IEICE TRANS. INF. & SYST., VOL.E97–D, NO.7 JULY 2014

PAPER Special Section on Cloud and Services Computing

Fast Recovery and Low Cost Coexist: When Continuous Data Protection Meets the Cloud

Yu GU[†], ***Nonmember***, **Chuanyi LIU**[††], ***and*** **Dongsheng WANG**[†††a)], ***Members***

SUMMARY Cloud computing has rising as a new popular service paradigm with typical advantages as ease of use, unlimited resources and pay-as-you-go pricing model. Cloud resources are more flexible and cost-effective than private or colocation resources thus more suitable for storing the outdated backup data that are infrequently accessed by continuous data protection (CDP) systems. However, the cloud achieves low cost at the same time may slow down the recovery procedure due to its low bandwidth and high latency. In this paper, a novel block-level CDP system architecture: MYCDP is proposed to utilize cloud resources as the back-end storage. Unlike traditional delta-encoding based CDP approaches which should traverse all the dependent versions and decode the recovery point, MYCDP adopts data deduplication mechanism to eliminate data redundancy between all versions of all blocks, and constructs a version index for all versions of the protected storage, thus it can use a query-and-fetch process to recover version data. And with a specific version index data structure and a disk/memory hybrid cache module, MYCDP reduces the storage space consumption and data transfer between local and cloud. It also supports deletion of arbitrary versions without risk of invalidating some other versions. Experimental results demonstrate that MYCDP can achieve much lower cost than traditional local based CDP approaches, while remaining almost the same recovery speed with the local based deduplication approach for most recovery cases. Furthermore, MYCDP can obtain both faster recovery and lower cost than cloud based delta-encoding CDP approaches for any recovery points. And MYCDP gets more profits while protecting multiple systems together.

key words: *CDP, block-level, cloud, deduplication, MYCDP*

1. Introduction

Continuous data protection (CDP), also called continuous backup or real-time backup, is a methodology that continuously captures and stores data modifications independent of the primary data, enabling recovery from any point in the past. To provide the most fine-grained data recovery, it must archive every data update, thus the backup volume could be too huge to overwhelm the capacity of CDP system, especially in data-intensive and long-term scenario. To settle this problem, CDP systems adopt various technologies to reduce storage space usage. The mainstream approach uses delta-encoding scheme to reduce the space consumption, which results in dependency between versions. The recovery procedure in these solutions should traverse all the dependent versions according to the recovery point and take computation to decode corresponding version. The more dependent versions and complicated encoding used, the longer time the recovery would take. So the backup cost and recovery speed, which are two most important factors of CDP systems, become conflicting in these solutions. New ideas need to be adopted to solve this predicament.

During the past decade, Cloud computing rises as a new service paradigm, and becomes more and more popular. Many IT systems are built on cloud platform now, utilizing the relatively unlimited resources of cloud with a pay-as-you-go pricing model [1]. And cloud resources are more flexible and cost-effective than private or colocation resources [2].

As in CDP scenarios, most recoveries demand versions not too far ago from present, and data of relatively old versions are rarely needed. Using private or colocation resources to contain these huge volume of data leads to tremendous waste brought by idling resources. Therefore, CDP systems can greatly reduce their backup cost by using scalable cloud resources to store these long-tail data. However, cloud resources always have low bandwidth and high latency to their customers, which could extremely slow down the recovery procedure. Thus it is a big challenge to achieve low backup cost while ensuring high recovery speed within cloud based CDP systems. No previous work has studied this problem.

To solve this problem, a novel CDP architecture, named MYCDP, is proposed in this paper. MYCDP is a block-level CDP that can be compatible with all types of file systems and operation systems, and can utilize various kinds of cloud resources as the back-end storage. Unlike traditional delta-encoding based CDP approaches, MYCDP adopts data deduplication mechanism to eliminate data redundancy between all versions of all blocks, and uses query-and-fetch process to recover version data instead of delta-decoding process used by delta approaches. By using specific architecture and data structures, MYCDP can make low cost and fast recovery coexisting in cloud based CDP scenarios. It also supports deletion of arbitrary versions without risk of invalidating some other versions.

The main contribution we try to present in this paper is the first cloud based CDP which can make fast recovery and low cost coexist by using data deduplication technology. The rest of the paper is organized as follows: Sect. 2 presents the related work. Section 3 discusses MYCDP's design principle, system architecture and key processing procedures.

Manuscript received October 30, 2013.
Manuscript revised January 14, 2014.

[†]The author is with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China.
[††]The author is with the Software School, Beijing University of Posts and Telecommunications, Beijing 100876, China.
[†††]The author is with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China.
a) E-mail: wds@tsinghua.edu.cn

DOI: 10.1587/transinf.E97.D.1700

The evaluation and analysis are presented in Sect. 4. Finally in Sect. 5 we conclude and discuss future work.

2. Related Work

Many studies have been done on continuous data protection. Laden et al. [3] proposed four alternative CDP architectures based on storage controller and compared them with respect to write performance and storage space usages. Peabody [4] demonstrated that for many workloads, most of disk sectors written contain identical content to previously written sectors, motivating the need for content-based coalescing. Trap-array [5] designed a disk array architecture that provided timely recovery to any point-in-time, leveraging RAID controller's XOR operations upon consecutive versions of blocks. Clotho [6] added a new layer in Linux I/O hierarchy to gather block-level versions and utilized differential encoding between the block versions to reduce disk space occupancy. VDisk [7] presented a secure, block-level versioning system that added file-grain versioning to a standard, unmodified file system by logging updates to a read-only log. Mariner [8] is an iSCSI-based storage system supporting CDP, which employed a track-based log that unified the long-term logging required for CDP and short-term logging for low-latency disk writes. SWEEPER [9] dynamically determines the selection algorithm based on user specified recovery time and recovery point objectives, which allows trade-offs between recovery time and data correctness. CDP systems take trial-and-error test or manual checkpoints to ensure consistence, and TH-CDP [10] encapsulates the checkpoint information into I/O request packet queue to make checkpoints without suspending applications. Although above researches are good references for us on CDP architecture, data compression, block-level modification gathering, data consistency ensuring and recovery speed accelerating, they all focus on local based CDP, no cloud based CDP has been studied yet. As we mentioned, there are new challenges in cloud based CDP scenarios which need to be studied carefully, and some existing approaches should be reconsidered.

There are some research on data backup and recovery based on cloud computing paradigm. Wood et al. [2] have proved that using public cloud resources to perform archiving of infrequently needed data is cheaper than using private or colocation resources, which motivated us to store CDP data to cloud. Cumulus [11] backs up a file system to cloud storage using a least-common-denominator cloud interface, thus supports many kinds of cloud services. The same kind of interface can be used by MYCDP. And DR-Cloud [12] utilizes multiple clouds cooperatively to backup data to ensure higher data reliability, lower total cost and faster recovery than one cloud, which can be adopted by MYCDP to ensure high data reliability.

In recent years, data deduplication has become a commodity component in data intensive storage systems. CASPER [13] and Venti [14] divide files into fixed-size chunks before deduplication; LBFS [15] uses Rabin fingerprint algorithm [16] to divide files into variable-size chunks, then eliminates duplicate chunks by comparing their unique identifiers. Their results have proved the effectiveness of data deduplication on various real world data. And Lillibridge et al. [17] adopted cache, container capping and forward assembly area techniques to improve restore speed of deduplication based backup systems, which can be used by MYCDP to speed up recovery further. There are also many commercial products based on data deduplication technique, such as EMC Data Domain [18], Quantum DXi Series [19] and Symantec PureDisk [20], which are good design references for MYCDP's deduplication module.

3. MYCDP System

3.1 Design Philosophy

As mentioned above, storing backup data in the cloud could reduce total cost. But there are some challenges when designing such a CDP system. In general, three matters are the most important:

- To achieve the best compatibility with various kinds of systems to be protected and cloud platforms to be utilized.
- To ensure high recovery speed while using cloud resources.
- To decrease total cost furthest by reducing the occupancy of cloud resources.

We design MYCDP as a block-level CDP system, thus it can protect various systems with different kinds of file systems and operation systems by using appropriate block write bypass mechanisms in block-level layer such as the SCSI protocol. It also has the minimal performance impact to the protected system than file-level and application-level CDP systems. To leverage resources from various cloud platforms exposing different interfaces, MYCDP adopts a least-common-denominator cloud storage interface, i.e. get/put/delete data, like Cumulus [11] does.

Although pricing models of different cloud service providers are diversified, they mainly consist of three factors [21]: storage pricing, data transfer (in/out) pricing and request (get/put/delete/etc.) pricing. To reduce total cost, MYCDP should decrease the backup volume by a certain lossless data compression mechanism.

Delta-encoding based CDP systems need to read a snapshot as the base version and some deltas to decode the exact data. Since more data needs to be fetched, recovery will be greatly slowed down in cloud mode. So in MYCDP, we employ the data deduplication mechanism to eliminate data redundancy instead of delta-encoding. MYCDP takes each block data as an atomic data chunk and stores only unique chunks. It can reduce the duplications among versions of same and different blocks, so it is more effective in saving storage space than delta-encoding approaches which can only utilize similarity between versions of same block.

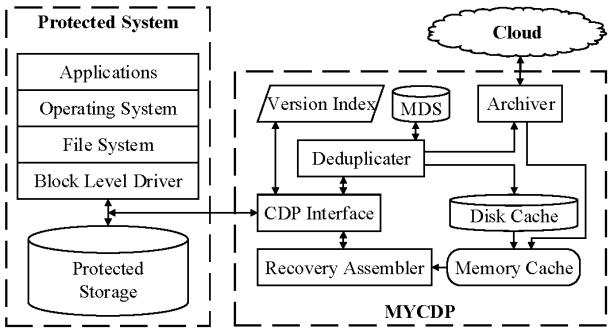


Fig. 1 MYCDP architecture.

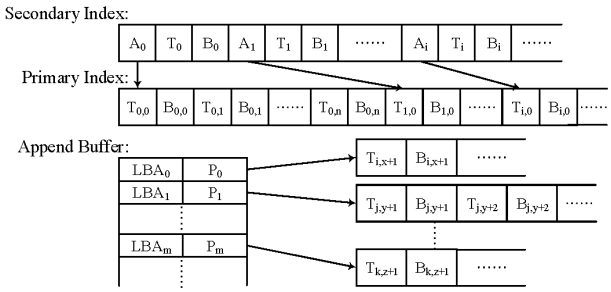


Fig. 2 Version_Index data structure.

And a specified version can be restored by directly reading the chunks composing that version, without any delta-decoding computation. So it's cheaper and faster in cloud mode. In addition, MYCDP can support deletion of arbitrary versions without risk of invalidating some other versions, while the delta-encoding approaches cannot.

MYCDP adopts a specific version index data structure and management mechanism to collaborate with deduplication. And the data frequently needed by recovery is held locally with a hybrid cache consisting of an on-disk large cache and an in-memory small cache to speed up recovery processes.

MYCDP ensures consistence of file system by a widely used trial-and-error mechanism.

3.2 System Architecture

As shown in Fig. 1, MYCDP system consists of several components and data structures. We introduce them briefly as follows and explain their behavior and relationship in the next subsection where version backup, recovery and delete procedures are depicted.

- CDP_Interface

MYCDP exposes the CDP_Interface to the protected system to handle block-level backup and recovery requests. While backup, the protected system bypasses all the block-level write requests to the CDP_Interface. Every request is divided into a certain number of tuples ⟨*LBA*, *Timestamp*, *Block_Data*⟩, each indicating the logical block address, occurrence time and new data of a single block write contained in that request. The *Timestamp* also represents the concept of version. The recovery request is also sent to the CDP_Interface, with a parameter tuple ⟨*Timestamp*, *Start_LBA*, *End_LBA*⟩ indicating the point-in-time and block range to be recovered.

- Version_Index

The Version_Index is used to contain information about all historical versions of all blocks, i.e. the mapping from ⟨*LBA*, *Timestamp*⟩ to *Block_Data*, which can be used to query any version of any block.

Because of the huge amount of block versions, e.g. billions of versions could be produced by a usual usage of a 1TB hard disk with 4KB block size, traditional tools such as database or hashtable cannot hold and update these information efficiently. So we designed a specific data structure, named Version_Index, to solve this problem. As shown in Fig. 2, the Version_Index consists of an on-disk two-stage index and an in-memory append buffer. It is updated periodically, not in real time. The on-disk index holds version records until the last update, while new version records since the last update are contained in the append buffer.

The on-disk primary index of the Version_Index is an sequential array composed of fixed-size tuples $\langle T_{i,j}, B_{i,j}\rangle$, and each tuple represents the j^{th} write to the i^{th} block. The $T_{i,j}$ and $B_{i,j}$ are *Timestamp* and *Block_Data_ID* of that write, where the *Block_Data_ID* is the unique identifier of the *Block_Data*.

The on-disk secondary index of the Version_Index is an sequential array composed of fixed-size tuples $\langle A_i, T_i, B_i\rangle$, where the A_i points to the start offset of records of the i^{th} block in the primary index, the T_i and B_i are *Timestamp* and *Block_Data_ID* of the latest write to that block. So we can use it to greatly accelerate the version lookup for each block.

The in-memory append buffer utilizes a hashtable to hold all *LBA*s of newcome block write since the last index update, and each *LBA* has a pointer to a queue containing records of new writes to the corresponding block. Considering the robustness, MYCDP writes down all these new records to a log file, which can be used to recover the append buffer after a system fault.

If there are multiple client systems to be protected, MYCDP maintains a set of primary and secondary indexes for each client, while using the same append buffer for all clients to utilize memory space efficiently. And a value of client number is added to each hashtable elements of the append buffer.

- Deduplicater

The Deduplicater takes each *Block_Data* as an atomic data chunk for deduplication because of the small block size. For each chunk, a *Chunk_ID* (also called *Block_Data_ID*) is generated by taking cryptographic hash function on its content. The Deduplicater checks the uniqueness of each chunk through comparing its *Chunk_ID* with others, and only unique chunks will be stored in cloud or local cache.

- MDS

The MDS maintains three kinds of metadata information.

The *Chunk_Info* contains metadata of all chunks, consisting of tuples ⟨*Chunk_ID*, *Cloud_RC*, *Cloud_Location*, *Cache_RC*, *Cache_Location*⟩, where the *Cloud_RC* and *Cache_RC* indicate the corresponding chunk's reference count (i.e. redundancy rate) referred by the cloud and the Disk_Cache, and the *Cloud_Location* and *Cache_Location* are location information being used to fetch that chunk from the cloud and the Disk_Cache.

The *Latest_Version* is an array recording the *Chunk_ID*s of all blocks' latest version in sequence.

The *Recent_Writes* is an FIFO queue consisting of a certain number of tuples ⟨*LBA*, *Chunk_ID*⟩ indicating those recent block writes.

Multiple *Latest_Version* and *Recent_Writes* will be contained in the MDS when protecting multiple client systems. While a single *Chunk_Info* is used in this situation to eliminate duplicated chunks between client systems.

The implementation of the MDS can refer to lots of practical deduplication systems such as Data Domain [18].

- Archiver

The Archiver is responsible for saving/restoring all unique chunks to/from the cloud. It transfers chunks to the cloud in batches to reduce the request cost, and uses multiple threads to put/get data to/from the cloud to furthest utilize network bandwidth.

- Disk_Cache

The Disk_Cache contains a dynamic set of unique chunks on local disk to speed up recovery. It uses a raw disk partition without any file system, and employs a block allocation bitmap to manage storage space. The partition's block size is set to the minimum value of the protected systems' block size, and each chunk is stored continuously. Thus a chunk can be located simply by its start LBA. Since all chunks are archived to the cloud, there is no need to put data back to the cloud during cache replacement, largely reducing the data transfer cost. MYCDP uses SSD as the Disk_Cache due to its good performance on random or small read and write requests.

- Recovery_Assembler

The Recovery_Assembler is in charge of assembling recovery data using chunks.

- Memory_Cache

The Memory_Cache holds a set of hot chunks in memory to reduce duplicated chunk reading, and uses the LRU algorithm to replace chunks.

3.3 Processing Procedures

3.3.1 Version Backup Procedure

As a CDP system, MYCDP runs nonstop to process continuous stream of block-level write requests from the protected system. The CDP_Interface interprets each request to separated single block writes, and sends each *Block_Data* to the Deduplicater. The *Block_Data_ID* (*Chunk_ID*) is generated by the Deduplicater, then the CDP_Interface records all these writes as versions to the Version_Index.

The sequential structure and big size of the Version_Index's primary index increases the complexity of appending version records, so we make a compromise on its update granularity to a time period. During every periodic update, MYCDP generates a brand new primary index by appending records of each block in append buffer to the tail of that block's records in the old primary index. Although the size of primary index can be in the range of hundreds of MB to tens of GB, the update speed is very fast by sequential reading from old index and sequential writing to new index. The update period is determined according to practical conditions. When update begins, a new append buffer will be created to hold newcome records, and the old append buffer will be destroyed after update. And the fixed-size secondary index can be updated without space reallocation after new primary index is generated.

For each newcome *Block_Data* (chunk), the Deduplicater checks whether it is unique by query the MDS by its *Chunk_ID*. It sends only unique chunks to the Disk_Cache and Archiver. It is also responsible for updating metadata in the MDS.

All chunks referred by the *Latest_Version* and *Recent_Writes* in the MDS are deduplicated and stored in the Disk_Cache. So they logically represent a real-time mirror and a modification buffer of the protected storage respectively, except occupying less space due to deduplication. To achieve this goal, the Deduplicater increases each newcome chunk's *Cache_RC* by two to indicating its new reference by the *Latest_Version* and *Recent_Writes*. Then if a chunk's *Cache_RC* equals to two, it is an uncached chunk and will be sent to the Disk_Cache. The *Latest_Version* and *Recent_Writes* are also updated using the information of each newcome block write. Every time a *Chunk_ID* is replaced or extruded by new one from the *Latest_Version* or *Recent_Writes*, its *Cache_RC* will be decreased by one. And spaces of chunks with *Cache_RC* equaling to zero are recycled.

As the final step, the Archiver stores unique chunks to the cloud. A background garbage collection process also keeps running in MYCDP to delete unreferenced chunks (with *Cloud_RC* equaling to zero) periodically.

3.3.2 Version Recovery Procedure

When the CDP_Interface receives a version recovery request, it will start a recovery process, which includes several work threads: one for query version records from the Version_Index, some for reading data from the SSD Disk_Cache, some for getting data from cloud, and one for assembling data and sending data back to the protected system. All these threads run concurrently, which forms a re-

covery pipeline.

For each block, the version demanded by a recovery request is the version with the biggest *Timestamp* which is no bigger than the recovery request's *Timestamp* parameter. The query thread searches the append buffer of the Version_Index first for appropriate version record. If not found, it searches the on-disk index. As the primary and secondary indexes are both sequential arrays with fixed-size elements, querying them is highly efficient. For the i^{th} block, the query thread gets the tuple $\langle A_i, T_i, B_i \rangle$ and A_{i+1} from the secondary index first, and check if the T_i is no bigger than the recovery *Timestamp*. If so, this record is just the wanted record, and no further search is needed. Otherwise, since the A_i and A_{i+1} are the start and end offset of the i^{th} block's records in the primary index respectively, the query thread uses the binary search algorithm to find the appropriate record from the primary index.

The *Chunk_ID* of the found record is delivered to the assembling thread with the block's *LBA*. In the scenario of recover-to-original-disk, if the found record is the latest version record in the Version_Index, the corresponding block is unchanged and need not to be restored. In this case, a NULL flag with the block's *LBA* will be delivered to the assembling thread.

As we mentioned above, most recoveries demand versions not too far ago from present, so recovery versions of most blocks are just their latest versions, and recovery versions of other blocks are likely generated by recent writes of entire protected storage. The Disk_Cache holds exactly these two kinds of chunks to accelerate recovery speed. And even chunks of old versions may also be contained by the Disk_Cache due to chunk duplication.

For each *Chunk_ID* being queried out, the query thread gets its location from the MDS. Reading requests of chunks within the Disk_Cache are sent to those disk reading threads, and others are sent to those cloud fetching threads. These data reading threads read chunks' data and send them to the assembling thread.

The assembling thread maintains an assembly queue in memory, which is a moving recovery window consisting of many successive blocks. Each element in the queue represents a block need to be restored. For each chunk arrived, the thread copies its data to all positions it appearing in entire queue. To improve I/O performance of the protected storage, block data are sent back in batches and in order approximately. While in some cases, if a few front blocks are not completed, the subsequent batch of completed blocks can be sent out first. This strategy mitigates the impact caused by bandwidth and latency limitations of both local disk and cloud. All unchanged blocks will not be included in the assembly queue in recover-to-original-disk mode.

The Memory_Cache holds a certain number of hot chunks in memory to further reduce chunk reading, thus the Disk_Cache and the Memory_Cache form a hybrid cache to speed up recovery.

3.3.3 Version Delete Procedure

Traditional delta-encoding based CDP systems have dependency between versions, directly deleting some versions from them may make some other versions unrecoverable. While in MYCDP, any version can be deleted without impact to other versions. For version deletion, a request with a parameter tuple $\langle$*Start_Timestamp*, *End_Timestamp*$\rangle$ is sent to MYCDP. Then for each block, all versions belonging to that period except the last one of them can be removed precisely from the cloud to save storage cost. The delete procedure filters the Version_Index to generate a new index excluding records with *Timestamp* between the *Start_Timestamp* and *End_Timestamp*, and decreases the *Cloud_RC* of the chunk referred by each of those records by one. Then sometime later, data and metadata of chunks with *Cloud_RC* equaling to zero can be removed in batches from cloud and the MDS by garbage collection process.

4. Evaluation and Analysis

A proof-of-concept prototype is implemented which includes all the main components of MYCDP. And a variant of TH-CDP [10], named TH-CCDP, is also implemented for comparison. TH-CCDP is a typical delta-encoding based block-level CDP system, which holds all blocks' latest version and all deltas of each block write in the history. To make a fair comparison, TH-CCDP also archives all data to the cloud, and uses a disk cache to hold all blocks' latest version and deltas of recent writes. And it stores deltas in fixed-size batches to improve I/O performance. More specifically, for data referred by the *Latest_Version*, MYCDP allocates space on demand with the assist of its MDS, while TH-CCDP allocates eventually needed space at the beginning to avoid complicated metadata management. For data referred by the *Recent_Writes*, they both allocate space on demand.

To evaluate MYCDP's effectiveness, we designed an experimental environment. We used the Open Storage Toolkit from Intel Labs [22] to create six iSCSI targets each with 4KB block length and 10GB size based on a SATA disk in a Linux PC, and mounted them to a Linux server with the qemu-kvm tool installed. And we modified the source code of the Intel's Toolkit to bypass all block-level write requests to outside and restore block data from outside through network.

We created six virtual machines (VM) directly using the six iSCSI targets by the qemu-kvm tool, and installed windows XP with 4KB block size NTFS file system on each VM. For each XP, we turned off its virtual memory, redirected main temporary folders to a ramdisk, and installed some common software such as Office, browser, SSH client and etc. Then we invited five colleagues in our lab to use one VM each as their daily work machine for one week. And we also used the sixth VM ourselves for two weeks. The VMs' daily operations include installing/removing software, sending/receiving emails using Outlook, and creat-

ing/modifying/copying/deleting files/directories. The final disk utilizations of those VMs are in the range of 71% to 96%. And we wrote down all the separated block writes of those VMs since being created to six traces respectively as our test benches. The first five VMs generated five 31GB~38GB test benches, and the sixth VM generated a 52GB test bench.

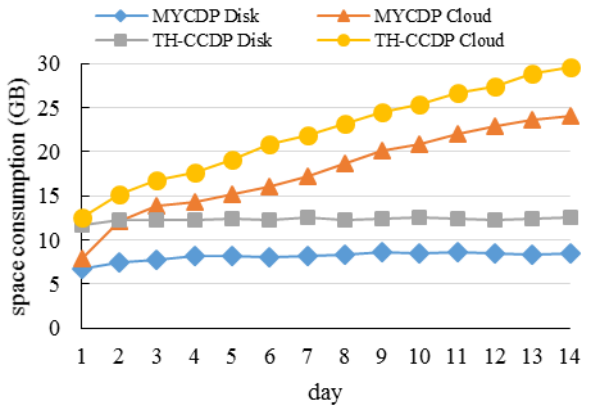
The reason why we chose XP is that XP has relatively smaller size than other OSs. So we can use less data to form a proper data classification characters similar to a typical personal computer, which can accelerate our experiment. According to the experimental results of Peabody [4], various OSs with various file systems have considerable block writes which contain identical content to previous block writes. So we think our experimental results on XP are instructive to cases using other OSs, especially other windows OSs using same NTFS file system.

We deployed both MYCDP and TH-CCDP to a Linux PC with an Intel Core i5-3470 quad-core CPU and 4GB memory. The Version_Index and MDS of MYCDP were held in a SATA disk. Metadata used by TH-CCDP was also contained in that disk. And a 180GB Intel 525 Series mSATA SSD was also installed to that PC. Both MYCDP and TH-CCDP used the SSD as the Disk_Cache with 4KB block size to hold data of the latest version and one million recent writes. We also set the size of the Memory_Cache to 512MB to hold 128k chunks in MYCDP. It should be noted that the Memory_Cache is useless to TH_CCDP due to inexistent reuse among data of base version and deltas.

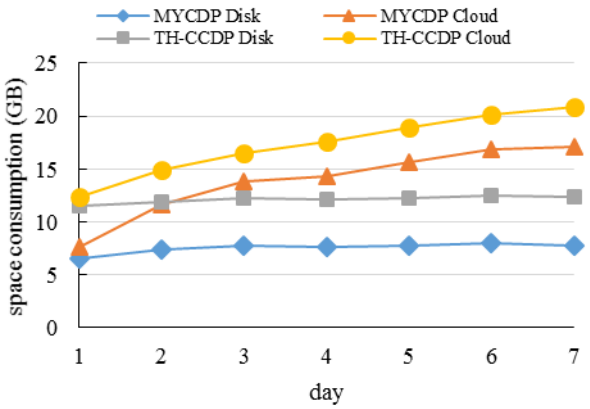
To facilitate evaluation, MYCDP and TH-CCDP used a back-end storage module with adjustable parameter tuple $\langle Bandwidth, Latency \rangle$. Thus we used $\langle 50Mbps, 150ms \rangle$ to simulate cloud-like storage according to our test results in real world environment, and used $\langle 500Mbps, 15ms \rangle$ to simulate local-like storage according to the parameters of typical local disk storages. And we think modest differences on these parameters will not affect our conclusions below. More specifically, the difference of latency can be greatly covered by MYCDP's multithreading and pipeline mechanism during recovery, and since our evaluation is to qualitatively estimate the effectiveness of MYCDP relative to local-based approaches and delta-encoding approaches, the modest difference of bandwidth will not overturn related conclusions.

The evaluation focused on two import factors: total cost and recovery speed of MYCDP and TH-CCDP in different situations. And for the various pricing models of cloud and local resources, we use the storage space consumption to represent the cost. The smaller space is used, the lower storage and transfer cost is spent. And the request cost of the cloud can be ignored due to its very low price [21].

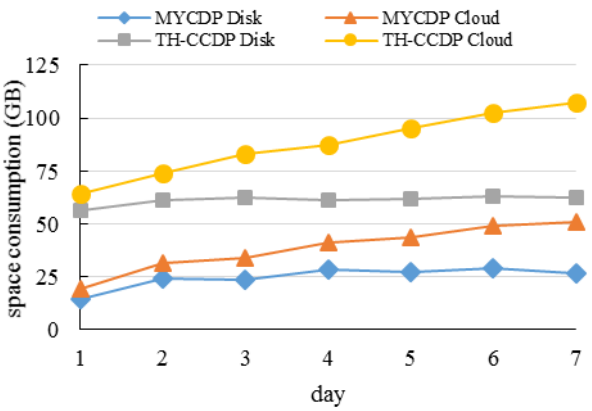
Before data analyses, it should be noted that all the experimental results of the first five VMs have very similar characteristics. Since the purpose of this evaluation is to qualitatively estimate the effectiveness of MYCDP, we only present results of the first VM for single VM cases. And the first VM's disk usages at the end of the first day and the seventh day are 5.7GB and 8.1GB respectively.



(a) Protect the sixth VM.



(b) Protect the first VM.



(c) Protect the first five VMs together.

Fig. 3 Storage space consumption comparison.

First, we ran MYCDP and TH-CCDP with different test benches to see their storage space consumptions, and the result is shown as Fig. 3.

The three parts of Fig. 3 show the space consumption of MYCDP and TH-CCDP when they are protecting the sixth VM, the first VM, and the first five VMs together respectively. We can see space usage of the cloud becomes bigger and bigger than space usage of the Disk_Cache as time passed. So by utilizing the flexible and cost-effective

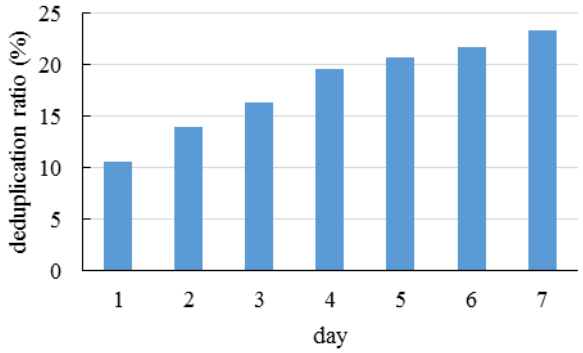


Fig. 4 MYCDP's deduplication ratio when protecting the first VM.

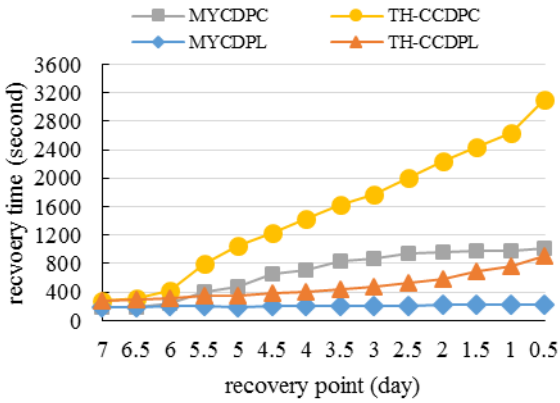


Fig. 5 Recovery time comparison.

cloud resources to hold those outdated and infrequently accessed data with large volume, MYCDP can achieve much lower cost than local based CDP systems. And the figure also shows that MYCDP consumes both smaller local and cloud space than TH-CCDP in same condition. Although the space allocation strategy of data referred by the *Latest_Version* gives MYCDP some superiority in disk and cloud space consumption, that superiority became negligible very quickly. According to the analysis of trace, about 84% blocks of the first VM have been written at the beginning of the third day, and 96% blocks have been written in the end. And other VMs have similar phenomenon. So the comparison is generally fair except the first two days. Furthermore, the more systems being protected, the more advantages MYCDP has. That's because data deduplication can eliminate redundancy globally while delta-encoding can only eliminate similarity of versions of same block. Obviously, more systems have more redundancy. In addition, Data operations causing block relocation will generate more versions in block-level. For example, the disk defragmentation will produce numerous block changes while no upper-level data is changed in deed. Delta-encoding is inefficient to handle this problem, and consumes more storage space. While for data deduplication, such case only increases size of the Version_Index, and no more data will be stored. So MYCDP can achieve lower cost than TH-CCDP in cloud mode, especially in multi-client situations. In addition, MYCDP also reduces the wear on SSD by decreasing data writes.

Figure 4 shows the deduplication ratio when MYCDP protects the first VM. Since MYCDP uses fixed-size block for deduplication, we cannot expect MYCDP having very high deduplication ratio in single VM cases. The result is reasonable according to the experimental results of Peabody [4]. And we can see the initial deduplication ratio is relatively low and later deduplication ratios grow higher, that's because most initial data is OS and application data, while regular data (especially user data) is written later. As we mentioned above, MYCDP's space allocation strategy has superiority in the first few days, which just makes up the weakness of low deduplication ratio in those days in the space consumption comparison with TH-CCDP. In later days, MYCDP's space consumption advantage over TH-CCDP really comes from high deduplication ratio.

Then, to evaluate the recovery speed, we employed four candidate schemes: MYCDPC and TH-CCDPC using cloud-like back-end storage, MYCDPL and TH-CCDPL using local-like back-end storage. And we measured their recovery time (RT) with different recovery point in whole-disk-recover mode of the first VM, shown as Fig. 5.

In Fig. 5 the horizontal axis denotes the recovery point, e.g., the value of 6.5 represents recovering to the version of half a day ago. We can see if the recovery point is not too far ago from present, the RT of cloud mode is almost the same as the RT of local mode due to the effectiveness of local cache. And this is the most cases in real world CDP scenarios. When recovering to earlier points, local based CDP systems recover faster than corresponding cloud based CDP systems. And MYCDP is always faster than TH-CCDP with any recovery point in both local and cloud mode. In addition, when recovering to earlier points, the RT of MYCDPL has no obvious changes, the RT of MYCDPC increases first then reaches an upper limit, and the RT of TH-CCDPL and TH-CCDPC keep growing sharply and endlessly. It is because MYCDP only reads chunks of exact data from local or cloud, which have a limited overall size, while TH-CCDP need to read a base version and unlimited deltas. Although TH-CCDP can insert more snapshots during the history to shorten the delta chain and speed up recovery, it will consume much more local and cloud storage space.

In Fig. 6, Speedup_C denotes the speedup of MYCDPC against TH-CCDPC in same scenario with Fig. 5, while Speedup_L denotes the speedup of MYCDPL against TH-CCDPL. We can see the earlier version to be recovered, the bigger speedup MYCDP achieves against TH-CCDP roughly in both local and cloud mode.

Finally, we evaluated the hybrid cache mechanism of MYCDP. We used a 512MB/1GB/1.5GB Memory_Cache in turn to recover the first VM. And we also held 0.5/1/2 million recent writes in the Disk_Cache and did it again. The result is shown as Fig. 7.

We can see enlarging the Memory_Cache and Disk_Cache can benefit the recovery speed of MYCDP in different pattern. From Fig. 7 (a), larger Memory_Cache can

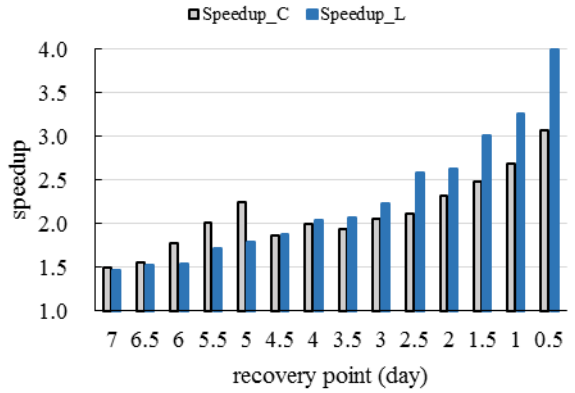


Fig. 6 Recovery speedup of MYCDP.

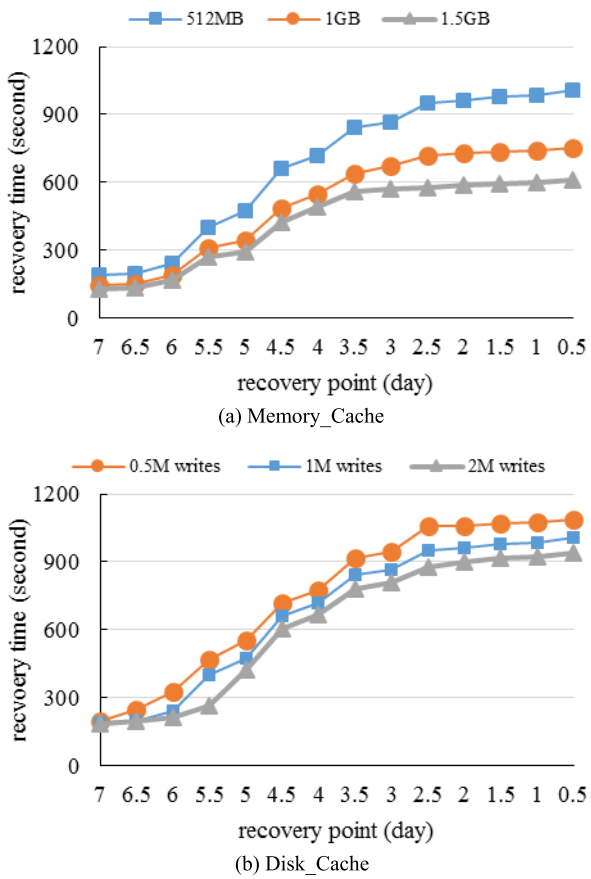


Fig. 7 Recovery time of MYCDP with different cache settings.

make MYCDP recover faster for any recovery point due to reduce both local disk and cloud I/O, and can reach smaller upper limit of RT earlier. While Fig. 7 (b) illustrates that larger Disk_Cache can guarantee high recovery speed with earlier recovery point, and can only reduce the upper limit of RT slightly. So we can separately adjust the Memory_Cache and Disk_Cache according to practical conditions to achieve more cost-effective recovery performance.

And the final space consumptions of the Disk_Cache with 0.5/1/2 million recent writes are approximately 6.7/7.8/8.8 GB respectively. More specifically, the space consumptions of chunks referred by the *Recent_Writes* are about 1.1/2.3/3.9 GB respectively, and the space consumptions of chunks referred by the *Latest_Version* are the same (about 5.9 GB). Note that some recent writes are also latest version writes, and all data of recent writes and latest version writes are deduplicated together, so there are many chunks in Disk_Cache referred by both the *Recent_Writes* and *Latest_Version*. That's why the sum of space consumption of chunks referred by the *Recent_Writes* and *Latest_Version* is bigger than the space consumption of Disk_Cache. So the disk space usage of MYCDP increases very slowly with the increase of cached recent writes, due to deduplication. In addition, even the disk space usage of MYCDP caching 2M recent writes is smaller than the disk space usage of TH-CCDP caching 1M recent writes.

5. Conclusion and Future Work

According to the evaluation, by utilizing flexible and cost-effective cloud resources, MYCDP can achieve much lower cost than traditional local based CDP systems. Its data deduplication mechanism can reduce the space consumption of both cloud and local storage, which can further reduce the total cost compared with the cloud based delta-encoding CDP scheme. The more systems being protected together, the more cost MYCDP saves. With the query-and-fetch based recovery process and the effective disk/memory hybrid cache, MYCDP can reduce data transfer and computation to achieve faster recovery than cloud based delta-encoding CDP scheme. And for most cases that the recovery point is not too far ago from present, the recovery speed of MYCDP is almost the same with the local based deduplication approach.

While single cloud has limitations of resources and reliability, we plan to borrow the mechanism proposed by DR-Cloud [12] into MYCDP to leverage resources of multiple clouds in the future.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (61202081) and the 863 Program of China (2012AA012609).

References

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A Berkeley view of cloud computing," Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS, vol.28, p.13, 2009.

[2] T. Wood, E. Cecchet, K.K. Ramakrishnan, P. Shenoy, J. Van der Merwe, and A. Venkataramani, "Disaster recovery as a cloud service: Economic benefits & deployment challenges," 2nd USENIX Workshop on Hot Topics in Cloud Computing, 2010.

[3] G. Laden, P. Ta-Shma, E. Yaffe, M. Factor, and S. Fienblit, "Architectures for controller based CDP," FAST. 2007, vol.7, pp.21–36, 2007.

[4] C.B. Morrey and D. Grunwald, "Peabody: the time travelling disk," Proc. 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, 2003. (MSST 2003), pp.241–253, 2003.
[5] Q. Yang, W. Xiao, and J. Ren, "Trap-array: A disk array architecture providing timely recovery to any point-in-time," ACM SIGARCH Computer Architecture News, vol.34, no.2, pp.289–301, 2006.
[6] M. Flouris and A. Bilas, "Clotho: Transparent data versioning at the block I/O level," MSST. 2004, pp.315–328, 2004.
[7] J. Wires and M.J. Feeley, "Secure file system versioning at the block level," Proc. 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pp.203–215, Lisbon, Portugal, 2007.
[8] M. Lu, S. Lin, and T. Chiueh, "Efficient logging and replication techniques for comprehensive data protection," 24th IEEE Conference on MSST. 2007, pp.171–184, 2007.
[9] V. Akshat, V. Kaladhar, R. Ramani, and J. Rohit, "SWEEPER: An efficient disaster recovery point identification mechanism," FAST. 2008.
[10] Y. Sheng, D. Wang, J. He, and D. Ju, "TH-CDP: An efficient block level continuous data protection system," IEEE International Conference on Networking, Architecture, and Storage, 2009. NAS 2009, pp.395–404, 2009.
[11] M. Vrable, S. Savage, and G.M. Voelker, "Cumulus: Filesystem backup to the cloud," ACM Trans. Storage (TOS), vol.5, no.4, p.14, 2009.
[12] Y. Gu, D. Wang, and C. Liu, "DR-Cloud: Multi-cloud based disaster recovery service," J. Tsinghua Science and Technology, vol.19, pp.13–23, Feb. 2014.
[13] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T.C. Bressoud, and A. Perrig, "Opportunistic use of content addressable storage for distributed file systems," USENIX Annual Technical Conference, General Track, pp.127–140, 2003.
[14] S. Quinlan and S. Dorward, "Venti: A new approach to archival storage," FAST. 2002, vol.2, pp.89–101, 2002.
[15] A. Muthitacharoen, B. Chen, and D. Mazieres, "A low-bandwidth network file system," ACM SIGOPS Operating Systems Review, vol.35, no.5, pp.174–187, 2001.
[16] M.O. Rabin, "Fingerprinting by random polynomials," Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.
[17] M. Lillibridge, K. Eshghi, and D. Bhagwat, "Improving restore speed for backup systems that use inline chunk-based deduplication," FAST. 2013, pp.183–198, Feb. 2013.
[18] B. Zhu, K. Li, and R.H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," FAST. 2008, vol.8, pp.269–282, 2008.
[19] Quantum Corporation, "Quantum DXi-Series disk backup systems," http://www.quantum.com/products/disk-basedbackup/index.aspx, accessed Oct. 27. 2013.
[20] Symantec Corporation, "Symantec NetBackup PureDisk," http://www.symantec.com/netbackup-puredisk, accessed Oct. 27. 2013.
[21] Amazon Corporation, "Amazon simple storage system (Amazon S3)," http://aws.amazon.com/s3, accessed Oct. 27. 2013.
[22] Intel Labs, "Open storage toolkit from Intel Labs," http://sourceforge.net/projects/intel-iscsi/, accessed Oct. 27. 2013.

Yu Gu received the B.S. degree in Computer Science and Technology from Tsinghua University in 2002. He is now a Ph.D. student in Department of Computer Science and Technology of Tsinghua University.

Chuanyi Liu received the B.S. degree in Computer and Information Technology from Beijing Jiaotong University in 2004, and ceived the Ph.D. degree in Computer Science and Technology from Tsinghua University in 2009. He is now an assistant professor of computer science & engineering at Beijing University of Posts and Telecommunications.

Dongsheng Wang received the B.S. and Ph.D. degrees from Harbin Institute of Technology. He is now a professor of Research Institute of Information Technology in Tsinghua University. He is also a senior member of China Computer Federation, and member of IEEE.