GHOST: GPGPU-Offloaded High Performance Storage I/O Deduplication for Primary Storage System

Chulmin Kim
KAIST, Electical Engineering
Daejeon, South Korea
cmkim@core.kaist.ac.kr

Ki-Woong Park
KAIST, Electical Engineering
Daejeon, South Korea
woongbak@core.kaist.ac.kr

Kyu Ho Park
KAIST, Electical Engineering
Daejeon, South Korea
kpark@kaist.ac.kr

ABSTRACT

Data deduplication has been an effective way to eliminate redundant data mainly for backup storage systems. Since the recent primary storage systems in cloud services are expected to have the redundancy, the deduplication technique can also bring significant cost saving for the primary storage. However, the primary storage system requires high performance requirement about several GBs per second. Most conventional deduplication techniques targeted the performance requirement of 200-300MB/s.

In an attempt to achieve a high performance storage deduplication system at the primary storage, we thoroughly analyze the performance bottleneck of previous deduplication systems to enhance the system to meet the requirement of the primary storage. The new performance bottleneck of deduplication in the primary storage lies on not only key-value store lookup, also computation for data segmentation and fingerprinting due to recent technology improvement of flash devices such as SSD. To overcome the bottlenecks, we propose a new deduplication system utilizing GPGPU. Our proposed system, termed GHOST, includes the followings to offload and optimize the deduplication processing in GPGPU: (1) In-Host Data Cache, (2) Destage-aware Data offloading to GPGPU and (3) In-GPGPU Table Cache of key-value store. These techniques improve the offloaded deduplication processing about 10-20% on the reasonable workload of the primary storage compared to the naive approach. Our proposed deduplication system can achieve 1.5GB/s in maximum which is about 5 times of the deduplication systems used CPU only.

Categories and Subject Descriptors

H.3 Information Systems [**INFORMATION STORAGE AND RETRIEVAL**]: General

General Terms

Design, Performance

PMAM 2012, February 26, 2012 New Orleans LA, USA

Keywords

Deduplication, GPGPU, Primary Storage

1. INTRODUCTION

Cloud computing [1, 26, 16] is an evolving computing paradigm that is being a trend of the recent computing industry. Basically, the cloud system earns its profit by delivering resources such as computing and storage to customers on demand. Among the computing resources, storage space is becoming the most serious problem due to its maintenance cost. Whole storage space should be maintained even during the users are offline while other resources of the offline users can be passed to other online users. Moreover, MTTF(Mean Time To Failure) of storage devices such as HDD is very short compared to other resources [25].

Storage deduplication [22] maintains only one unique copy of same data chunks. It can bring huge benefit to the cloud providers with large primary storage servers suffering the storage cost problem. However, existing deduplication systems [29] have targeted mainly the backup storage so far. There were two reasons. First, the overhead of storage deduplication was not negligible. Storage deduplication system should maintain a table called key-value store [6]. Due to the big table size which does not fit into the main memory, table lookups required burden disk accesses. This problem limits the maximum performance of the entire system. Even though there was trials to reduce this overhead [29, 15], they report 200-300MB/s as the maximum throughput of the system which is less than the maximum write throughput of just one recent SSD device about 320MB/s [24]. Second, the benefit of deduplication on the primary storage was forecasted not to be high enough. Before the cloud system, the primary storage was used for the databases of services which are oriented more on read operations and have less possibility of duplicated data. In that kind of systems, the deduplication is expected only to save little storage space costing huge part of the throughput.

Currently, the reasons in the past have resolved almost by the significantly evolved technology and computing trend. The main memory size of server systems for the primary storage usually reaches to tenth GBs in these days. It will decrease the possibility of disk accesses due to the table lookup. Appearance of Flash-based device such as SSD with fast random read latency also mitigated the time taken for disk accesses during table lookups. Not only it mitigated the overhead, also it makes the deduplication more precious in the primary storage where adopted SSDs as the storage media for the performance reason. Since the cost of SSD

is higher than that of HDD, the amount of cost reduced by the saved space will get larger. Cloud computing trend in which an individual user rents the virtual machine from the cloud system is enlightening the possibility of high deduplication rate in storage system of the cloud system. Multiple individual users will probably install same OSes and applications, and share the same files such as document, music or video files. The recent observation [18] about the practical deduplication in the individual computing environment shows the high deduplication rate within a set of multiple users (50% deduplication rate in a set of 819 users).

In the phase of writing to the storage media, adoption of only four SSDs can achieve the performance of the primary storage up to several GBs per second since each SSD can perform above 320MB/s for sequential write and 520MB/s for sequential read [24]. However, during the deduplication processing, the system meets the bottlenecks again from the data segmentation and fingerprinting. This computational overhead was also existed in the past , but it was hidden by the bottleneck in the table lookup. Now, it is being unveiled because the table lookup overhead is mitigated by SSD and enlarged main memory space. The data segmentation and fingerprinting of the deduplication processing can be executed in parallel. Therefore, multi-threaded processing will be the solution for these bottlenecks. As known generally, there are two choices of multithreaded processing: Multicore CPUs and GPGPU(General Purpose Graphic Processing Unit). By comparing both approaches through the preliminary experiments, we figured out that GPGPU is more favorable to the deduplication in the perspective of cost effectiveness.

In this paper, we propose a GPGPU offloaded deduplication system, termed GHOST. To design it, we examine whether each part of typical deduplication processing fits into GPGPU or not. Following the examination, our system fully offloads the deduplication processing to GPGPU to utilize its computing power and broad memory access bandwidth.

First of all, we suggest the system with In-Host Data Cache (IHD-Cache) to enable the offloading without loss of performance. It is responsible for being the buffer of data to be offloaded rapidly and the traditional functionality of a write/read cache for the primary storage.

Due to the long latency of parallel deduplication operations offloaded to GPGPU(though the throughput is incredible), our system in the primary storage might suffer invalidation of deduplication processing due to the events such as frequent write cache hits. To avoid the deduplication result being useless due to such events, our system includes Destage-aware Data Offloading to GPGPU. It selects data from the collaboration with the employed destaging algorithm which forecasts the data not to have write cache hit based on the spatial and temporal locality [11, 10]. (destage : write data from cache to underlying disks)

Moreover, we maintain In-GPGPU Table Cache (IGT-Cache) which is the hot part of the whole table entries with more possibility of deduplication to utilize the broad memory access bandwidth of GPGPU. It can reduce much of table lookup overhead in proportional to the deduplication ratio in an optimistic view. In addition, we adopt *summary vector* [29] for IGT-Cache to mitigate the overhead of the table cache lookup. It reduces uncoalesced memory access pattern caused by the unnecessary lookups on IGT-Cache.

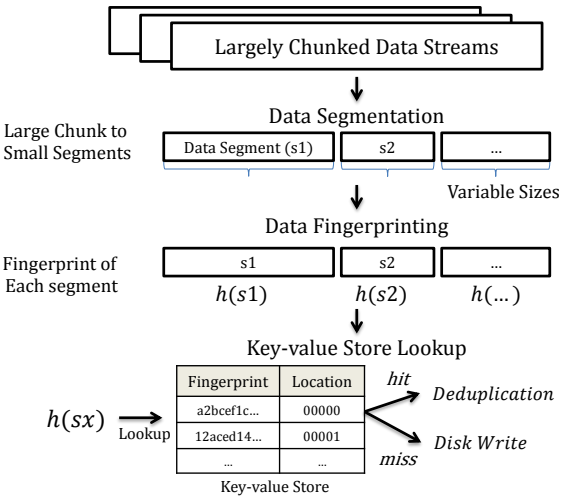

Figure 1: Typical Deduplication Processing

GHOST is a part of our development project for a manycore-based cloud computing platform, called MN-MATE [21]. In our platform, GHOST is integrated into storage layer as the key primitive for a high performance storage deduplication. Consequently, GHOST can also be extended to cloud-based services such as an outsourced storage service and a content delivery service over the Internet.

2. BACKGROUND

2.1 Storage Deduplication

Generally, the deduplication process consists of the steps shown in Figure 1. First of all, an largely chunked incoming data stream is divided into multiple pieces of data segments. The size of each piece can be variable or fixed. Variable-length segmentation [23] needs more computation, instead, it brings higher deduplication rate than those of fixed-length segmentation [18].

Secondly, each segment should have its own fingerprint to verify the uniqueness. For the purpose, one of various hash algorithms [7, 17] can be adopted to generate the fingerprint. The goal of the hash algorithms in fingerprinting is to avoid fingerprint collision between different data segments. Usually, the hash algorithm with low collision rate such as SHA1 [7] results in more complexity during the hash computation.

In the third step, the deduplication process should look up the table called key-value store [6]. During the lookup, disk accesses can occur due to the oversized table which is lying on both main memory space and disk space. Finally, the deduplication process decides whether to store the data segments to the underlying storage system or not depending on the result of the lookup in the key-value store.

2.2 GPGPU Characteristics

GPGPU is the abbreviation of general purpose graphic processing unit. The recently released GPU products support not only the primitives specialized for the graphic processing, but also the execution of general instructions used in the CPU. Heterogeneous computing model such as CUDA [20] makes existing programs run on GPGPU with maximized

Table 1: Specification of GTX480

Core Clock Frequency	1,401 MHz
Max.number of threads	23,040
Memory Size	1,535 MB
Memory Transfer Bandwidth (Host to Device)	2,756 MB/s
Memory Transfer Bandwidth (Device to Host)	3,198 MB/s
Memory Transfer Bandwidth (Device to Device)	121,000 MB/s
Cost (in dollars)	390$*

*Amazon.com, December, 2011

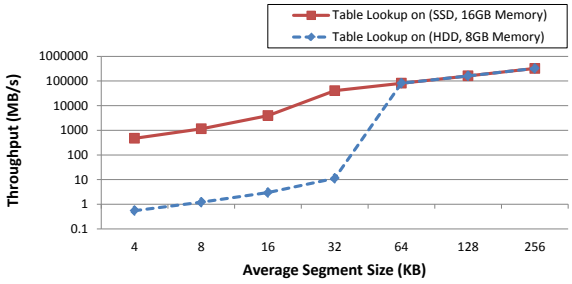

Figure 2: Comparison of Index Table Lookup in Different Specifications varying Average Segment Size

parallelism.

To use the computation power of GPGPU, the user should follow the specific sequence: (1) put data to be calculated from the host to GPGPU, (2) execute the GPGPU kernel and (3) return the result data from GPGPU to the host. Since GPGPU can only access the memory space of itself, memory transfer between the host main memory and the GPGPU memory is unavoidable. After transferring the kernel code and data region, GPGPU can execute its kernel code in parallel. In the end of the parallel computation, the data region with computation results should be transferred to be used in the host later.

Depending on the products, the specifications such as maximum number of threads GPGPU can execute are different. Table 1 shows the specifications for GTX480 [19] of our target system.

3. ANALYSIS

In this section, we analyze the performance overhead of each step in typical deduplication processing shown in Figure 2. Mainly, we compared the deduplication overhead appeared in the past machine specification to that in the primary storage system [2] expected to be relatively high-cost system. During the comparison, we assume the high-cost system can equip with Flash devices, enlarged memory space and GPGPU. As a result, we expect to see what was and what will be the crucial bottleneck in the past and the future, respectively.

3.1 Key-value Store Lookup

Key-value store table was the most time consuming part of typical deduplication processing. The result of the table lookup decides the uniqueness of a certain data segment. The size of this table varies depending on the average segment size and the total amount of data stored. For example, the table size is about 10GB if the average data segment size is 8KB and the amount of stored data is 4TB [29]. In the past, the major part of the table resides in the disk space since the size is over the small main memory size. A table lookup mostly resulted in one disk access in case of hash table data structure. This overhead was the outstanding problem which leads to low bandwidth of the deduplication system.

Figure 2 shows the calculated performance result of the table lookup in different specifications. Commonly, the storage size of both specifications is set to 20TB. We assume that one specification has HDD(10ms latency [5]) and 8GB Memory(1us latency [5]) as shown in the experiment of the previous work [29]. Another specification includes SSD(12.5us latency [24]) and 16GB Memory(1us latency) due to the assumption for the primary storage server. In addition, we applied 81% of disk access rate to the calculation while the number of disk accesses is equal to the number of table entries lying on the disk space. The previous work [29] insists that it improves 99% of disk accesses by its techniques. Among them, only one technique named *Summary Vector* is available for our target system since other techniques are based on the assumption that the target system is a backup system. According to the experiment in that work, the available technique reduces 19% of disk accesses in the large sized table. The metric for the throughput is the amount of data which fingerprints have looked up the table per second.

As a result, we can see that the performance saturation is about 1GB/s even though when the data segment size is 8KB which is relatively small. Moreover, we did not calculate any performance advantage from prefetching and spatial or temporal locality of table entries. From these factors, the overhead of key-value store lookup is mitigated a lot compared to that in the past though it is not eliminated.

Since we assumed that the primary storage server has GPGPU also, we can have a question how much performance improvement of the table lookup there is in GPGPU. To answer the question, we did the table lookup experiment on Intel Xeon 3.00GHz CPU and GTX480 GPGPU as illustrated in Table 2. The size of the table was set to 500MB for both. Also, we assumed that all the data segments already have been transferred to GPGPU memory, and each data segment already has its fingerprint. According to the graph, GPGPU also can improves the table lookup function compared to the capability of CPU. It is due to not only broaden memory bandwidth of GPGPU, also massive thread-level parallelism of GPGPU though it is not proportional to the number of threads which GPGPU has.

3.2 Segmentation

There are two ways of data segmentation, fixed and variable-length segmentation. Fixed-length segmentation segments a largely chunked data stream(usually about 1-64MB [9, 27]) into fixed-length blocks so that it does not incur any computation overhead. Variable-length segmentation divides the stream into variable sized segments [23]. To decide the segmentation boundary, the sliding window of certain bytes seeks the data stream. Sliding the window, the data within the window is hashed and the hash result is compared with the magic number of the variable-length segmentation algorithm. If it matches, the data stream is chopped at the

Table 2: Comparison of Index Table Lookup in CPU and GPGPU

# of Table Lookups	Time Consumed for Table Lookups	
	CPU (Intel Xeon 3.00GHz)	GPGPU (GTX480)
40,000	30ms	1ms
80,000	98ms	1ms
160,000	160ms	1ms
320,000	370ms	3ms
640,000	649ms	4ms
1,280,000	1,452ms	8ms

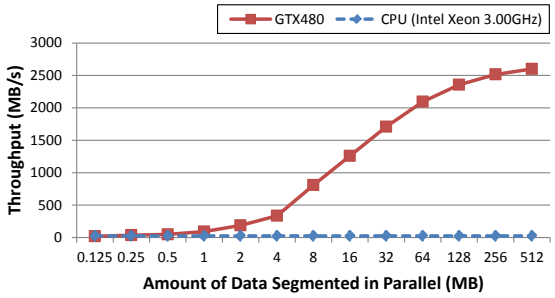

Figure 3: Comparison of Segmentation in CPU and GPGPU

current location of the sliding window's endpoint.

In previous works [18, 29], variable-length segmentation brought definitely better deduplication rate due to the capability detecting shifted data. However, in the most deduplication systems, it was not used due to the high computation overhead. The comparison result of segmentation performance in CPU and GPGPU is shown in Figure 3. The y-axis of the graph indicates the amount of segmented data per second. Each segmentation operation divides a large data chunk of 128KB into data segments with the average size of 4KB while the window size of the segmentation algorithm is set to 63 bytes. In case of the single threaded CPU result, it is limited to 25MB/s while GPGPU result approaches 2.6GB/s in maximum. Since it is able to parallelize, multiple number of cores about 100 also can increase the performance up to that of GPGPU. However, even though the performance is approached, GPGPU is still better for this functionality in the perspective of cost since the cost of GTX480 is almost equal to that of Intel Xeon E5620 CPU with only 4-cores and 2.4GHz clock frequency [19, 13].

3.3 Fingerprinting

Fingerprinting is equal to hashing since the fingerprint is obtained by hashing certain data segment. Figure 4 shows the comparison of hashing capability in CPU(Intel Xeon 3.0Ghz) and GPGPU(GTX480) which generate fingerprints for 4KB data segments. Of course, massively parallelized GPGPU earned much higher throughput than CPU. The saturation shown in case of GPGPU is also the memory transfer bottleneck between GPGPU memory and host main memory in Table 1. According to the graph, the maximum

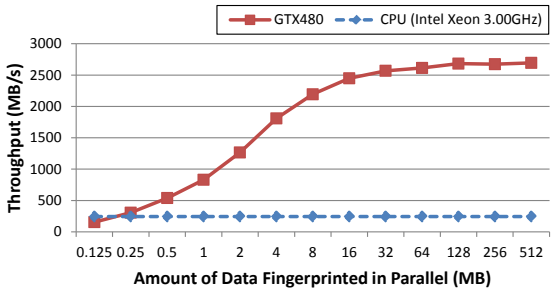

Figure 4: Comparison of Fingerprinting in CPU and GPGPU

throughput can be achieved by parallel hashing for 128MB of data which is similar with the amount of data needed for the maximum segmentation throughput in Figure 3.

Comparison of 1 CPU and 1 GPGPU might be not fair enough. In the perspective of cost, CPU chip with 4 cores is equal to 1 GPGPU as mentioned in [19, 13]. According to the comparison result, 7 times of CPU result would approach to the GPGPU result. GPGPU has better cost efficiency even only for fingerprinting.

Moreover, GPGPU will be used also for the functionality of segmentation which requires same data set with fingerprinting as mentioned previously. It means only one memory transfer to GPGPU will be required for 2 kinds of functionality. The graphs shown in Figure 3, 4 for segmentation and fingerprinting include the overhead of GPGPU memory transfer in each throughput result. Thus, efficiency of GPGPU will be much better than what the graphs indicate, so that the use of GPGPU for the segmentation and fingerprinting is not comparable with the use of multiple CPUs.

3.4 Summary

Through the analysis so far, we reviewed the bottleneck of typical deduplication processing in the past, and forecasted how each bottleneck is going to be mitigated or unveiled in the primary storage server system employed GPGPU, enlarged main memory space and Flash based device.

In summary, we can achieve the throughput of several GB per second by offloading the entire steps of the deduplication processing to GPGPU. We have verified that each step of the deduplication processing shows better performance in GPGPU. Moreover, not only achieving the high throughput, the deduplication processing with GPGPU can bring more deduplication rate and low collision rate by dedicating GPGPU's remaining computation power for better segmentation and hashing algorithms with more complexity in computation.

4. DESIGN OF GHOST

Based on the results shown in the analysis section, we propose a complete deduplication system which offloads most of functionalities to GPGPU, called GHOST. Our design is expected to fully exploit the computational power and the wide memory access bandwidth of GPGPU. For the purpose, we propose the system architecture with In-Host Data Cache (IHD-Cache) which interacts with GPGPU. The deduplication processing in the system follows the sequence of data

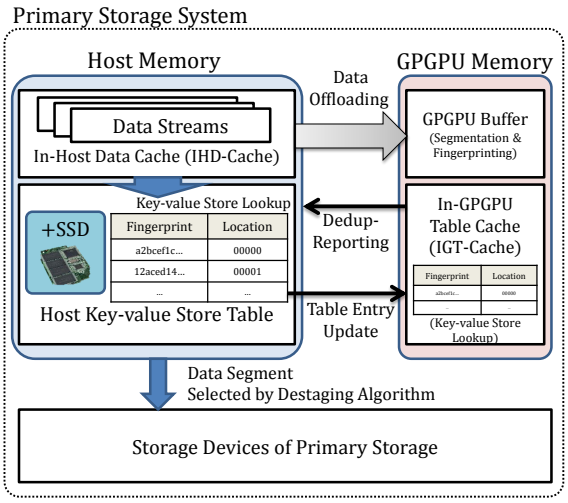

Figure 5: Proposed System Architecture

offloading, deduplication, and result reporting. We explains each part of the system and its related operations. In the last part of this section, we describes how the whole deduplication processing works for certain offloaded data depending on the decisions made during the deduplication processing.

4.1 IHD-Cache

In our proposed system, the cache structure inside the system is required due to multiple reasons. First of all, without the cache, the data segment inserted into GPGPU should be popped out again if the negative deduplication decision is made. This will almost double the memory transfer overhead between host and GPGPU. Second, since we assume the system to be a primary storage, the cache structure can obtain the traditional benefit such as better destaging throughput which is crucial in the high performance storage system.

Figure 5 describes our proposed deduplication system architecture with GPGPU. For the listed reasons, we inserted the cache structure named IHD-Cache into the system. In relation with IHD-Cache, the following operations will simultaneously runs on the system: (1) the cache receives requests from the clients of the storage, (2) the destaging algorithm sends the data segments in IHD-Cache which is not able to deduplicate to the underlying storage, and (3) GPGPU proceeds the deduplication for large amount of data transferred from IHD-Cache. The unique part of our system is the part that offloads data to GPGPU, executes the deduplication processing and returns the result back to host. In the below, we describe this part in detail.

4.2 Destage-aware Data Offloading to GPGPU

4.2.1 *Amount of Data Offloaded at Once*

The performance benefit of GPGPU is obtained when the data is processed in parallel. The execution time of the parallel jobs is usually same regardless of the number of threads if the number is below the maximum. Figure 4 of the analysis section showed the hash throughput varying the amount of data. Since the data segments to be hashed is fixed to 4KB in this case, larger data size indicates there are more thread-level parallelism for data hashing. The throughput of the hashing in GPGPU is saturated to 2.6 GB/s which is the bottleneck of the memory transfer to GPGPU when the amount of data offloaded is above 128MBs (similar result obtained for the segmentation in Figure 3). From the results, we can set the amount of data to be offloaded at once to 128MBs.

Also, we can set the appropriate size of IHD-Cache from the offloading data size. To do the role of buffer for the offloading, the cache size should exceed the offloading size. However, too large IHD-Cache size only uses up the main memory space without precious benefit which should be reserved for key-value store. In our prototype system, we set the size of IHD-Cache to be two times of the data offloading size.

4.2.2 *Selection of Data Offloaded*

What data should be offloaded is closely related with the efficiency of deduplication in GPGPU. Since we are targeting the primary storage, there are expected user operations such as modification. Traditionally, destaging algorithms [11, 10] in the cache structure were aware of them, and took advantage from them by relying on the spatial and temporal locality. By not destaging the data expected to have modification, it reduced the number of write operations sent to the underlying storage. It is called *write cache hit* [11]. During the deduplication processing, the same phenomenon can occur. When a data segment is modified, the previous deduplication result or the result currently being generated of the data segment becomes useless.

When such case appears many times, the efficiency of deduplication processing gets lower. In other word, the data expected not to have modification soon should be offloaded to GPGPU in our system. Fortunately, the algorithm for destaging can be applied to the selection of data to be offloaded in the same manner. The destaging algorithm maintains its own data structure such as doubly linked list sorted by the order of destaging [11, 10]. The early entries of the list are expected not to have modification soon by the algorithm. During the data offloading phase, the offloading part is permitted to look up and pick the data from the early entries of the list.

We call this collaboration scheme with the destaging algorithm as Destage-aware Data Offloading (Data Offloading in Figure 5). The system might have the different result depending on the capability of destaging algorithms, but it is clear that it can take advantage from the destaging algorithms to reduce the problem cases.

4.3 Key-value Store Management

4.3.1 *IGT-Cache by Hot/Cold Separated Table Entries*

In the comparison shown in Table 2, key-value store lookup in GPGPU is also better than that in CPU due to the broad memory access bandwidth and parallel memory access of GPGPU. Our system maintains the part of the key-value store in GPGPU named In-GPGPU Table Cache (IGT-Cache) to utilize this aspect of GPGPU. However, GPGPU memory has limited size about 1.5GB in case of GTX480 which is not available to contain all of the table. 128MB among the limited GPGPU memory space is already used for the buffer for the segmentation and fingerprinting. When the system let

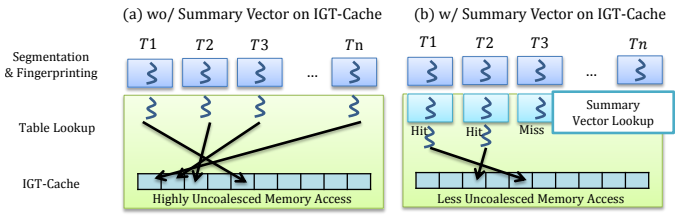

Figure 6: Better Key-value Store Lookup in GPGPU using Summary Vector Concept

all the data segments look up IGT-Cache, only part of the table can be looked up so that the missed lookup will result additional overhead for a certain data segment(two times lookup, once in GPGPU and another in the host table).

To avoid such duplicated lookup cases, first of all, the system should increase the hit ratio in GPGPU table lookup. For the purpose, we maintain the table cache of the whole table which is constructed by hot/cold separation in GPGPU. From the history, the table entries are sorted by the deduplication count per each entry. Upper entries among the entire table filling approximately 1.2GB of memory are selected for IGT-Cache entries. They are periodically transferred into or evicted out from IGT-Cache during the system idle time to avoid the interrupt on the deduplication processing(Table Entry Update in Figure 5).

Once an entry is placed in IGT-Cache, the data segment with the same fingerprint of the entry should hit in the table cache since all of data segments are given the chance to look it up. Therefore, the entries in IGT-Cache and the host table are explicit in our prototype system to utilize the main memory space efficiently.

4.3.2 *Summary Vector for IGT-Cache*

Unfortunately, IGT-Cache can not avoid major part of host table lookup because of the deduplication rate which is usually below 50%. For those who will miss the table cache, it is better not to look up the table cache at all since the unnecessary lookups only make the memory access pattern more uncoalesced as shown in Figure 6.

In our proposed system, *Summary Vector* concept [29] is applied to IGT-Cache to resolve the uncoalesced GPGPU memory access problem in Figure 6. Originally, this concept is adopted to reduce the unnecessary disk access during key-value store lookup. When an entry is inserted into the table, a key is hashed by several simple hash functions. The results of those hashing decides the offset in a vector of certain bytes, and the bits on the offsets of the vector are set to 1. During the lookup, the offsets are generated in the same manner, and the bits on the offsets are read. If any of the bits have the value of 0, the key definitely does not reside in the table since it guarantees that the key was never been inserted into the table.

Applying the concept of the summary vector to IGT-Cache, the deduplication system can avoid most of IGT-Cache lookups which will miss the cache. One might doubts why the summary vector in GPGPU is only for the cache of the table, not the whole table. For the proper operation of the summary vector concept, only one summary vector should exist for a table. As mentioned earlier, we cannot contain whole table in GPGPU which means that a consistent set of the table should reside in the host space. If the

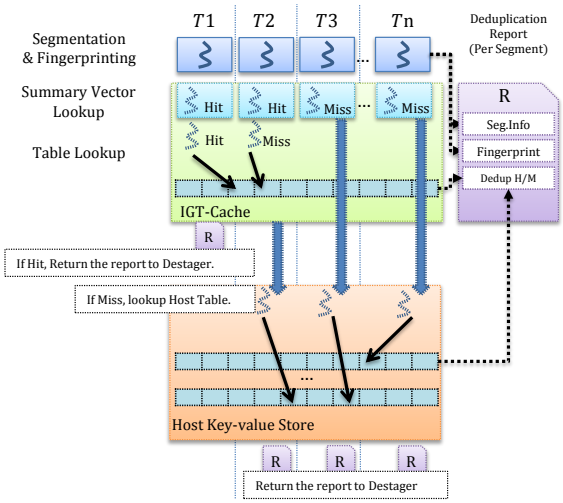

Figure 7: Deduplication Flows of Proposed System

location of the summary vector for whole table is GPGPU, every table lookup should refer GPGPU memory which is not acceptable due to the overhead.

The summary vector of IGT-Cache only can be looked up by the deduplication thread of GPGPU. Updates of the summary vector should be done at certain time since the hot entries according to the hot/cold separation policy may changes after certain time. Our system updates IGT-Cache entries periodically while the system is idle. The update of the summary vector is also done at this moment. For the accuracy of the summary vector, it should be built from the bottom by scanning the all entries residing in IGT-Cache.

4.4 Parallel Execution of GPGPU

After a number of data streams are offloaded to GPGPU, the deduplication processing will be done for those data. As explained previously, our system offloads whole steps of deduplication to GPGPU: segmentation, fingerprinting and key-value store lookup.

The discussion on segmentation and fingerprinting in GPGPU is trivial. They are almost unlimitedly executed in parallel. Any combination of previous works of hashing algorithms and variable-length segmentation algorithms such as [7, 17] and [23] can be adopted in this system. The algorithm with better functionality is recommended since the computation overhead will be hidden by the thread-level parallelism. In our prototype system, we used Rabin's Chunking [23] and SHA1 [7] for segmentation and fingerprinting, respectively. During these steps, the deduplication report per data segment shown in Figure 7 is updated. The segmentation updates the segmentation information such as the endpoint of certain segment, and the fingerprinting records the fingerprint of the data segment in the report.

After the segmentation and fingerprinting, GPGPU deduplication processing tries to look up IGT-Cache first before the complete table image in the host. Figure 7 explains the key-value store lookup procedure in detail. Assume that there are four threads ($T1, T2, T3, Tn$) as shown in the figure. All of them firstly check the summary vector of IGT-Cache. Among them, *T1* and *T2* hits the summary vector. They can look up IGT-Cache at this point. According to the de-

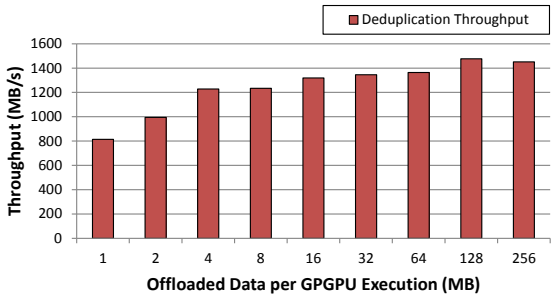

Figure 8: Deduplication Throughput of Proposed System varying Size of Offloaded Data

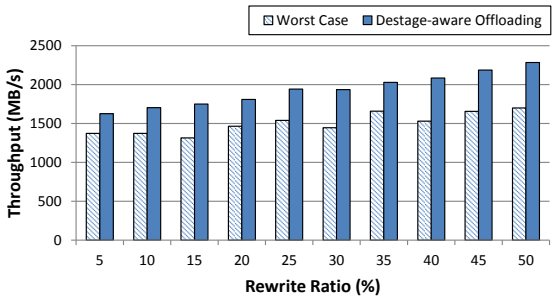

Figure 9: Deduplication Throughput of Proposed System varying Data Offloading Policy

cision of the table cache, *T1* which hits the table cache can be deduplicated while *T2* should look up the table in the host again. *T3* and *Tn* which missed the summary vector of the table cache directly look up the table in the host. According to the decision of the summary vector, IGT-Cache and the host table, each segment is branched to the one of the three flows as explained. When each flow ends up with GPGPU, the result of the deduplication processing in GPGPU is reported to IHD-Cache so that the data segment can be destaged or deduplicated. (Dedup-Reporting in Figure 5)

5. EXPERIMENTS

In this section, we examine how much performance impact each part of our GPGPU offloaded deduplication system has. For the purpose, we assumed that the underlying storage provides enough throughput by doing memory copy instead of disk write. The prototype of our GPGPU-offloaded deduplication system is implemented in the user level. The part of inside architecture implementation is borrowed from LORE [3]. Our test machine equipped with GTX480, Intel Dual-core Xeon 5160 CPU and 4GB of main memory. We assume the table resides on only main memory space, not in the disk space. Also, the segmentation in the system chops the stream into fixed-length segments. These features will be included in our further works.

5.1 Tuning Data Offloading Parameters

5.1.1 Varying Amount of Offloaded Data at once

In the experiment shown in Figure 8, we measure the system throughput of our prototype system varying the amount of data per GPGPU dedup-operation. We generate all the segments of the workload to be unique. Segment size, and cache size are set to 32KBs and 256MB. The elapsed time for the throughput measurement is stopped when there is no dirty data left in the cache. In similar with the results in Figure 3, 4 of the analysis section, this result also gives us the lesson that 128MBs is the appropriate size for the data offloading. Exceeding the best point results in the performance degradation since GPGPU deduplication processing waits for a while till the timeout when the offloading size is not fully filled.

5.1.2 Varying Selection Policy of Offloaded Data

Our data selection policy for offloading to GPGPU is collaboration with the employed destaging algorithm. To verify how much it is effective, we measured the system throughput varying the rewrite ratio. The rewrite ratio means the possibility that a certain request with unique ID is written again into the system in the specific time period (1 second for the prototyped system).

In Figure 9, we compared Destage-aware Offloading with the worst case while our prototype implementation used Least-Recently-Written(LRW) algorithm for the destaging policy. The worst case indicates the offloading which follows the reverse order of the destaging algorithm's order. While the worst case remains at the similar throughput level regardless of the rewrite ratio, Destage-aware Offloading shows the increasing throughput result. The main difference is how much of deduplication processing have been invalidated. This difference tells how much the offloaded deduplication works efficiently depending on the data selection policy.

In the real system, the rewrite ratio might not be always high over 50%. However, our deduplication system is affected by relatively low rewrite ratio since GPGPU offloading brings long latency per each data segment. Thus, the data selection policy is treated as one of the essential part of our system.

5.2 Impact of IGT-Cache

5.2.1 Existence of IGT-Cache

Though key-value store lookup overhead is mitigated by SSD and enlarged memory space as mentioned in the analysis section, its portion among the total time consumed for the deduplication processing is still not negligible. To enhance this overhead, we suggested IGT-Cache. According to the explanation in the design section, the system can take advantage from it when the deduplication rate gets higher.

To verify the effect, we measured the throughput varying the deduplication rate of the workload, and compared the results of the systems with and without IGT-Cache. Figure 10 illustrated the results. In this results, the summary vector on IGT-Cache is disabled to examine the effect of only IGT-Cache existence. The x-axis of the bar graph indicates the deduplication rate on IGT-Cache. To compare with the system without IGT-Cache, we assumed that the deduplication rate on IGT-Cache is equal to the deduplication rate of the entire system. As expected, the time consumed for key-value store lookup is reduced in proportional to the deduplication

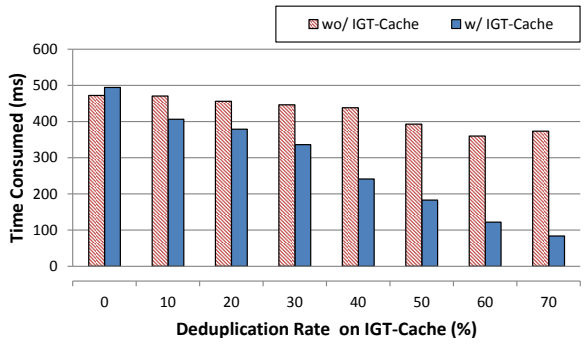

Figure 10: Improved Key-value Store Lookup by IGT-Cache

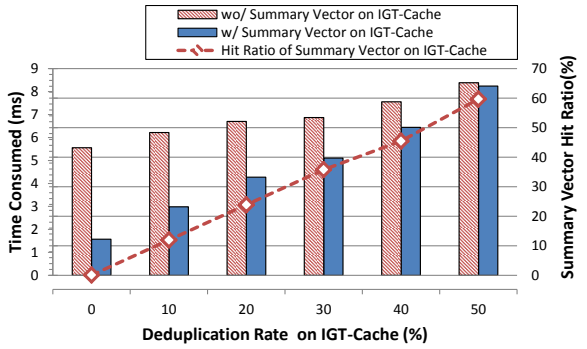

Figure 11: Effect of Summary Vector on IGT-Cache

rate. When a data segment finds its identical entry in IGT-Cache, it can avoid to look up the host table which takes more time than that in IGT-Cache lookup.

This result is a bit optimistic since we assumed the deduplication rate of the system is equal to the hit ratio in IGT-Cache. However, according to the result, it definitely brings the improvement of key-value store lookup even for the workload with low deduplication rate.

5.2.2 *Existence of Summary Vector on IGT-Cache*

Though we verified the advantage of IGT-Cache, it also has the performance overhead in the worst case such as the 0% deduplication rate shown in Figure 10. To mitigate the overhead when there is no chance for the deduplication, we adopted *Summary Vector* on IGT-Cache.

Figure 11 illustrates the effect of the summary vector. The y-axis of this graph indicates the time consumed for IGT-Cache lookup while the y-axis of Figure 10 was the time consumed for overall key-value store lookup. With the summary vector, the time taken for the lookup on IGT-Cache is reduced from 5.6ms to 1.6ms when the deduplication rate is 0%. Since the summary vector prohibits the data segments which will miss the table cache to look up the table cache, the rest of data segments can access the table cache with less uncoalesced memory access pattern in GPGPU. It reduces the time for the table cache lookup.

Though it cannot obtain the same performance advantage when the deduplication rate is high, it is acceptable since the existence of IGT-Cache is already obtaining the huge amount of performance benefit as shown in Figure 10.

Figure 11 additionally shows the line graph indicating the

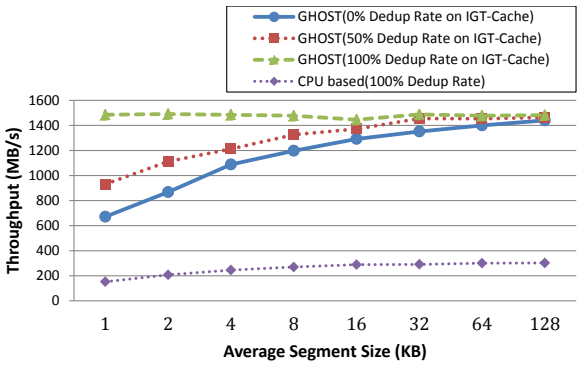

Figure 12: Comparison of CPU-Based Deduplication and GHOST varying Deduplication Rate

hit ratio of the summary vector on IGT-Cache. The hit ratio means the possibility to allow the table cache lookup. All the value of the line graph were achieved in the following environment: vector size of 22.5MB, Bloom Filter with four hash functions and 500MB of IGT-Cache. The values have the difference from the deduplication rate less than 10%. It is the pretty accurate rate, thus, we can obtain the performance advantage shown in the bar graph of Figure 11.

5.3 Macro Throughput

5.3.1 *Varying Deduplication Parameters*

So far, we have measured the performance of each part of the design we proposed. In Figure 12, we compare the overall throughput result with CPU-based deduplication system. For the fair comparison, we replaced the GPGPU offloading part to CPU-based fingerprinting, and disabled IGT-Cache.

To see the effect of the deduplication rate, we includes the throughput result of GHOST with the various deduplication rate on IGT-Cache. CPU-based deduplication system in the graph has the deduplication rate of 100%. We exclude the result of CPU-based deduplication system with other deduplication rates since they are almost same with that of 100% shown in the graph.

As shown in the figure, our GHOST system achieved about 1.5GB/s in maximum. Due to IGT-Cache, the graph of GHOST with 100% IGT-Cache Hit shows the regular performance with the maximum value while others show the degradation when the segment size gets smaller. In comparison with CPU-based deduplication system, the throughput of GHOST is six-times better that that of CPU-based deduplication system which is much above the border line of GPGPU cost effectiveness.

In addition, our prototype does not contain yet the variable-length segmentation which can earn much benefit from GPGPU. After including the feature, the difference between our GHOST system and CPU-based system will get larger.

6. RELATED WORKS

Dutch et al [18] investigated 857 desktop computers to study the practical deduplication. They showed the deduplication rate of the individual desktop users' filesystems approaching below 50% in the set of 857 filesystems regardless of the average data segment size or segmentation algorithm. The result enlightened the future of deduplication in the

primary storage of cloud environment.

There was a trial to utilize multicore CPU to accelerate the deduplication by Guilherme et al [4]. They suggested how the deduplication process can be executed in parallel using parallel programming model, however, the bottleneck of the deduplication such as table lookup or computational overhead was out of the scope of their work.

As the capability of GPU is extended to the area of CPU, lots of works to utilize the strength of GPGPU have researched recently. Packet Shader [12] is the representative of them, and it uses GPGPU to the packet forwarding of software router. They improved the software router performance more than four times by massive thread-level parallelism of GPGPU and several optimization schemes.

There were two works [8, 28] to utilize GPGPU in the deduplication category. The first one [8] suggested GPU accelerated storage system. Within the suggested system, it includes the deduplication feature and GPGPU is used for the feature to calculate fingerprint of data segment. Although the purpose of GPGPU usage is simliar with ours partially, their work assumed that GPGPU resides in client side and clients have responsibility to calculate the fingerprint while we let clients do nothing. Another work in [28] concentrates on how efficiently segmentation and fingerprinting can be done in CPU and GPGPU heterogeneous architecture. Though they suggested the rule for the offloading in the heterogeneous architecture in the perspective of segmentation and fingerprinting, their work did not consider the whole deduplication processing while our proposed system deals with the whole steps.

7. DISCUSSION

7.1 SandyBridge Architecture vs. GPGPU Architecture

Currently, Intel shipped the new type of CPU which codename is SandyBridge [14] to face GPGPU architecture of the companies such as NVIDIA. SandyBridge has heterogeneous architecture of multicore CPU and GPU. Though the GPU inside the chip is much stronger than previous inner GPU of motherboard chipset, it has less thread-level parallelism than GPGPU. However, its strength is not on the computation power. SandyBridge can share the memory and even L3 cache with the host system while GPGPU cannot.

The degree of thread-level parallelism which our system needs does not reach to the maximum capability of GPGPU. More importantly, the performance of the proposed system is limited by the bandwidth of memory transfer between host and GPGPU. These two factors forecast SandyBridge can be a replacement of GPGPU in our proposed deduplication system. We are currently working on the development of the deduplication system on SandyBridge architecture, and waiting for the relase of SDK from Intel.inc.

7.2 Fault Tolerance

Since the proposed system is using IHD-Cache which borrows the memory space from the volatile DRAM, the fault tolerance can be an essential issue. Moreover, because we maintain IGT-Cache and a part of the host table in volatile memory of GPGPU, the table management is also related with the same issue. So far, our prototype system is not tolerable for the sudden power failure. However, there are several plans to be tolerable enough.

For IHD-Cache, battery-backed DRAM or non-volatile RAM can be used since the cache space is relatively small(about 256MB). Asynchronous write due to the cache structure makes a user believe his data is stored well. Losing data due to the power failure will destroy the trust of the system. Unfortunately, without them, our system can face the problem of losing data.

As mentioned in the context, IGT-Cache is generated using the system idle time. It means that the system can have enough time to back up the generated IGT-Cache into the somewhere of the permanent disks.

Most of the host table in the memory space also can be backed up periodically. Small amount of table entry updates within the backup period can be managed in the small scratch pad with battery-backed DRAM or non-volatile RAM.

8. CONCLUSION

To provide high performance storage deduplication for primary storage system, we thoroughly reviewed conventional deduplication systems. We consequently showed the new bottleneck of deduplication processing in the primary storage system with Flash-based devices and enlarged memory space. Based on the result, the deduplication processing offloaded to GPGPU, called GHOST, is proposed to perform the high throughput to be used in the primary storage system.

Our deduplication system features three remarkable achievements: First, we developed the full deduplication processing system which is optimized for offloading the deduplication functionality to GPGPU. Second, destage-aware Data Offloading enhanced the efficiency of deduplication processing in GPGPU especially for the primary storage. Third, IGT-Cache mitigated the remaining overhead of key-value store lookup by placing the hot table entries in GPGPU. To optimize the table lookup in GPGPU, we also adopted *Summary Vector* for IGT-Cache.

Our experiments prove that the proposed deduplication can perform about 1.5GB/s while the deduplication without GPGPU offloading can have 300MB/s in maximum. Though spending large cost on multicore CPUs can bring the close throughput, the adoption of GPGPU is better in the perspective of cost effectiveness. Moreover, it can be adopted the existing system the system provider has without massive system upgrade.

Acknowledgement

The authors wish to thank their anonymous referees for all of their invaluable comments and suggestions in advance. The work presented in this paper was supported by MKE (Ministry of Knowledge Economy, Republic of Korea), Project No. 10035231-2010-01.

9. REFERENCES

[1] Amazon.com. Amazon Elastic Cloud Service. `http://aws.amazon.com/ec2/`, 2007.

[2] Amazon.com. Amazon Simple Storage Service. `http://aws.amazon.com/S3/`, 2007.

[3] Sung Hoon Baek and Kyu Ho Park. Prefetching with adaptive cache culling for striped disk arrays. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 363–376, Berkeley, CA, USA, 2008. USENIX Association.

[4] Guilherme Dal Bianco, Renata Galante, and Carlos A. Heuser. A fast approach for parallel deduplication on multicore processors. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1027–1032, New York, NY, USA, 2011. ACM.

[5] Biplob Debnath, Sudipta Sengupta, and Jin Li. ChunkStash: speeding up inline storage deduplication using flash memory. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 16–16, Berkeley, CA, USA, 2010. USENIX Association.

[6] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[7] D. Eastlake, 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). 2001.

[8] Abdullah Gharaibeh, Samer Al-Kiswany, Sathish Gopalakrishnan, and Matei Ripeanu. A GPU accelerated storage system. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 167–178, New York, NY, USA, 2010. ACM.

[9] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, New York, NY, USA, 2003. ACM.

[10] Binny S. Gill, Michael Ko, Biplob Debnath, and Wendy Belluomini. STOW: a spatially and temporally optimized write caching algorithm. In *Proceedings of the 2009 conference on USENIX Annual technical conference*, USENIX'09, pages 26–26, Berkeley, CA, USA, 2009. USENIX Association.

[11] Binny S. Gill and Dharmendra S. Modha. WOW: wise ordering for writes - combining spatial and temporal locality in non-volatile caches. In *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies - Volume 4*, pages 10–10, Berkeley, CA, USA, 2005. USENIX Association.

[12] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: a GPU-accelerated software router. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 195–206, New York, NY, USA, 2010. ACM.

[13] Intel. Intel Processor Pricing. `http://www.intc.com/pricelist.cfm`, 2011.

[14] Intel. Intel Xeon Processor E3-1200 Family Datasheet. `http://www.intel.com/content/www/us/en/processors/xeon/xeon-e3-1200-family-vol-1-datasheet.html`, 2011.

[15] Mark Lillibridge, Kave Eshghi, Deepavali Bhagwat, Vinay Deolalikar, Greg Trezise, and Peter Camble. Sparse indexing: large scale, inline deduplication using sampling and locality. In *Proccedings of the 7th conference on File and storage technologies*, pages 111–123, Berkeley, CA, USA, 2009. USENIX Association.

[16] NexR. Co. Ltd. iCube Cloud Computing and Elastic-Storage Service. `http://www.icubecloud.com`, 2010.

[17] Florian Mendel and Vincent Rijmen. Cryptanalysis of the tiger hash function. In *Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security*, ASIACRYPT'07, pages 536–550, Berlin, Heidelberg, 2007. Springer-Verlag.

[18] Dutch T. Meyer and William J. Bolosky. A study of practical deduplication. In *Proceedings of the 9th USENIX conference on File and stroage technologies*, FAST'11, pages 1–1, Berkeley, CA, USA, 2011. USENIX Association.

[19] NVIDIA. NVIDIA GeForce GTX 480/470/465 GPU Datasheed. 2010.

[20] NVIDIA. CUDA:Parallel Programming Computing Platform. `http://www.nvidia.com/object/cuda_home_new.html`, 2011.

[21] Kyu Ho Park, Youngwoo Park, Woomin Hwang, and Ki-Woong Park. MN-Mate: Resource Management of Manycores with DRAM and Nonvolatile Memories. *High Performance Computing and Communications, 10th IEEE International Conference on*, 0:24–34, 2010.

[22] Sean Quinlan and Sean Dorward. Venti: A New Approach to Archival Storage. In *Proceedings of the Conference on File and Storage Technologies*, FAST '02, pages 89–101, Berkeley, CA, USA, 2002. USENIX Association.

[23] M. O. RABIN. Fingerprinting by Random Polynomials. In *Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University*, 1981.

[24] Samsung. 128GB 2.5-inch SSD 830 Series. `http://www.samsung.com/us/computer/memory-storage/MZ-7PC128N/AM-specs`, 2011.

[25] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX conference on File and Storage Technologies*, Berkeley, CA, USA, 2007. USENIX Association.

[26] Azure Service. Microsoft, Windows Azure Platform. `http://www.microsoft.com/windowsazure/`, 2010.

[27] K. Shvachko, H. Huang, S. Radia, and R. Chansler. The hadoop distributed file system. In *26th IEEE (MSST2010) Symposium on Massive Storage Systems and Technologies*, MSST '10, 2010.

[28] Zhi Tang and Youjip Won. Multithread Content Based File Chunking System in CPU-GPGPU Heterogeneous Architecture. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on*, pages 58 –64, june 2011.

[29] Benjamin Zhu, Kai Li, and Hugo 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'08, pages 18:1–18:14, Berkeley, CA, USA, 2008. USENIX Association.