Adaptive Pipeline for Deduplication

Jingwei Ma[1], Bin Zhao[1], Gang Wang*[1], Xiaoguang Liu*[2]
[1] *College of I.T., Nankai University, Tianjin, China*
[2] *College of C.S., Nankai University, Tianjin, China*
mjwtom@gmail.com, yunshuishanmu@163.com, {wgzwp, liuxg74}@yahoo.com.cn

***Abstract*—Deduplication has become one of the hottest topics in the field of data storage. Quite a few methods towards reducing disk I/O caused by deduplication have been proposed. Some methods also have been studied to accelerate computational sub-tasks in deduplication. However, the order of computational sub-tasks can affect overall deduplication throughput significantly, because computational sub-tasks exhibit quite different workload and concurrency in different orders and with different data sets. This paper proposes an adaptive pipelining model for the computational sub-tasks in deduplication. It takes both data type and hardware platform into account. Taking the compression ratio and the duplicate ratio of the data stream, and the compression speed and the fingerprinting speed on different processing units as parameters, it determines the optimal order of the pipeline stages (computational sub-tasks) and assigns each stage to the processing unit which processes it fastest. That is, "adaptive" refers to both data adaptive and hardware adaptive. Experimental results show that the adaptive pipeline improves the deduplication throughput up to 50% compared with the plain fixed pipeline, which implies that it is suitable for simultaneous deduplication of various data types on modern heterogeneous multi-core systems.**

I. Introduction

The digital universe cracked the zettabyte barrier in 2010 [6]. In 2011, the amount of information created and replicated will surpass 1.8 zettabytes (1.8 trillion gigabytes) - growing by a factor of 9 in just five years.

To mitigate storage cost of such huge volumes of data, data deduplication is exploited, which identifies duplicated data and eliminates it to save storage space. The storage space can be reduced by a factor of 10 to 20 [1] with deduplication.

To avoiding the performance degradation caused by extra deduplication step. Many methods have been proposed to eliminate I/O bottleneck and to accelerate the computational sub-tasks. For example, Bloom Filter [2], Sparse Indexing [9], Extreme Binning [3] and Cook Hash [12] [5] are proposed to reduce disk I/O. Data Domain File System (DDFS) [15] combined Bloom Filter, Stream-Informed Segment Layout (SISL) and Locality Preserved Cache (LPC) together and reduced the disk I/O to 1%. In addition, GPU [8] and PadLock engine [10] are used to accelerate hash calculation and encryption in deduplication systems. To build a high performance deduplication system, pipeline can exploit currency among computational sub-tasks, and therefore make the most of computing resources [7].

However, few notice another important factor on deduplication throughput - the order of the computational sub-tasks. Since the output of the predecessor sub-task is the input of the successor sub-task, different orders may yield very different performance. For example, putting compression after duplication identification is better for highly duplicate data sets and the reverse order may be better for other kinds of data. However, neither order can defeat the other in all situations. Also, a general-purpose CPU may compress a data block much faster than calculate its SHA-1 digest, while PadLock engine [14] in a VIA CPU is just the opposite. An adaptive method is promising for this problem.

This paper proposes an adaptive pipelining model for deduplication. The order of stages is arranged according to the data type and the execution time of each stage on different processing units. We tested the model in our deduplication prototype. The results show that the adaptive pipeline improves throughput up to 50% compared with the plain fixed pipeline.

II. Related Work

Data deduplication was proposed mainly to solve the storage and network overhead, so early researches usually focused on compression ratio rather than throughput [13]. However, the throughput is low due to disk bottleneck.

Data Domain File System (DDFS) [15] is one of the earliest studies that try to solve the disk bottleneck problem. It used Bloom Filter, Stream-Informed Segment Layout and Locality Preserved Cache to reduce disk I/O. With the above methods, 99% of disk I/O was reduced. Sparse Indexing [9] was also used to eliminate the disk bottleneck. The data was divided into segments and fingerprints are sampled. According to the similarity of the sample fingerprints, several champion segments are selected to compare with a new incoming segment. This method reduces the RAM to disk ratio and also the disk accesses.

To make full use of the resources, Guo *et al.* [7] presented a modular, event-driven, client pipeline design for *source deduplication* [11]. It can achieve high backup throughput (1 GB/sec for unique data and 6 GB/sec for duplicate data) and restore throughput (1 GB/sec for single stream and 430 MB/sec for multiple streams) and good deduplication efficiency (97%), at high capacities (123 billion objects, 500 TB of data per 25 GB of system memory).

The computational sub-tasks in deduplication also seriously affect the performance of the system. Ma *et al.* [10] used Padlock engine in VIA CPU to accelerate SHA-1 and AES calculation in deduplication systems and got an ideal throughput on low power consumption platform. The throughput per watt was improved up to 15 times. Li *et al.* [8] used GPU to accelerate hash calculation in deduplication and with the help of GPU, the hash throughput is improved remarkably.

These researches mainly focused on optimizing single computational sub-tasks in deduplication and ignored the relationship among these sub-tasks. For example, DDFS and Venti put compression after duplications having been detected. This is suitable for data sets with high duplicate ratio, but not suitable for all situations. For data sets with high compression ratio and relative lower duplicate ratio, the reverse order may be better.

Increasingly, heterogeneous multi-core platforms are deployed. Nowadays, deduplication systems typically run on multi-core platforms. Considering this kind of platform is composed of different processing units, which run deduplication sub-tasks at different speeds, the problem becomes further complicated. Our adaptive pipelining model takes the characteristics of both data type and hardware platform into account. The model considers two cases, in which all of the pipeline stages are the same size or not, and chooses different strategies to assemble the optimal pipeline in both cases.

III. Adaptive Pipelining Model

During one deduplication procedure, typically the computational sub-tasks shown below are performed [15] [13]. First, a hash value is calculated as the unique fingerprint of a chunk or a file. Then the fingerprint is compared with previous ones to determine if the chunk is unique. These two sub-tasks are the basic parts of deduplication. Usually some other parts are added to meet different needs. Compression is often performed to get a further storage space saving. If a remote disk is used to store the data, the data will be encrypted for safety.

Duplication identification must be performed after fingerprinting since the input of the latter is the output of the former. Encryption must be performed after deduplication identification to avoid encrypting duplicate data. The reverse order inevitably causes performance degradation. Similarly, encryption and compression also have the partial order relation. However, the optimal order of some sub-tasks depends on the type of data to be deduplicated and the hardware platform. Compression can be put either before or after duplication identification. For the "compression ahead" order, the amount of work of hash calculation is reduced. For the "compression behind" order, duplicate data compression is avoid. Neither order can defeat the other in all situations. So we come up with two orders of computational sub-tasks in deduplication. One is Compression$\rightarrow$ Hash calculation$\rightarrow$ Duplication Identification$\rightarrow$ Encryption (CHIE for short). The other is Hash calculation$\rightarrow$ Duplication Identification$\rightarrow$ Compression$\rightarrow$ Encryption (HICE for short). To make full use of multi-core platforms, computational sub-tasks are organized into a pipeline [7]. Then we have two types of pipeline, CHIE and HICE.

We build a quantitative model to depict the deduplication pipeline and determine the optimal pipeline order according to some parameters of data set and hardware platform. In the rest of this section, we will introduce the adaptive pipelining model in detail. Although our discussion and experiments are both based on pipelined deduplication systems, the adaptive model is also fit for serial systems.

A. Pipeline Balance

The degree of inequality of stage size affects the pipeline throughput seriously. The speed of a pipeline is limited by the slowest stage. So we consider two situations: the *balanced pipeline*, in which all of the stages are almost the same size; and the *unbalanced pipeline*, in which some stages are much bigger than others. For a balanced pipeline, we can obtain the optimal throughput by simply distributing stages to processing units evenly. The processing units are rarely idle during the running of the pipeline. However, for a unbalanced pipeline, this task distribution will cause some processing units to spend a large part of time in idling. To obtain throughput close to the optimal, we must eliminates idling by some techniques, such as pipeline reordering, sub-tasks combination, and multithread.

Fig. 1 shows a balanced pipeline composed of three stages. Although it is run on a machine with only two cores rather than three, load balance is still easy to reach since the three stages are all the same size. Suppose that the execution time of the first stage on a data chunk is T_0 (it is just the size of the first stage), and those of the second and the third stages are T_1 and T_2 respectively. The total amount of work the pipeline done, or equivalently, the total time spent by the two cores, is $T_0+T_1+T_2$. Since the stages are distributed between the two cores evenly, the average processing time of a data chunk is $\frac{T_0+T_1+T_2}{2}$. More generally, for a pipeline composed of M stages running on a machine with N cores, and the execution time of the i-th stage is T_i, for $0 \leq i < M$, the average processing time of a data chunk is $\frac{\sum_{i=0}^{M-1} T_i}{N}$.

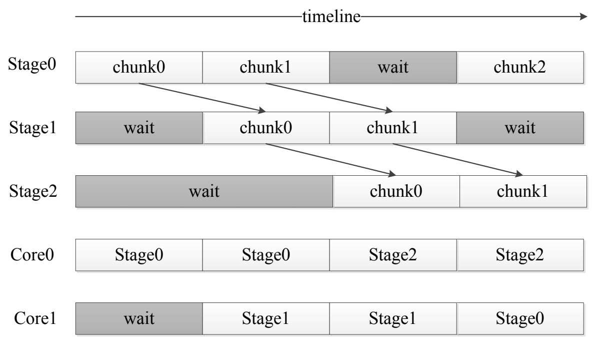

Fig. 1. Balanced Pipeline

An unbalanced pipeline is shown in Fig. 2. The light grids denote effective computation time, and the dark grids denote idle time caused by waiting between stages. The first stage spend much longer time than the other two. It limits the speed of the whole pipeline.

B. Notations

Before give our adaptive pipelining model, we assign the variables:

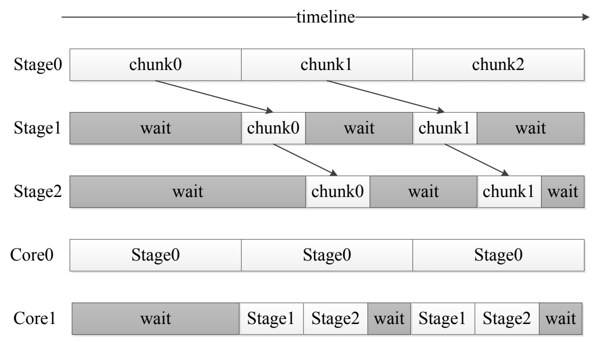

Fig. 2. Unbalanced Pipeline

- S is the input data size.
- S_C is the size of data to be compressed.
- S_H is the size of data to be fingerprinted.
- R_D is the duplicate ratio of data.
- R_C is the compression ratio of data.
- C is the throughput of compression.
- H is the throughput of hash calculation (fingerprinting).
- T_C is the compression time.
- T_H is the hash time.
- T_I is the identification time.
- T_E is the encryption time.
- T^S_{CHIE} is the serial execution time of the CHIE pipeline.
- T^S_{HICE} is the serial execution time of the HICE pipeline.
- N is the number of processing units.
- T^P_{CHIE} is the parallel execution time of the CHIE pipeline.
- T^P_{HICE} is the parallel execution time of the HICE pipeline.

C. Model for Balanced Pipeline

For a balanced pipeline, the serial execution time is the simple sum of the execution times of all stages. Since the serial execution time can be regarded as the total time spent by all processing units in a parallel execution, the parallel execution time is just the serial execution time divided by N.

Common compression, hash calculation and encryption algorithms used in deduplication are linear time algorithms. For example, LZJB, the compression algorithm used in our deduplication prototype is a linear time algorithm. Therefore, the time spent in compression in a CHIE pipeline is

$$T_C = \frac{S}{C} \tag{1}$$

The amount of data to be compressed is S since compression is the first step. Similarly, we have the hash calculation time

$$T_H = \frac{S_H}{H} = \frac{SR_C}{H} \tag{2}$$

The serial time of a CHIE pipeline is

$$\begin{aligned} T^S_{CHIE} &= T_C + T_H + T_I + T_E \\ &= \frac{S}{C} + \frac{SR_C}{H} + T_I + T_E \end{aligned} \tag{3}$$

The parallel execution time is $T^P_{CHIE} \approx \frac{T^S_{CHIE}}{N}$. Since there are start time and stop time, the parallel time of a pipeline is not exactly equal to the serial time divided by N. However, the start and stop time are negligible if the data set is large enough. So we can replace the approximately equal sign by a equal sign. In addition, if multithreading technique is used, the parallel time can be exactly equal to the serial time divided by N.

We can analyze the execution time for a HICE pipeline similarly. The hash calculation time is

$$T'_H = \frac{S}{H} \tag{4}$$

Since some duplicate data chunks are discarded, the size of data to be compressed is $S_C = S(1-R_D)$, and the compression time is

$$T'_C = \frac{S_C}{C} = \frac{S(1-R_D)}{C} \tag{5}$$

The size of fingerprints is independent of the order of the pipeline. So the size of fingerprints is the same as that in CHIE pipeline. So does the duplication identification time. Encryption is still performed on compressed unique data chunks, so the encryption time is the same as type pipeline type of CHIE. Therefore, the serial execution time of a HICE pipeline is

$$\begin{aligned} T^S_{HICE} &= T'_H + T'_C + T_I + T_E \\ &= \frac{S}{H} + \frac{S(1-R_D)}{C} + T_I + T_E \end{aligned} \tag{6}$$

The parallel execution time is $T^P_{HICE} \approx \frac{T^S_{HICE}}{N}$.

To determine the optimal pipeline order, we can simply compare formula (3) and formula (6). If $T^S_{CHIE} < T^S_{HICE}$, the CHIE pipeline is the optimal order. This inequality is simplified as

$$\begin{aligned} &\frac{S}{C} + \frac{SR_C}{H} < \frac{S}{H} + \frac{S(1-R_D)}{C} \\ &H + CR_C < C + H(1-R_D) \\ &\frac{R_D H}{(1-R_C)C} < 1 \end{aligned} \tag{7}$$

The HICE pipeline is selected if the following inequality is satisfied.

$$\frac{R_D H}{(1-R_C)C} > 1 \tag{8}$$

For a serial deduplication system, the execution time is the sum of the time spent by all the stages whether the stages are balanced or not. This is similar to the balanced situation except that it doesn't need to be divided by N. So this model can be applied to both serial and balanced pipelining deduplication systems. It selects the optimal order of the pipeline according to the compression ratio and the duplicate ratio of the data set, and the compression speed and the hash calculation speed on the given hardware platform. These parameters can be estimated by sampling the data set. If the distribution of these parameters is uniform, the estimation can guides the selection of pipeline order effectively.

D. Model for Unbalanced Pipeline

For a pipeline composed of unequal stages, it is a little hard to analyze precise workload and parallel execution time on a multi-core platform. However, if we suppose that the hardware platform equipped with enough processing units so that each stage is run by an unique processing unit, we can approximate the parallel execution time of an unbalanced pipeline by the time of the slowest stage.

In deduplication systems, hash calculation and compression generally spend much more time than other sub-tasks. One of them often is the slowest sub-task. Because of the relationship of these two sub-tasks, we can improve the throughput by reorganizing the pipeline.

1) Hash Bottleneck: If hash calculation is the slowest stage in deduplication pipeline, it will block the pipeline. Suppose that the hash bottleneck is caused by the hash throughput H is remarkably lower than the throughput of any other stage (for example compression throughput C) rather than other reasons. (such as fast compression caused by relatively high duplicate ratio.) In this case, different orders still exhibit different performance. If hash calculation is the first stage, the worst throughput is gained since the amount of work of hash calculation S_H is maximized to S, that is the bottleneck of the pipeline is maximized. The parallel execution time is

$$T^P_{HICE} = T_H = \frac{S}{H} \tag{9}$$

Compression reduce the amount of data. So we put the compression stage ahead to reduce the amount of work of hash calculation. Since the hash calculation time is shortened, another stage may become the slowest one. This new bottleneck usually will be compression, so we take it as an example. Therefore, the execution time of the pipeline is just the compression time

$$T^P_{CHIE} = T_C = \frac{S}{C} \tag{10}$$

Since we suppose that $H < C$, we have $T_H > T_C$. Therefore, the execution time is reduced.

If the hash calculation is still the slowest stage, the overall throughput is also improved since we improve the hash calculation performance by reducing the amount of data to be processed. The execution time of the pipeline is equal to the execution time of the postpositive hash calculation stage

$$T^P_{CHIE} = T'_H = \frac{S_H}{H} = \frac{SR_C}{H} \tag{11}$$

2) Compression Bottleneck: If compression is the slowest stage in the deduplication pipeline and the HICE order is used, we can not improve compression performance by reorder the pipeline. So we only consider the CHIE order. The execution time of the pipeline is equal to the compression time

$$T^P_{CHIE} = T_C = \frac{S}{C} \tag{12}$$

We also suppose that the compression bottleneck is caused by the compression throughput C is remarkably lower than the throughput of any other stage (for example hash throughput H). Since putting duplication identification before compression can avoid compressing duplicate data chunks, we try to improve throughput by putting hash calculation and duplication identification ahead, that is, use the HICE order instead of the CHIE order. The rest of the analysis is similar to the case of hash bottleneck. After putting hash calculation ahead, if another stage becomes the slowest one, the speed of the pipeline is equal to the speed of the new bottleneck. However, the new bottleneck is faster than the original compression stage, the pipeline throughput is improved. This new bottleneck usually will be hash calculation, so we take it as an example. The execution time of the pipeline is

$$T^P_{HICE} = T_H = \frac{S}{H} \tag{13}$$

If compression is still the slowest stage, the pipeline is also speeded up since the duplicated chunks are discarded and avoid being compressed. The execution time of the pipeline is

$$T^P_{HICE} = T'_C = \frac{S(1-R_D)}{C} \tag{14}$$

3) Other Bottleneck: Suppose that neither compression nor hash calculation is the bottleneck of the pipeline. Although this case is not common, we still take it into account. Since the reconfiguration of the pipeline only affects the amount of work of hash calculation and compression, it may not improve the throughput of the pipeline. Moreover, we should avoid introducing a more serious new bottleneck when try to optimize an old bottleneck. So we put the relative faster one of hash calculation and compression ahead.

Based on the above analysis, we can give a unified criterion of the selection of the pipeline order for all three cases - just putting the relative faster one of hash calculation and compression ahead. We choose pipeline type CHIE when $C > H$ and otherwise HICE.

IV. Implementation & Experimental Results

We implement our system in Linux operating system and we choose *target deduplication architecture* [11] in our system. We created five threads to complete the system. They are responsible for receiving data, hash calculation, compression, duplication identification and encryption. For special hardware platform, we take use of available co-processors to accelerate the computational sub-tasks. The communication among the threads is data transfer. Since shared memory model is used, data transfer is very fast.

A. Computational Sub-task Acceleration

To our observation, compression is often the most time consuming stage in the system. So we put the compression subtask on GPU if possible. The GPU algorithm first accumulates a number of data chunks into a batch. Then uploads the batch to the GPU [4]. GPU can compress those chunks in parallel because there is no dependence among them. We obtain a high compression throughput using GPU although CPU-GPU transmission overhead is introduced.

On platform with VIA CPU, the PadLock engine is exploited to accelerate SHA-1 and AES calculations.

B. Hardware Platforms and Data Types

Two hardware platforms and three data types are selected for the test. They are detailed as follows.

One platform is called 'VIA'. It is equipped with a VIA Nano processor L2200@1600MHz, 2GB RAM, two 32GB SATA II SSDs, a NVIDIA GTX 480 GPU, CUDA version 3.0. The operating system is 64-bit Redhat Linux AS 5 with kernel 2.6.18.

The other is called 'AMD'. It is equipped with a quad-core AMD Phenom(tm) II X4 945 Processor, 4GB RAM, one 500GB 7200 rpm SATA disk, a NVIDIA GTX 480 GPU, CUDA version 4.0. The operating system is the same as 'VIA'.

We name the platform in the test results with the hardware it used. 'GPU' indicates compression is done on the GPU. 'PadLock' indicates SHA-1 & AES calculations are performed by the PadLock engine.

Three data types are tested on each platform. One data type called 'MIRROR' consists of 4 one-GB SnapShots of a linux operating system. Another data type called 'SVN' is made up with 4 versions of data got from a subversion server, which is an open source version control system. It is of 4.5 GB. The last called 'KERNEL' is linux kernel source code from version 2.6.27.59 to version 3.1-rc4 downloaded from www.kernel.org. It is extracted and repacked into a tar file without compression.

C. Throughput of Sub-tasks, Compression Ratio and Duplicate Ratio

For different platforms and different data types, the throughput of compression is different. TABLE I shows the compression throughput of different platforms and different data types.

TABLE II shows the SHA-1 throughput of different platforms. SHA-1 calculation is only hardware platform related.

TABLE III shows the duplicate ratio and compression ratio of each data type. Because we take fixed-size chunk method, the duplicate ratio is not so high as variable-size chunk method.

D. Throughput Results Analysis

TABLE IV organizes all the parameters and the throughput of each order of pipeline on different hardware platforms and data types together.

TABLE I
COMPRESSION THROUGHPUT

Platform	Data Set	Throughput(MB/s)
VIA	MIRROR	84.22
AMD	MIRROR	237.75
VIA-GPU	MIRROR	172.22
AMD-GPU	MIRROR	297.09
VIA	SVN	62.53
AMD	SVN	154.70
VIA-GPU	SVN	121.29
AMD-GPU	SVN	166.59
VIA	KERNEL	64.70
AMD	KERNEL	180.81
VIA-GPU	KERNEL	161.74
AMD-GPU	KERNEL	264.61

TABLE II
SHA-1 THROUGHPUT

Platform	Throughput(MB/s)
VIA	62.88
AMD	201.88
PadLock	299.72

TABLE III
DATA DUPLICATE RATIO & COMPRESSION RATIO

Data Set	Duplicate Ratio	Compression Ratio
MIRROR	0.55	0.40
SVN	0.41	0.94
KERNEL	0.10	0.54

1) Balanced Pipeline: Since VIA CPU is a single-core CPU. On 'VIA' platform, these stages run sequentially and it's suitable for the balanced situation. So we compare the value of $\frac{R_D H}{(1-R_C)C}$ to 1. From the table we can see that if the value of $\frac{R_D H}{(1-R_C)C}$ is below 1 the pipeline type of CHIE is faster, and otherwise, pipeline type of HICE is faster. The result is consistent with our model.

2) Unbalanced Pipeline: AMD CPU is a multi-core CPU and the CPU resource is enough to let the threads run in parallel. So it's suitable for the unbalanced situation. So we just compare the throughput of hash calculation and compression. From the table, the throughput of pipeline type of CHIE is faster than the pipeline type of HICE if compression is faster than hash calculation, and the throughput of pipeline type of HICE is faster than the pipeline type of CHIE if compression is slower than hash calculation. It agrees with our model very well.

V. CONCLUSION

In this paper, we studied the relationship of the computational sub-tasks in deduplcation. An adaptive pipelining model for deduplication was developed. For different hardware platforms and data types, the model can determine the optimal order of the sub-tasks in the pipeline. It can make full use of the hardware platform and characteristics of the data type. We tested our model on two hardware platforms and three data types. The experimental results show that our model is effective to choose an optimal order of the computational sub-tasks.

TABLE IV
MODEL VALIDATION

Platform	Data Type	Compression Throughput (MB/s)	Hash Throughput (MB/s)	R_C	R_D	$\frac{R_D H}{(1-R_C)C}$	CHIE Throughput (MB/s)	HICE Throughput (MB/s)
VIA	MIRROR	84.22	62.88	0.40	0.55	0.68	53.26	45.62
VIA	SVN	62.53	62.88	0.94	0.41	6.87	32.69	36.18
VIA	KERNEL	64.70	62.88	0.54	0.10	0.21	40.67	33.53
PadLock	MIRROR	84.22	299.72	0.40	0.55	3.26	71.49	96.28
PadLock	SVN	62.53	299.72	0.94	0.41	32.75	50.95	61.21
PadLock	KERNEL	64.70	299.72	0.54	0.10	1.01	54.94	54.69
VIA-GPU	MIRROR	172.22	62.88	0.40	0.55	0.33	71.15	50.77
VIA-GPU	SVN	121.29	62.88	0.94	0.41	3.54	39.95	42.87
VIA-GPU	KERNEL	161.74	62.88	0.54	0.10	0.08	58.85	44.20
PadLock-GPU	MIRROR	172.22	299.72	0.40	0.55	1.60	109.67	123.01
PadLock-GPU	SVN	121.29	299.72	0.94	0.41	16.89	73.48	83.99
PadLock-GPU	KERNEL	161.74	299.72	0.54	0.10	0.40	97.78	89.86
AMD	MIRROR	237.75	201.88	-	-	-	232.29	192.75
AMD	SVN	154.70	201.88	-	-	-	144.49	148.31
AMD	KERNEL	180.81	201.88	-	-	-	178.09	189.79
AMD-GPU	MIRROR	297.09	201.88	-	-	-	287.57	190.59
AMD-GPU	SVN	166.59	201.88	-	-	-	147.24	152.09
AMD-GPU	KERNEL	264.61	201.88	-	-	-	219.35	173.67

Note that the parameters used in our tests are given by pre-tests. Moreover, we only consider single data stream in a test. However, mixture of multiple data streams is common in a real system. So effective dynamic parameter sampler and data stream separator are important future research topics. In addition, how to avoid cache migration caused by sub-tasks exchange between processing units (such as between CPU and GPU) is also an interesting problem.

ACKNOWLEDGMENT

This paper is partially supported by NSFC of China (60903028, 61070014), Key Projects in the Tianjin Science & Technology Pillar Program (11ZCKFGX01100).

REFERENCES

[1] T. ASARO and H. BIGGAR, "Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements," *The Enterprise Strategy Group*, 2007.

[2] B. B.H., "Space/time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[3] D. Bhagwat, K. Eshghi, D. Long, and M. Lillibridge, "Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup," in *IEEE 2009 International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2009, pp. 1–9.

[4] L. Costa, S. Al-Kiswany, and M. Ripeanu, "GPU Support for Batch Oriented Workloads," in *IEEE 28th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2009, pp. 231–238.

[5] B. Debnath, S. Sengupta, and J. Li, "Chunkstash: Speeding up inline storage deduplication using flash memory," in *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USENIX Association, 2010, pp. 16–16.

[6] J. Gantz and D. Reinsel, "Extracting Value from Chaos," *IDC research report IDC research report, Framingham, MA, June. Retrieved September*, vol. 19, 2011.

[7] F. Guo and P. Efstathopoulos, "Building a High-performance Deduplication System," in *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*. USENIX Association, 2011, pp. 25–25.

[8] X. Li and D. Lilja, "A Highly Parallel GPU-based Hash Accelerator for a Data Deduplication System," in *Parallel and Distributed Computing and Systems*. ACTA Press, 2009.

[9] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality," in *Proccedings of the 7th Conference on File and Storage Technologies (FAST)*. USENIX Association, 2009, pp. 111–123.

[10] L. Ma, C. Zhen, B. Zhao, J. Ma, G. Wang, and X. Liu, "Towards Fast Deduplication Using Low Energy Coprocessor," in *2010 Fifth International Conference on Networking, Architecture and Storage (NAS)*. IEEE, 2010, pp. 395–402.

[11] S. Maddodi, G. Attigeri, and A. Karunakar, "Data deduplication techniques and analysis," in *The 3rd International Conference on Emerging Trends in Engineering and Technology (ICETET)*. IEEE, 2010, pp. 664–668.

[12] R. Pagh and F. Rodler, "Cuckoo Hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

[13] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Data Storage," in *Proceedings of the USENIX 2002 Conference on File and Storage Technologies (FAST)*, vol. 4, 2002, pp. 89–102.

[14] VIA Technologies, Inc., "VIA Nano Processor," *http://www.viatech.com.cn/cn/downloads/whitepapers/processors/WP080529VIA_Nano.pdf*, 2008.

[15] 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 on File and Storage Technologies (FAST)*. USENIX Association, 2008, pp. 269–282.