Towards Fast De-duplication Using Low Energy Coprocessor

Liang Ma, Caijun Zhen, Bin Zhao, Jingwei Ma, Gang Wang and Xiaoguang Liu
College of Information Technical Science,Nankai University
Nankai-Baidu Joint Lab
Weijin Road 94, Tianjin, 300071, China
maliang318@gmail.com, wgzwp@163.com

***Abstract*—Backup technology based on data de-duplication has become a hot topic in nowadays. In order to get a better performance, traditional research is mainly focused on decreasing the disk access time. In this paper, we consider computing complexity problem in data de-duplication system, and try to improve system performance by reducing computing time. We put computing tasks on commodity coprocessor to speed up the computing process. Compared with general-purpose processors, commodity coprocessors have lower energy consumption and lower cost. Experimental results show that they have equal or even better performance compared with general-purpose processors.**

***Keywords*-De-duplication; Commodity coprocessors; Computing complexity; Low energy**

I. Introduction

With the development of information technology, data has become the foundation of all various trades and industries. How to store the massive backup data generated in the backup process has become a new hot spot. In the process of full backup, incremental backup and continuous data protection (CDP) [1]–[4], if we store the backup data directly without any treatment, it will cause large amounts of storage overhead. The data de-duplication technology emerged in recent years is a good solution to backup data overhead problem. By discarding the duplicate data produced in the backup process, the storage overhead problem can be well solved. For the example of Data Domain De-duplication File System (DDFS) [5], which applies data de-duplication to the backup data produced in a month, achieves a compression ratio of 38.54:1, and reduces the storage overhead and network bandwidth consumption greatly as expected.

Since there is a great performance gap between CPU and I/O operations, for the pursuit of better performance, traditional de-duplication systems [5], [6] focus on how to decrease the disk access time. However, with the emergence of new storage media such as SSD, this would no longer be a problem. Together with the development of network technology, which enlarges data transmission bandwidth greatly, CPU has to carry out more and more data computation tasks such as SHA-1 and compression during the data de-duplication process, which makes CPU a potential bottleneck. If CPU is occupied by too many highly complex tasks, the I/O performance will be greatly decreased. There are two possible ways to solve this problem: using more computing units to distribute the computing tasks and using special computing devices in individual computing unit.

Using more computing units can noticeably improve CPU performance and has better scalability [7], [8]. However, how to maintain data synchronization between computing units remains a big problem. Using special computing devices does not have synchronization problem, but has higher cost and longer development cycle.

We found that putting the computing tasks on high performance commodity coprocessors could be a good solution, which avoids introducing extra synchronization problem and has better computing performance, lower energy consumption and less cost.

In this paper, we present a new continuous data protection system based on data de-duplication: DedupCDP. Unlike traditional de-duplication systems, we pay more attention to reducing the computing pressure on CPU by cooperating with coprocessors. The rest of this paper is organized as follows: Section II surveys related work. Section III presents the architecture of DedupCDP system. Section IV describes how to fast de-depulication using coprocessor. In section V, we report on various simulation experiments with real data. Finally, we describe our conclusions and future work in Section VI.

II. Related Work

Early de-duplication storage systems, such as EMC's Centra Content Addressed Storage (CAS) [9], implemented de-duplication at file level. File level de-duplication storage systems use the fingerprint of a file to detect identical files. Since they only implement de-duplication at file level, such systems achieve only limited storage saving. Moreover, file level de-duplication storage systems have poor portability.

It is natural to implement de-duplication at block level because of the block read and write interface provided by kernel. Venti [10] computes a cryptographic hash for each disk contents and does data de-duplication by comparing cryptographic hashes of blocks. Venti achieves a better compression radio than systems implemented at file level and can be easily ported to other operating systems. Venti implements a large on-disk index with a cache for fast fingerprints lookup. Since there is no locality in fingerprints, its index cache is ineffective. Its throughput is still limited to less than 7MB/sec though 8 disks are used to lookup fingerprints in parallel.

Early studies only focused on compression radio but not on techniques to high throughput. To decrease the disk access time, DDFS [5] uses Bloom filter [11] to detect duplicates instead of direct fingerprints comparisons. DDFS uses Stream-Informed Segment Layout technique to take advantage of locality. Using these techniques, DDFS achieves 100MB/sec for single-stream throughput and 210MB/sec for multi-stream throughput.

The above studies have been mainly focused on compression radio and high throughput, not on CPU pressure which can be a big bottleneck in de-duplication storage systems in the near future. Our prototype is also implemented at the block level like DDFS, and use commodity coprocessor to release the CPU pressure.

III. System Architecture

DedupCDP is a data de-duplication CDP system based on Logic Volume Manager version 2 (LVM2). Our system uses SplitDownStream [12] architecture, and transfers data via NBD device. The whole system consists of three parts: backup client, de-duplication sever and backup server. Backup client is the data generating center, it makes a copy of each write request and sends it to the NBD device, and then does the original write operation. The primary task of de-duplication server is to filter duplicate data blocks. After receiving backup data from the backup client, de-duplication server performs the data de-duplication process. It discards the duplicate data, encrypts the metadata and non-duplicate data, then sends them to the backup server. The backup server decrypts the backup data, then stores the metadata and unique data separately.

Here, we detail each component of DedupCDP system (Figure 1): backup client, de-duplication server and backup server.

A. Backup Client

Backup client is running on the machine which requires backup service. It consists of three main modules: user interface module, backup engine module and data transmission module.

The user interface module provides an interface between system and user, including creation and management of CDP volumes. Besides, this module provides access to de-duplication server.

The most important module of the backup client is backup engine module, which runs at the block level, and provides CDP volume backup service. System dispatches fixed-size write requests to the CDP volume. The backup engine backups each data chunk which is written to the CDP volume. The backup engine does the following tasks for every write request to be backed up:

- Capture every write operation BIO to the original volume and make a copy of it.
- Change the target device of the copied BIO to the data transmission module.
- Insert the modified BIO to the list of Remote I/O Request Queue.
- Dispatch the original write request to the original device.

Since the backup engine module and data transmission module are asynchronous, the backup engine only puts the modified data into the remote I/O request queue without waiting for it to be sent, so it can reach a very high throughput. The data transmission module is responsible for sending the backup data to the de-duplication server via NBD device. Backup data are compressed before sending to save bandwidth. Meanwhile, system can get a better compression radio through data compression as well.

B. Deduplication Server

Deduplication server does the following tasks: receiving data de-duplication from backup client, creating index for duplicate data search and metadata for data recovery, and sending the metadata and the unique data to the backup server for storing.

As described in DDFS system, too many disk index searches can decrease system performance. For data coming from backup client, we use Bloom filter to do the data duplication check. However, Bloom filter has some probability of false positive rate, so we have to further check whether the data is duplicate or not. We do this by comparing the fingerprint of the data to be stored with fingerprints stored in system. As indicated in [11], the probability of false positive rate can be calculated in the following formula.

$$\left(1-(1-\frac{1}{m})^{ks}\right)^k \approx \left(1-e^{-\frac{ks}{m}}\right)^k \qquad (1)$$

Given our assumption that hash functions are perfectly random. In our system, the size of the Summary Vector is $80M$ ($m = 80M$) and the number of hash functions is 6 ($k = 6$), the size of test data is about $4G$ ($s = 1M$), so the false positive rata of our system is less than 1%. Most fingerprints have to be stored on disk since memory is not large enough to store all the fingerprints. We store the fingerprints using a hash table on disk for fast search. Fingerprint cache is also organized into a hash table with the same architecture. So when a cache miss occurs, we can easily locate the missing fingerprint on the disk. The cost is only one disk read operation.

Organizing fingerprints in memory and on disk consistently using hash technology results in low cache miss overhead. However, it doesn't guarantee efficient use of cache space. Since the fingerprints are stored in hash table in the order of their hash values, so, when a cache miss occurs, fingerprints with contiguous hash values are fetched

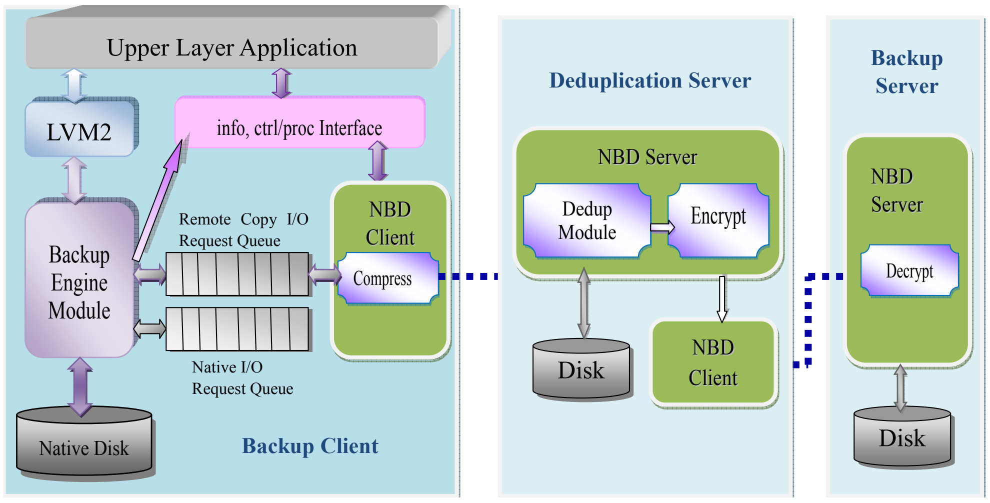

Figure 1. DedupCDP architecture.

into cache through a disk read operation. However, users may not access data chunks in that order. Therefore cache may be inefficient because of bad temporal locality. So we organize fingerprints on disks in two ways. Besides hash organization, we also store fingerprints in a temporal order (their arrival order) and establish a connection between the two organizations.

Hash index: Hash index is for efficient searching. Since fingerprints are used as keywords in disk index access process, we store fingerprints according to their hash values. Our system uses the first n bits of a fingerprint as its hash value to map the fingerprint to its corresponding hash bucket. Each bucket contains $2^{(160-n)}$ fingerprints. On the one hand, the smaller the value of n is, the larger the number of fingerprints in each bucket, therefore, more memory is needed to store each bucket; on the other hand, the larger the value of n is, the less the number of fingerprints in each bucket, the more the times of time-consuming disk index accesses. After a series of experiments, we found that when n is equal to 10, system can get the best performance. When one bucket is full or is replaced, its content will be written to disk. In this way, we keep 2^n bucket lists on disk which contain all data fingerprints stored in the system, as shown in Figure 2.

Our system first looks up the in-memory hash table for its duplicate copy when a new fingerprint comes. If failed, we will turn to the corresponding disk bucket list to look for its duplicate copy. If we found a duplicate one in either of the two processes, the new coming fingerprint is duplicate, otherwise it must be unique and needs to be stored.

Temporal index: Temporal locality index is for improving cache hit ratio. Our metadata is stored according to their arriving order in this structure to take advantage of temporal locality. In order to access temporal list efficiently, the pointer to the location of a fingerprint in the temporal

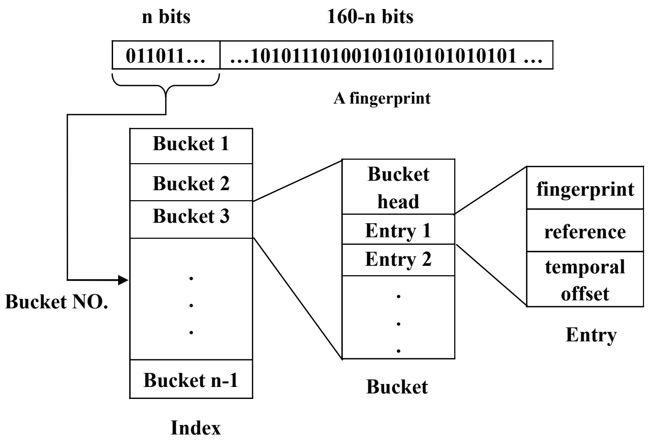

Figure 2. Structure of the disk index.

list is stored in its hash entry.

When we meet a duplicate data block, we can get the location of its duplicate fingerprint in the temporal list from its hash entry and fetch a large number of successive fingerprints to cache (as Figure 3 shows). We can see that only two disk read operations is induced by a cache miss. Moreover, since fingerprints are prefetched according to their arrival order, we will get good cache hit ratio if user access pattern is repetitive.

In addition, we do not directly send the backup data and metadata to the backup server in order to decrease IO time. We temporarily put these data and metadata into a in-memory container which we call segment. When the segment is full, we send the data and the metadata in the segment to the backup server all together.

In conclusion, for backup data chunks received, deduplication server does the following work:

1) Calculate the fingerprint of each data chunk.

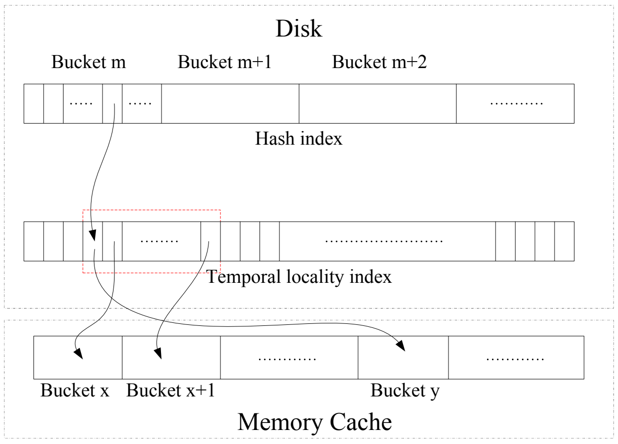

Figure 3. Fingerprint prefetching.

2) Using the Bloom filter vector to determine whether the fingerprint is duplicate, if not, jump to step 6.
3) Search the cache to look for the duplicate fingerprint, if exists, jump to step 7.
4) Use the first n bits of a fingerprint as its hash to map the corresponding bucket and search for the duplicate one, if not exists, jump to step 6.
5) According to the fingerprint record found in step 4, locate the temporal locality index location and read 2M fingerprints following that position to cache, then jump to step 7.
6) Data is unique, put it into data segment.
7) Put it's metadata to metadata segment.
8) Determine whether the data or metadata segment is full, if full, encrypt them and send them to backup server.

C. Backup Server

Backup server is a machine with large capacity of storage, which is used to store the backup data and metadata. When a segment coming, Backup server does the following work: decrypts the data received from de-duplication server, then writes the data and metadata to disk separately.

IV. Fast De-duplication Using Coprocessor

A. Computing Tasks In De-duplication

As described in section III, Deduplication server is the heart of the DedupCDP system. Deduplication server involves not only a great number of disk accesses, but also a lot of computing tasks caused by SHA-1 calculation and AES calculation for each data block coming from backup client. Random disk accesses and too many computing tasks together can bring a heavy burden to the CPU. With the use of new type of storage devices (such as SSD), disk access overhead will be reduced. In contrast, computing task may become the new bottleneck. Therefore CPU's computing power can directly affect system throughput.

Table I
ENERGY COMSUMPTION OF DELL 2850 AND VIA

nergy consumption(W)	*dell 2850*	*VIA*
max	392.57	26.547
min	236.73	18.477

As mentioned in section I, cluster can be used to solve this problem. However, cluster also introduces some problem, such as data synchronization problem, energy problem, cost problem and so on. We find that it is better to solve the computing complexity problem by putting computing task on commodity coprocessor such as GPU, Cell and coprocessor embedded in general-purpose CPU.

B. Accelerating Engines In VIA Platform

In our DedupCDP system, we put computing task on VIA processor to get a better performance. VIA processor is better than common x86 CPUs on aspect of computing tasks for the following reasons:

- VIA processor has a specialized coprocessor called PadLock. It has five accelerating engines: Random Number Generator (RGN), Advanced Cryptography Engine(ACE), Advanced cryptography Engine (version)(ACE2), Hash Engine (PHE) and Montgomery Multiplier (PMM), which cover most common computations in modern storage systems and distributed systems. PHE can accelerate SHA-1 calculations while ACE and ACE2 can accelerate AES calculations. With the help of acceleration engines, VIA processor performs better than common x86 CPUs in SHA-1 and AES calculating;
- Since computing tasks are put on the coprocessor, VIA processor can put more effort to do the data block fingerprints comparison work and disk index access tasks;
- VIA processor's power consumption is much lower than common x86 CPUs used in common PC servers (as Table I shows [13], [14]), which means that the former requires simpler cooling equipment, smaller space and therefore lower cost.

C. Accelerating APIs

VIA only provides a set of assembly instructions to use accelerating engines, so we encapsulate ACE/ACE2 and PHE instructions into two C functions: $VIA\_SHA1()$ and $VIA\_AES()$. Therefore, even novice programmers can use these two engines easily. APIs for other accelerating instructions can be defined in the similar way. In addition, due to the good portability of C language, these APIs can be ported to other system to do the similar work without any change to the interfaces.

Here are the function prototype of $VIA\_SHA1()$ and $VIA\_AES()$, whose parameters are described in Table **??** and Table III.

```
void
VIA_SHA1(long size,
         char *data,
         unsigned int *hash);
void
VIA_AES(char *source,
        char *dest,
        char *key,
        int mode,
        int blocknum,
        int keylen,
        char *itlvector,
        int crypt,
        int keygn,
        int digest);
```

Table II
THE PARAMETERS OF $VIA\_SHA1()$

Parameter	*Description*
$long\ size$	Data size to extract digest
$char\ *data$	Input data to be calculated digest
$int\ *hash$	The output digest

Table III
THE PARAMETERS OF $VIA\_AES()$

Parameter	*Description*
$char\ *source$	Input data to be encrypted/decrypted
$char\ *dest$	Output encrypted/decrypted data
$char\ *key$	Key used in encryption/decryption process
$int\ mode$	Encryption/Decryption mode in AES: ECB/CBC/CTR/CFB/OFB
$int\ blocknum$	The number of data block to be encrypted/decrypted
$char\ *itlvector$	Initail block used in several Encryption/Decryption mode
$int\ crypt$	0-encryption, 1-decryption
$int\ keygn$	How the key generated 0-generated by hardware 1-loaded from memory
$int\ digest$	0-encryption, 1-digest extraction

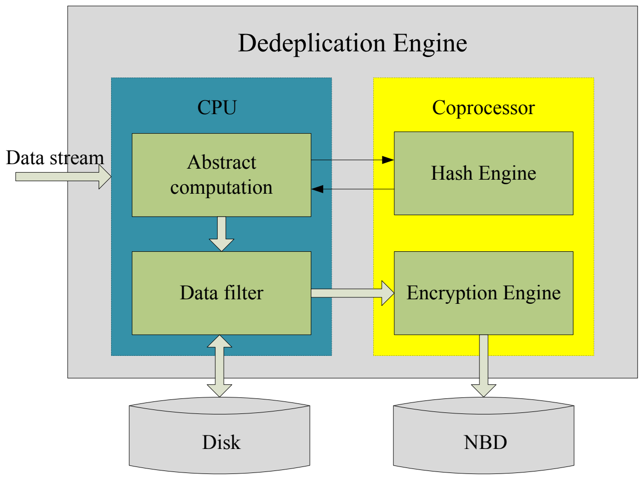

Figure 4. Deduplication in VIA platform.

D. Accelerating Engines In De-duplication

Figure 4 shows the data de-duplication process on VIA platform. When a data stream arrives, Abstract Computation module calculates the fingerprint of a data block with the help of PHE, then forwards the fingerprints to Data Filter module to do data de-duplication work. Then data block is sent to Encryption Engine module to encrypt data on the coprocessor. At last, the encrypted data is sent to the backup server via NBD device. Since the coprocessor does the computing tasks, main processor's burden is greatly reduced.

V. Experiment Results

A. Prototype Implementation

We implemented DedupCDP in the Linux platform. The underlying OS was RedHat AS server 5 (kernel version 2.6.18-128.el5). LVM2 2.02.39, device mapper 1.02.28 and NBD 2.9.12 were used. LZJB algorithm [15] was chosen as the compression/decompression algorithm, SHA-1 was chosen as the fingerprint algorithm, and AES algorithm was chosen as the encryption/decryption algorithm.

B. Experimental Setup

We performed our experiments in two hardware environments, "intel" and "via". In "intel" experiments, the backup client, de-duplication server and backup server were deployed on three single-core 2.66GHz Intel Xeon nodes. Each machine has 4GB of memory and a hardware RAID-0 composed of six disks. The disks in backup client are 37GB SAS disks of 10000 rpm, while disks in de-duplication server and backup server are 74GB disks of 15000 rpm; In "via" experiments, the configuration is same as that in "intel" experiments except that the de-duplication server was on a VIA platform, whose processor is a 1.6GHz VIA Nano. This machine has 2GB of memory and a software RAID-0 composed of two 500GB SATA disks of 7200 rpm. All nodes were connected by a Gigabit Ethernet.

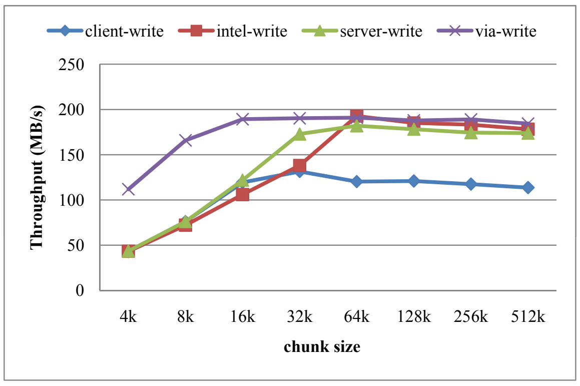

Figure 5. Sequential write.

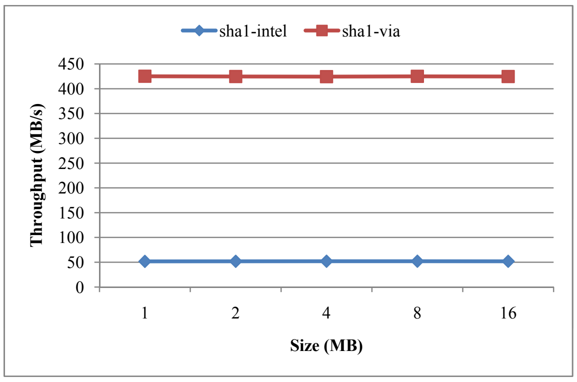

Figure 7. SHA-1 performance.

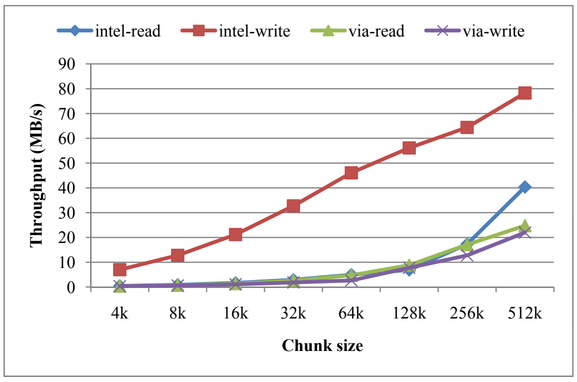

Figure 6. Random access.

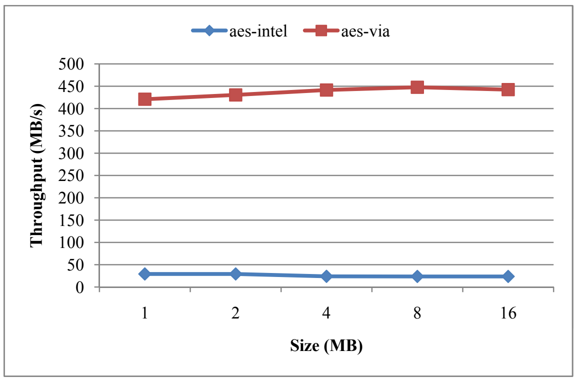

Figure 8. AES performance.

C. *Experimental Results*

1) Storage Subsystem Baseline Test: We first test the disk read and write performance of Intel and VIA platforms. Figure 5 shows the sequential write performance of Intel and VIA platforms, Figure 6 shows the random read and write performance of Intel and VIA platforms. We can see that VIA platform has better sequential read and write performance than Intel platform, while Intel platform has higher random read and write performance than VIA platform. The results are expected, because VIA platform uses less new (fast) disks.

2) Computing Subsystem Baseline Experiment: As mentioned above, we focus on computing complexity problem in our de-duplication system, and commit to improving system performance by reducing computing time. We put computing tasks on PadLock coprocessor to speed up computing tasks. The following three experiments test computing performance of VIA and Intel platforms.

Figure 7 and Figure 8 shows the performance of SHA-1 and AES calculations respectively. We can see that VIA's computing power is far beyond Intel as we expected. We will show that this advantage actually benefits overall system performance in the following subsections.

As we mentioned above, backup data are compressed on backup client server before sent to the deduplication server to save bandwidth. Therefore, choose a good compression algorithm can obviously improve system throughput. We test two most popular compression algorithms to choose a compression algorithm suitable for our system on two data sets. The first data set, which we call "Common", is composed of ten files provided by *www.maimucompression.com*. These files are typical common files. The second data set, which we call "dup", is a log file get from a running Linux system. These files are typical files that have high repetition rate. Figure 9 shows the running time of two algorithms on both data sets. Figure 10 shows the Compression ratio of two algorithms on both data sets. We can see that, for common files, LZJB algorithm is nearly the same as quicklz algorithm [16] in running time and compression ratio. However, for high repetition rate files, LZJB algorithm is much better than quicklz algorithm in both running time and compression ratio. So we choose LZJB algorithm as our compression algorithm.

3) Overall Performance Test: Figure 11 shows the logical size (the amount of data from user or backup application perspective) and the physical size (the amount of data stored in disk) of the system over time at data backup server.

We report results in a 4.4GB data set, which consist of

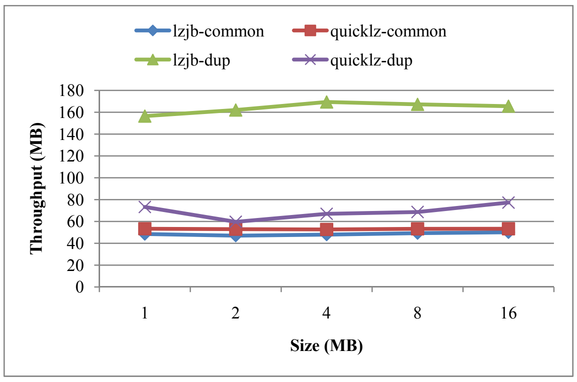

Figure 9. Compression throughput.

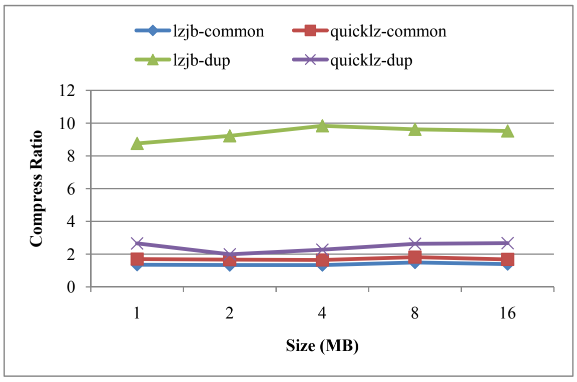

Figure 10. Compression ratio.

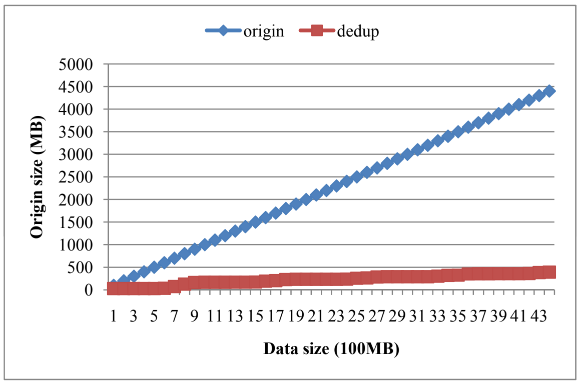

Figure 11. Compression ratio of dedup.

5 dump files from /var/crash directory in a Linux system. Dump files are the records of system memory when the kernel crashed. Different dump files have some repetition ratio and can be used as input data in our system. We input these dump files to backup client according to their generating time to simulate duplicate data generating process.

At the end of the test, the backup server has backed up about 410MB, and the original data size is 4.4GB, reaching a total compression ratio of $10:1$. Figure 12 shows the throughput of the DedupCDP system of both

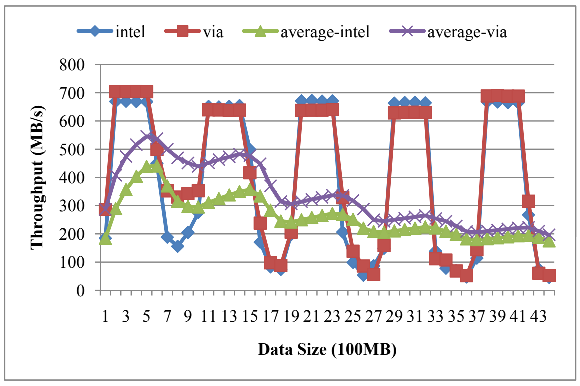

Figure 12. De-duplication throughput.

two kinds of hardware environments. In order to accurately test the performance of data de-duplication, deduplication server writes some information to the log after processing every 100MB data. So we can calculate system throughput using these information. Figure 12 shows both the instantaneous throughput (intel and via) and the average throughput (average-intel and average-via). We can see that the instantaneous throughput curves have obvious peaks and troughs. The peaks corresponding to cache hits, while the troughs corresponding to cache misses (that induces extra disk read operations, therefore the worse performance). We can see that VIA platform achieves a comparable and even better performance than Intel platform. At the end of the test, Intel platform achieves a cumulative de-duplication throughput of 178MB/s while VIA platform achieves a cumulative de-duplication throughput of 192MB/s. Although the disk performance is weak than Intel platform, VIA platform still improve the system performance by about 10%, we can see that the computation tasks occupy a certain proportion in De-duplication systems. This means that we achieve a comparable performance using a lower-energy, cheaper platform.

VI. Conclusions And Future Work

In this paper, we studied the computing tasks in data de-duplication system. We used VIA coprocessor called PadLock to do the computing tasks in data de-duplication system. We implemented a DedupCDP system and tested its performance on Intel and VIA platforms. The experimental results show that with the help of coprocessor, running time of SHA-1 calculation on VIA processor is reduced to $1/8$ than that on Intel CPU, while AES calculation running time is reduced to $1/23$ than that on Intel CPU. This advantage actually benefits the overall performance. VIA platform achieved the comparable throughput with lower energy and cost though its main processor is far interior to Intel CPU.

Our paper provides a framework and basic techniques to build low energy and inexpensive data de-duplication systems. This research can be extended in several directions.

We can use other high performance coprocessors such as GPU to fast de-duplication. We can use more VIA units to distribute the computing tasks in order to get a better performance as well.

ACKNOWLEDGEMENTS

This paper is supported partly by the National High Technology Research and Development Program of China (2008AA01Z401), NSFC of China (60903028), RFDP of China (20070055054), and Science and Technology Development Plan of Tianjin (08JCYBJC13000).

We would like to thank anonymous reviewers for their helpful comments.

REFERENCES

[1] J. Liu, T. Yang, Z. Li, and K. Zhou, "Tspscdp : A timestamp continuous data protection approach based on pipeline strategy," in *Proceedings of the 2008 Japan-China Joint Workshop on Frontier of Computer Science and Technology*, Dec. 2008, pp. 96 – 102.

[2] X. Li, C. Xie, and Q. Yang, "Optimal implementation of continuous data protection (cdp) in linux kernel," in *Proceedings of the 2008 International Conference on Networking, Architecture, and Storage*, Jun. 2008, pp. 28 – 35.

[3] P. Yang, P. Xiao, and J. Ren, "Trap-array : A disk array architecture providing timely recovery to any point-in-time," in *ACM SIGARCH Computer Architecture News*, May 2006, pp. 289–301.

[4] M. Lu, T. cker Chiueh, and S. Lin, "An incremental file system consistency checker for block-level cdp systems," in *Proceedings of the 2008 Symposium on Reliable Distributed Systems*, Washington, DC USA, 2008, pp. 157–162.

[5] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *Proceedings of the 6th USENIX Conference File and Storage Technologies*, California, USA, Feb. 2008, pp. 1–14.

[6] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing : large scale, inline deduplication using sampling and locality," in *Proceedings of the 7th USENIX Conference File and Storage Technologies*, California, USA, Feb. 2009, pp. 111–123.

[7] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge, "Extreme binning: Scalable, parallel deduplication for chunk-based file backup," in *MSCOT'09*, London,UK, Sep. 2009, pp. 1–9.

[8] T. Yang, H. Jiang, D. Feng, and ZhongyingNiu, "Debar: A scalable high-performance de-duplication storage system for backup and archiving."

[9] (2008, Jan.) Emc centera: Content addressed storage system,data sheet. [Online]. Available: http://www.emc.com/collateral/hardware/data-sheet/c931-emc-centera-cas-ds.pdf

[10] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Venti : A new approach to archival storage," in *Proceedings of the Conference File and Storage Technologies*, California, USA, Jan. 2002, pp. 89–101.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," in *Communications of the ACM*, New York, USA, Jul. 1970, pp. 422–426.

[12] G. Laden, P. Ta-Shma, E. Yaffe, P. Factor, and P. Fienblit, "Architectures for controller based cdp," in *Proceedings of the 5th USENIX conference File and Storage Technologies*, California, USA, Feb. 2007, pp. 107–121.

[13] "poweredge 2850," in *http://support.dell.com*.

[14] "Vb8001," in *http://www.via.com.tw*.

[15] (2009) lzjb algorithm. [Online]. Available: http://cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/fs/zfs/lzjb.c

[16] quicklz algorithm. [Online]. Available: http://www.quicklz.com/