mv, PostMark, and IoMeter, respectively. We therefore can conclude that ST-CDP has the space overhead similar to TRAP but at the same time provides higher reliability and faster recovery as will be evidenced next.

Having observed the space efficiency of the new ST-CDP, our next experiment is to measure and compare the recovery times of the CDP solutions. In the storage industry, the recovery time is often referred to as RTO, recovery time objective. Businesses demand quick recovery or fast RTO because smaller RTO means shorter down time of business operations. Therefore, RTO is a very important performance parameter for data protection technologies. In our experiments, we used the storage server with RAID-5 consisting of four disks with capacity of 300 GB as shown in Fig. 8. The storage server is shared by a dozen clients in an office environment that run typical office applications such as coding, word processing, simple databases, mails, and web. After running for an entire business day, we observed

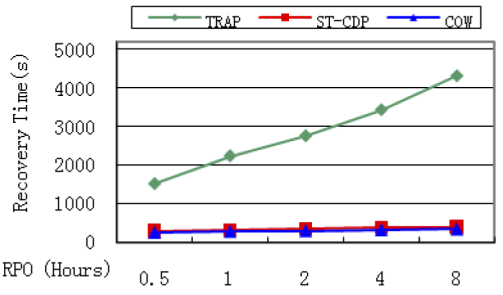


Fig. 19. Recovery time as a function of recovery point for block size 64 KB.

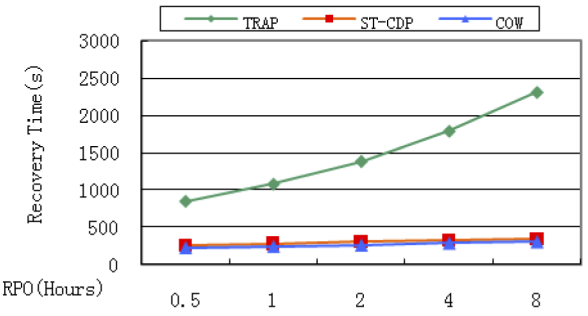


Fig. 20. Recovery time as a function of recovery point for block size 8 KB.

about 5 GB changed data. We then try to recover data to different RPOs of the day. We measured the recovery times of native TRAP, COW CDP, and ST-CDP and compared their respective recovery times.

Fig. 19 shows the measured recovery time as a function of RPO for a block size of 64 KB. As can be seen from this figure, TRAP's recovery time increases as RPO increases because of XOR computation of long parity logs. On the other hand, the recovery time of ST-CDP keeps flat while RPO changes. For example, to recover data to a half hour ago, the native TRAP recovery program takes about 850 seconds. To recover data to 8 hours ago, it takes about 2,315 seconds, about 3 times longer. With ST-CDP, on the other hand, it takes about 278 seconds to recover data to a half hour ago and it takes about 345 seconds to recover data to 8 hours ago, only about a 30 percent increase in RTO. This linear increase in recovery time is due to the fact that ST-CDP needs to walk back all incremental snapshots until the most recent full-clone snapshot although parity computation needs to be done on only one subparity log. However, the recovery time increase of ST-CDP is substantially smaller than the native TRAP mechanism. During the 8 hours experiments, ST-CDP took 22 snapshots with subparity logs inserted in between. We noticed that the COW CDP has a similar recovery time performance as that of ST-CDP. However, COW CDP takes much larger storage space than S-CDP, as demonstrated previously.

The disparity of recovery time between the native TRAP and ST-CDP is larger for smaller block sizes. Fig. 20 shows the recovery time comparison for a block size of 8 KB. It can be seen from this figure that it takes about 1,515 seconds to recover data to half an hour ago for native TRAP. To recover data to eight hours ago, it will take more than 4,320 seconds. Again, the recovery time of ST-CDP stays relative flat. As shown in Fig. 20, the recovery time varies from 296 seconds to 397 seconds. Note that ST-CDP took 16 snapshots during the eight hours experiments for this configuration. Smaller

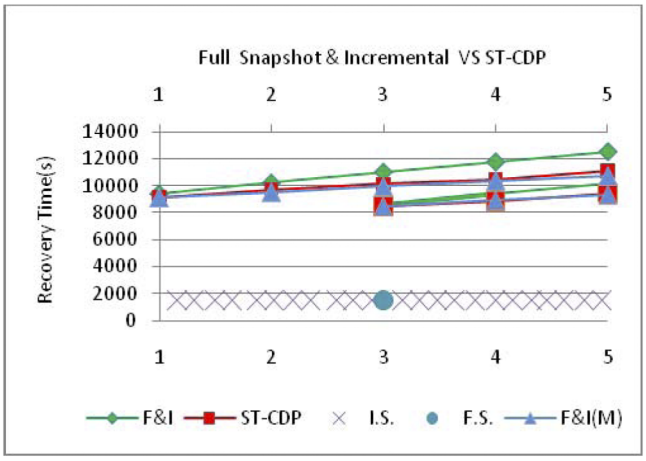


Fig. 21. Recovery time comparison among Full Snapshot+incremental, Full Snapshot+incremental+write-coalescing, and ST-CDP.

block sizes have space advantages but take longer time to perform the XOR function both at runtime and recovery time because of longer CDP logs for the same operation period.

In addition to the experiments in the real application environment, we have also measured RTO of SPC-1 traces using a full snapshot with incremental CDP in between (F&I) and compared it to our ST-CDP. Fig. 21 shows the measured RTO as a function of RPO for two experiments. The first experiment starts with a full snapshot followed by incremental CDP for F&I, and followed by normal ST-CDP operations with an incremental snapshot inserted at every 10-minute interval for ST-CDP (shown by X's in Fig. 21). The second experiment inserts an additional full snapshot at hour three of benchmark run. Both experiments ran for total five hours. As shown in Fig. 21, RTOs of both data protection schemes are very close immediately after the full snapshots were taken and linearly increase as RPO increase. As shown in Fig. 21, ST-CDP shows slightly better RTO than F&I. We believe this difference mainly results from the fact that ST-CDP implemented write coalescing during recovery process, as evidenced by additional experiments of implementing write coalescing in F&I as shown in blue line in Fig. 21.

In general, the recovery time of ST-CDP increases linearly with RPO after a full snapshot is made as shown in Fig. 21. In practical storage systems, a full snapshot is made daily or weekly depending on the data protection and recovery requirement. The objective of ST-CDP is to provide finer recovery granularity between these full snapshots at the cost of linearly increased RTO. We expect the length of ST-CDP chains to be less than a day of storage operations in a typical storage environment. As a result, the additional RTO of ST-CDP is expected to be manageable. For example, during two hours of operations shown in Fig. 21, the RTO of ST-CDP increases from 9,460 to 10,115 s compared to the RTO of F&I from 9,406 to 11,083 s. After F&I made a full snapshot at hour three, the RTO of F&I becomes 8,576 s at RPO of three and is increased to 10,129 s at RPO of five. The RTO of ST-CDP with no full snapshot at hour three, on the other hand, becomes 11,168 at RPO of five. The RTO difference of the two data recovery schemes at RPO of 5 is within 11 percent.

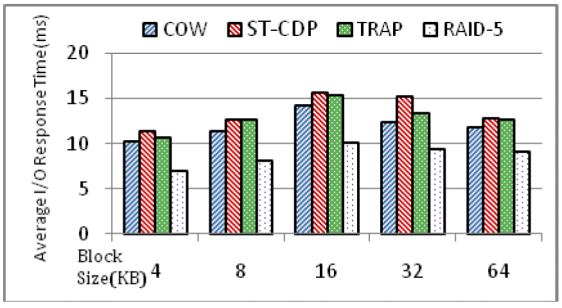


Fig. 22. IOMeter performance in terms of average I/O response time as a function of block sizes. 70 percent reads, 30 percent write, and 100 percent random accesses.

Now, let us consider the possible impact of the CDP solutions on the runtime application performance. Since ST-CDP carries out XOR computations and snapshot operations at runtime, an immediate question is how it impacts application performance. Our next experiment is to evaluate the performance impact of ST-CDP on applications. For this purpose, we run IOMeter to measure the IO performance while enabling the ST-CDP module. Fig. 22 shows our measured IOMeter results in terms of average I/O response time as functions of block sizes for 70 percent reads, 30 percent writes, 100 percent random I/Os. We plotted four performance bars corresponding to COW CDP, ST-CDP, TRAP, and RAID-5 alone with no data protection program running. Performance of RAID-5 is used as a reference for us to observe the negative impacts of the three data protection technologies. It is observed that all three data protection solutions have some degree of negative performance impacts. We noticed that snapshot COW CDP has the least performance impact and ST-CDP has the most. TRAP is in between but close to that of COW-CDP. However, the performance difference among the three techniques is very small and well under 10 percent.

Fig. 23 shows the performance impact of the three data protection techniques when we run SPC-1 traces. We varied the number of processes from 5 to 20 as shown in the figure. The performance impacts are much larger than what we have seen for IOMeter. But the three CDP solutions show similar performance behaviors. Similar observations are shown in Fig. 24 for Postmark performance. Postmark performance of ST-CDP is similar to that of TRAP but much slower than COW. It is clear from Fig. 24 that all the three data protection schemes have significant impact on IO performance. We

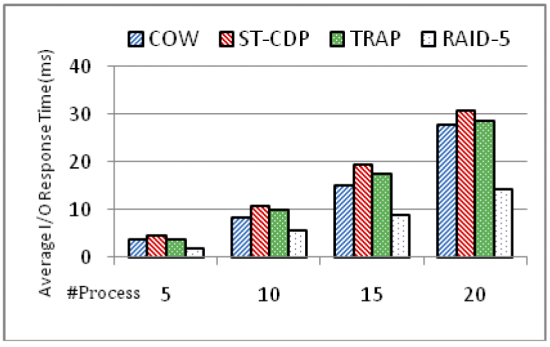


Fig. 23. SPC-1 performance in terms of average I/O response time as a function of number of processes.

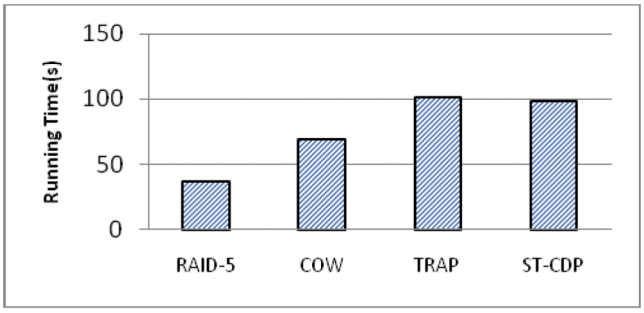


Fig. 24. PostMark performance in terms of time overhead of the three data protection techniques.

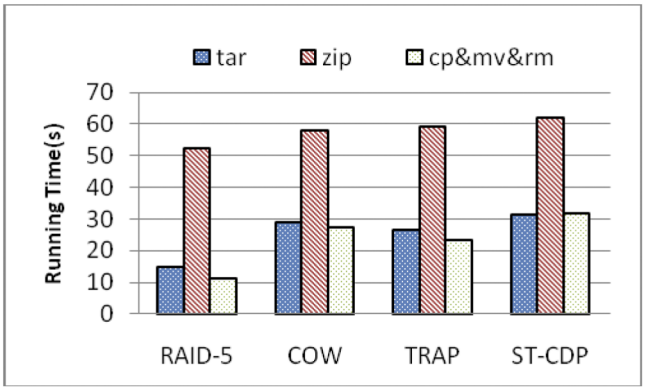


Fig. 25. Time overhead comparison of the three data protection techniques while running microbenchmark.

noticed that PostMark is a write intensive and mostly random workload that requires a lot of additional operations for the purpose of CDP. Similarly, we also observed negative performance impacts of all data protection techniques on I/O performance when running the microbenchmarks as shown in Fig. 25.

In summary, all the CDP solutions evaluated in our experiments have negative performance impact to some extent. However, the performance difference among the three is not very significant. Therefore, space overheads and recovery times are the parameters one should consider with more weight in choosing a solution. The negative performance impact can be minimized by offloading the data protection functions to lower level storage systems. Our ST-CDP can be easily implemented at storage device with embedded processors. This is one of our future research works.

8 Conclusions

In this paper, we have presented a new storage architecture capable of providing CDP, referred to as ST-CDP. It is based on the TRAP [41] technology that keeps logs of parities of changed data blocks and interspersed with snapshot data. A mathematical model has been developed to guide the optimization of the design. Based on the analytical model, we have designed and implemented ST-CDP in the Linux kernel. The implementation is done at the block device level as an independent device driver that can be added to MD software RAID device. Extensive experiments have been carried out to show that the implementation is fairly robust. Standard benchmarks are used to evaluate the performance and cost of the implementation. Numerical results have shown that the overhead is manageable. The major advantage of ST-CDP is low RTO that is RPO independent.

As a future work, we are fine tuning ST-CDP and studying various trade-offs between space costs and recovery times for different applications.

Acknowledgments

This research is sponsored in part by the Chinese 863 Plan under the Grant No. 2009AA01A402, the National Natural Science Foundation of China under Grant No.60933002 and NSFC60736013, the National Basic Research 973 Program of China under Grant No.2011CB302303, innovation fund of WNLO. Qing Yang's research is sponsored by US National Science Foundation (NSF) under grants CCF-0610538, CCF-0811333, and CCF-1017177. 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. The authors are very grateful to the anonymous reviewers for providing detailed comments that have improved the quality of the paper greatly. The authors appreciate Kerry Yang's help in proofreading the paper.

References

[1] B. Berliner and J. Polk, "Concurrent Versions System (CVS)," http://www.cvshome.org, 2001.
[2] A. Chervenak, V. Vellanki, and Z. Kurmas, "Protecting File Systems: A Survey of Backup Techniques," *Proc. Joint NASA and IEEE Mass Storage Conf.*, 1998.
[3] T.P.P. Council, "TPC BenchmarkTM C Standard Specification," http://www.tpc.org/tpcc, 2005.
[4] L.P. Cox, C.D. Murray, and B.D. Noble, "Pastiche: Making Backup Cheap and Easy," *Proc. the Fifth USENIX Symp. Operating System Design and Implementation*, 2002.
[5] J. Damoulakis, "Time to Say Goodbye to Backup?," *Storage*, vol. 4, no. 9, pp. 64-66, Nov. 2006.
[6] M. Flouris and A. Bilas, "Clotho: Transparent Data Versioning at the Block I/O Level," *Proc. the 12th NASA Goddard, 21st IEEE Conf. Mass Storage Systems and Technologies (MSST '04)*, pp. 315-328, 2004.
[7] D.K. Gifford, R.M. Needham, and M.D. Schroeder, "Cedar File System," *Comm. the ACM*, vol. 31, no. 3, pp. 288-298, Mar. 1988.
[8] Y. Hu and Q. Yang, "DCD—Disk Caching Disk: A New Approach for Boosting I/O Performance," *Proc. the 23rd Ann. Int'l Symp. Computer Architecture (ISCA)*, 1996.
[9] Intel, "IoMeter: Performance Analysis Tool," http://www.iometer.org, 2011.
[10] J. Katcher, "PostMark: A New File System Benchmark," Technical Report 3022, Network Appliance, 1997.
[11] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, "Designing for Disasters," *Proc. the Third USENIX Conf. File and Storage Technologies*, 2004.
[12] D. Korn and E. Krell, "The 3-D File System," *Proc. the USENIX Summer Conf.*, pp. 147-156, 1989.
[13] G. Laden, P. Ta-Shma, E. Yaffe, M. Factor, and S. Fienblit, "Architectures for Controller Based CDP," *Proc. the Fifth USENIX Conf. File and Storage Technologies*, pp. 107-121, 2007.
[14] X. Li, C. Xie, and Q. Yang, "Optimal Implementation of Continuous Data Protection (CDP) in Linux Kernel," *Proc. the Int'l Conf. Networking, Architecture, and Storage (NAS '08)*, pp. 28-35, 2008.
[15] "Linux Kernel Drivers," http://sourceforge.net, 2011.
[16] M. Lu, S. Lin, and T. Chiueh, "Efficient Logging and Replication Techniques for Comprehensive Data Protection," *Proc. the 24th IEEE Conf. Mass Storage Systems and Technologies (MSST '07)*, pp. 171-184, 2007.
[17] K. McCoy, *VMS File System Internals.* Digital Press, 1990.
[18] C. Morrey III and D. Grunwald, "Peabody: The Time Travelling Disk," *Proc. IEEE Mass Storage Conf.*, 2003.

[19] C. Morrey III and D. Grunwald, "Content-Based Block Caching," *Proc. the 14th NASA Goddard, 23rd IEEE Conf. Mass Storage Systems and Technologies (MSST '06),* 2006.

[20] L. Moses, "An Introductory Guide to TOPS-20," Technical Report TM-82-22, USC/Information Sciences Inst., 1982.

[21] K. Muniswamy-Reddy, C. Wright, A. Himmer, and E. Zadok, "A Versatile and User-Oriented Versioning File System," *Proc. the Third USENIX Conf. File and Storage Technologies,* 2004.

[22] A. Muthitacharoen, B. Chen, and D. Mazières, "A Low-Bandwidth Network File System," *Proc. the Eighth ACM Symp. Operating Systems Principles,* 2001.

[23] K. Norvag and K. Bratbergsengen, "Log-Only Temporal Object Storage," *Proc. Eighth Int'l Workshop Database and Expert Systems Applications (DEXA '97),* 1997.

[24] B. O'Neill, "Any-Point-in-Time Backups," *Storage, special issue on managing the information that drives the enterprise,* Sept. 2005.

[25] D. Patterson, A. Brown, P. Broadwell, G. Candea, M. Chen, J. Cutler, P. Enriquez, A. Fox, E. Kiciman, and M. Merzbacher, "Recovery-Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies," Technical Report UCB/CSD-0201175, U.C. Berkeley, 2002.

[26] Z. Peterson and R. Burns, "Ext3cow: A Time-Shifting File System For Regulatory Compliance," *ACM Trans. Storage,* vol. 1, no. 2, pp. 190-212, 2005.

[27] J. Piernas, T. Cortes, and J. García, "TPCC- UVA: A Free, Open-Source Implementation of the TPC-C Benchmark," http://www.infor.uva.es/~diego/tpcc-uva.html, 2005.

[28] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz, "Pond: The OceanStore Prototype," *Proc. the Second USENIX Conf. File and Storage Technologies (FAST),* pp. 1-14, 2003.

[29] M. Rosenblum and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," *ACM Trans. Compute System,* vol. 10, no. 1, pp. 26-52, Feb. 1992.

[30] A. Sankaran, K. Guinn, and D. Nguyen, "Volume Shadow Copy Service," *POWER,* vol. 14, Mar. 2004.

[31] 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," *Proc. the Seventh ACM Symp. Operating Systems Principles,* pp. 110-123, 1999.

[32] M. Seltzer, K. Bostic, M. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *Proc. Winter 1993 USENIX Tech. Conf.,* pp. 307-326, 1993.

[33] D. Smith, "The Cost of Lost Data," *J. Contemporary Business Practice,* vol. 6, no. 3, 2003.

[34] C.A.N. Soules, G.R. Goodson, J.D. Strunk, and G.R. Ganger, "Metadata Efficiency in Versioning File Systems," *Proc. the Second USENIX Conf. File and Storage Technologies (FAST),* pp. 43-58, 2003.

[35] "Storage Performance Council SPC-1 Specification," http://www.storageperformance.org/spces, 2011.

[36] "Storage Performance Council, SPC-1 Benchmark Results," http://www.storageperformance.org/results, 2011.

[37] P. Ta-Shma, G. Laden, M. Ben-Yehuda, and M. Factor, "Virtual Machine Time Travel Using Continuous Data Protection and Checkpointing," *ACM SIGOPS Operating Systems Rev.,* vol. 42, no. 1, pp. 127-134, 2008.

[38] "University of Massachusetts SPC Traces," http://www.ssrc.ucsc.edu/wikis/ssrc/SoftwareTraces/SpcTraces, 2011.

[39] J. Wires and M.J. Feeley, "Secure File System Versioning at the Block Level," *ACM SIGOPS Operating Systems Rev.,* vol. 41, no. 3, pp. 203-215, June 2007.

[40] W. Xiao, Y. Liu, Q. Yang, J. Ren, and C. Xie, "Implementation and Performance Evaluation of Two Snapshot Methods on iSCSI Target Storages," *Proc. the 14th NASA Goddard, 23rd IEEE Conf. Mass Storage Systems and Technologies (MSST '06),* 2006.

[41] Q. Yang, W. Xiao, and J. Ren, "TRAP-Array: A Disk Array Architecture Providing Timely Recovery to Any Point-in-Time," *Proc. the 33rd Ann. Int'l Symp. Computer Architecture,* pp. 289-301, 2006.

[42] B. Zhu, K. Li, and H. Patterson, "Avoiding the Disk Bottleneck in the Data Domain Deduplication File System," *Proc. the Sixth USENIX Conf. File and Storage Technologies (FAST),* pp. 269-282, 2008.

[43] N. Zhu and T. Chiueh, "Portable and Efficient Continuous Data Protection for Network File Servers," *Proc. the 37th Ann. IEEE/IFIP Int'l Conf. Dependable Systems and Networks (DSN '07),* pp. 687-697, 2007.

Jing Yang is working toward the PhD degree in computer architecture from Huazhong University of Science and Technology. His current research interests include I/O architecture, file system, network storage system, and computer architecture.

Qiang Cao received the MS and PhD degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 2000 and 2003, respectively. He is currently an associate professor in the Department of Computer Science and Technology at Huazhong University of Science and Technology. His current research interests include computer architecture, massive storage system, file system, I/O architectures, Data Storage, and distributed systems. He is a member of the IEEE.

Xu Li received the PhD degree in computer science from Huazhong University of Science and Technology in 2008. He joined WUHAN WISDRI Co. at China in 2008. His current research interests include computer architecture, reliability of network storage system.

Changsheng Xie received the MS degree in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1988. He is currently a full professor of computer science at Huazhong University of Science and Technology, and deputy director of Wuhan National Laboratory of Optoelectronics. His primary research interests include computer architecture, network storage, and high density and performance storage technology. He is a memeber of the IEEE.

Qing Yang received the MS degree from University of Toronto and the PhD degree from University of Louisiana, Lafayette, in 1985 and 1988, respectively. He is currently a Distinguished engineering professor in the Department of Electrical, Computer and Biomedical Engineering, University of Rhode Island where he has been a faculty member since 1988. His research interests include Computer Architectures, Parallel and Distributed Computing, I/O architectures and Data Storage, Storage Networking, Embedded Computer Systems and Applications, Neural-Machine Interface, Computer application in biomedical engineering. He is an IEEE fellow and a member of the ACM SIGARCH.