University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers | Computer Science and Engineering, Department of

2011

SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services in Data Centers

Lei Xu
University of Nebraska - Lincoln, lxu@cse.unl.edu

Jian Hu
University of Nebraska-Lincoln, jhu@cse.unl.edu

Stephen Mkandawire
University of Nebraska – Lincoln, smkandawire@gmail.com

Hong Jiang
University of Nebraska - Lincoln, jiang@cse.unl.edu

Follow this and additional works at: http://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Xu, Lei; Hu, Jian; Mkandawire, Stephen; and Jiang, Hong, "SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services in Data Centers" (2011). *CSE Conference and Workshop Papers*. 206.
http://digitalcommons.unl.edu/cseconfwork/206

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

2011 31st International Conference on Distributed Computing Systems Workshops

SHHC: A Scalable Hybrid Hash Cluster for Cloud Backup Services in Data Centers

Lei Xu, Jian Hu, Stephen Mkandawire and Hong Jiang
University of Nebraska-Lincoln
{*lxu, jhu, smkandaw, jiang@cse.unl.edu*}

***Abstract*—Data deduplication techniques are ideal solutions for reducing both bandwidth and storage space requirements for cloud backup services in data centers. Current data deduplication solutions rely on comparing fingerprints (hash values) of data chunks to identify redundant data and store the fingerprints on a centralized server. This approach limits the overall throughput and concurrency performance in large scale systems. Furthermore, the slow seek time associated with hard disks degrades the performance of hash lookup operations which are mainly random I/Os.**

In this paper we present a scalable hybrid hash cluster (SHHC) to maintain a low-latency distributed hash table for storing data fingerprints. Each hybrid node in the cluster is composed of RAM and Solid State Drives (SSD) to take advantage of the fast random access inherent in SSDs. This distributed approach makes the system scalable, balances the load on the hash store and significantly reduces the latency of the hash lookup process.

I. Introduction

In a recent study [4], IDC predicted that by the year 2020, the amount of digital information created and replicated in the world will grow to almost 35 billion terabytes. The report further indicates that nearly 75% of digital information is a copy, i.e. only 25% is unique. With emerging trends of cloud computing, it is therefore important to employ strategies such as data deduplication to improve both storage capacity and network bandwidth utilization.

In the deduplication process, duplicate data is detected and only one copy of the data is stored, along with references to the unique copy of data, thus removing redundant data. The most common deduplication technique splits data into chunks of non-overlapping data blocks [12], calculates a fingerprint for each chunk using a cryptographic hash function (e.g. SHA-1) and stores the fingerprint of each chunk in a hash table (chunk index). Each chunk stored on the storage system has a unique fingerprint in the chunk index. To determine whether a chunk is already stored on the system or not, the fingerprint of the incoming data item is first looked up in the chunk index and if there is a match, the system only stores a reference to the existing data. Otherwise the incoming chunk is considered unique and is stored on the system and its fingerprint inserted in the chunk index.

Cloud based backup applications generally benefit the most from deduplication due to the nature of repeated full backups of existing data. Removing redundant or duplicate data makes it possible to reduce storage capacity requirements and data traffic over the network. For hash based in-line deduplication systems, the chunk index and the associated fingerprint lookup operations can present a throughput bottleneck due to limited system memory and disk I/O performance.

Various techniques [3], [5], [6], [12] have been proposed to address the disk I/O problem and improve the throughput of fingerprint lookups. ChunkStash [3] stores the chunk index on SSD and employs flash aware data structures and algorithms to get the maximum SSD performance benefit and outperforms a hard disk index based in-line deduplication system by 7x-60x. However, even at such orders of magnitude of lookup performance, maintaining the chunk index on a centralized server puts the server itself in the critical path and presents a performance bottleneck when scaling at rates of hundreds of thousands of concurrent backup client requests.

Given the current trends in cloud computing, a typical datacenter has to handle numerous numbers of concurrent client backup requests for both small and large datasets. To this end we propose a highly scalable and low latency hybrid hash cluster (SHHC) for cloud storage. SHHC is a distributed fingerprint store and lookup service which can scale to handle a very large number of concurrent backup clients while maintaining high backup throughput for hash based in-line deduplication storage systems.

II. Background and Motivation

A. Motivation

In-line data duplication lends itself as a natural technique for use in data centers in the cloud. Removing redundant data in real time makes it possible to reduced data traffic over the network and improves storage utilization of storage systems. The core of in-line deduplication is the fingerprint store and fingerprint lookup mechanism and therefore, optimizing throughput at this critical path is vital for the performance of the whole backup service.

Solutions to improve in-line deduplication throughput have been proposed [3], [5], [6], [12]. However, in the current approaches the chunk index is still maintained as a centralized resource and therefore presents a performance bottleneck when scaling to serve hundreds of thousands of concurrent backup clients – a typical scenario in cloud data centers.

1545-0678/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDCSW.2011.31

To establish this point, we developed a simulator and used it to compare the throughput of a single hash server to that of a clustered approach. In this simulation we issued hash value queries to the distributed hash cluster for different numbers of cluster nodes, where one node represents a single server centralized approach. For each given configuration of the hash cluster, we injected a work set of SHA-1 fingerprints of 8KB chunks at different rates. As illustrated in Figure 1, the execution time for a given number of fingerprints is a decreasing function of the number of nodes in the cluster, which represents higher throughput.

Further, serving hundreds of thousands of users introduces an enormous requirement for storage and studies [4] indicate that the amount of digital information will continue to grow. However, the size and structure of the hash store can place a limit on the maximum amount of data that can be stored on a given storage system. Therefore, there's need for system designs to allow hash stores that support Exabyte scale data stores.

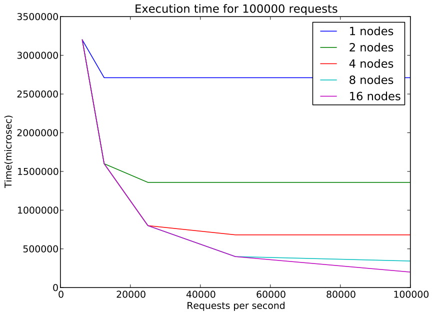

Figure 1. Throughput of fingerprint lookup operations

In-memory distributed cache systems have been widely used [7], [10], which is a testimony that the performance and cost effectiveness of systems such as our proposed distributed cache system for large scale computing infrastructures have proven.

B. *Related Work*

A typical backup data center has to scale to handle numerous client backup requests concurrently in order to meet the ever shrinking backup windows. Our work has been motivated by the need for high throughput in hash based deduplication storage systems in the cloud. It works in concert with other flash based index lookup techniques that reduce the latency of index lookups.

ChunkStash [3] takes advantage of SSD properties to improve in-line deduplication. It maintains an in-memory hash table to index the signatures on SSD by using cuckoo hashing to providing one flash read per signature lookup. By keeping the chunk index on SSD, [3] avoids the disk I/O bottleneck and reduces the latency of index lookups by serving such lookups from a flash-based index. However, the compact index for fingerprint in RAM limits the total size of the hash index, and the approach does not deal with the problem of scaling to serve millions of concurrent backup client requests.

DDFS [12] employs a compact in-memory data structure to identify new segments and exploits temporal locality of fingerprints of duplicate segments to achieve high cache hit ratios. However, the centralized chunk index in DDFS is still a bottleneck compared to a distributed architecture.

Sparse Indexing [5] uses sampling and exploits the inherent locality within backup streams to solve the chunk-lookup disk bottleneck problem that in-line, chunk-based deduplication schemes face. However, the approach is not for a cloud backup environment and therefore, does not consider a distributed hash store.

Extreme Binning [2] proposed a scalable deduplication technique for non-traditional backup workloads that are made up of individual files with no locality among consecutive files in a given window of time. It selects a representative chunk ID of a file's IDs to be stored in RAM to index IDs of this file left on the disk. However, because most IDs are stored together with the real data chunks on the disk, a miss in RAM leads to a seek on the disk, suffering the disk I/O penalty.

dedupv1 [6] optimizes throughput by using an SSD based index system. While taking advantage of SSD's good random reads and sequential operations, [6] alleviates the SSD random writes problem by delaying I/O operations.

While the above approaches reduce the latency of index lookups, none of them considers a distributed architecture of the chunk index, thus leading to a bottleneck for the deduplication process.

III. Design and Implementation

A. *Overall Architecture*

In addition to the growing volumes of data from enterprises, datacenters now have to handle even more growth in data traffic due to the proliferation of mobile user computing. Driven by this understanding our design takes into account that cloud storage clients range from enterprise users to millions of personal and mobile users who regularly backup entire data-sets on their devices. There are two main design considerations for our solution:

- The system should scale to handle numerous number of concurrent backup requests while maintaining high throughput.
- All clients connect to the datacenter through the Internet and therefore the cost of bandwidth (both in energy consumption and capacity constraints) must to be considered.

Based on the above observations, we design the system using a multi-tiered architecture model as shown in Figure 2. The system is made up of four major components:

- *Client Application*, an Operating System -specific application running on client host machines or mobile devices. The main functions of this application are 1) collecting changes in local data, 2) calculating data fingerprints and 3) communicating with the cloud back-up service to selectively upload new data that has not yet been backed up.
- *Web Front-end Cluster*, a highly scalable web front-end cluster responds to requests from the clients and generates an upload plan for each back-up request by querying hash nodes in the hash cluster for the existence of requested data blocks in the cloud storage. If the data doesnt exist, the web cluster transfers the new data to the cloud storage for long term storage. In order to make use of chunk locality [3], [5] of backup data streams, the web front-end aggregates fingerprints from clients and sends them as a batch to hybrid nodes.
- *Hybrid Hash Cluster*, a novel scalable hybrid hash cluster designed for fast response to fingerprint lookup queries. Owing to the nature of the environment in which this service will operate, a large number of simultaneous queries are expected. As a result, the hash cluster is designed for high load-balancing, scalability and minimizing the cost for each query. The hash cluster provides the fingerprint storage and lookup service, and can be viewed as middleware between the clients and the cloud storage.
- *Cloud Storage*, a back-end cloud-based storage service (e.g. Amazon S3) which provides reliable, robust and economical data storage.

Separating the fingerprint store and lookup operations into a different layer from the cloud storage offers, among others, the following advantages:

- Highly optimized and scalable fingerprint storage and lookup mechanism.
- Like any other layered approach, functionality and resources can be added transparently. For example, the hash cluster can easily and transparently be scaled to thousand of nodes.
- Reduced network traffic to the cloud storage backend.
- As a specialized hash engine service, SHHC can connect to and work with any cloud storage service provider.

B. The Hybrid Hash Cluster

To serve simultaneous backup requests from hundreds of thousands of users, a single node hash server indisputably becomes a bottleneck. Therefore, in order to address this issue, we propose a scalable hybrid hash cluster (SHHC) that maintains a low-latency distributed hash table (DHT) [1], [7], [11] for storing data fingerprints and responds to clients as to whether the data already exists in the cloud storage or not.

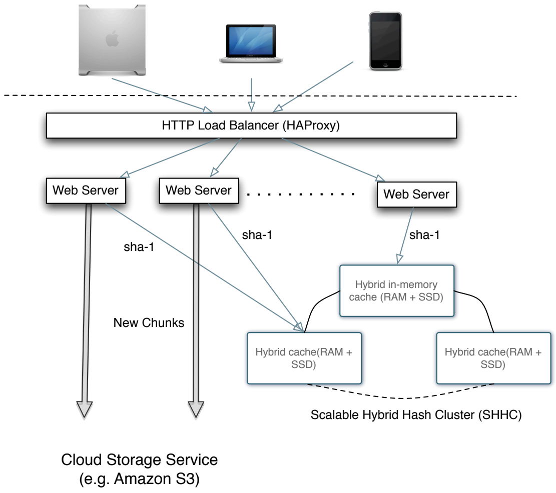
HTTP Load Balancer (HAProxy)
Web Server
Web Server
Web Server
sha-1
sha-1
sha-1
New Chunks
Hybrid in-memory cache (RAM + SSD)
Hybrid cache(RAM + SSD)
Hybrid cache(RAM + SSD)
Scalable Hybrid Hash Cluster (SHHC)
Cloud Storage Service (e.g. Amazon S3)

Figure 2. The overall architecture of the cloud-based back-up service

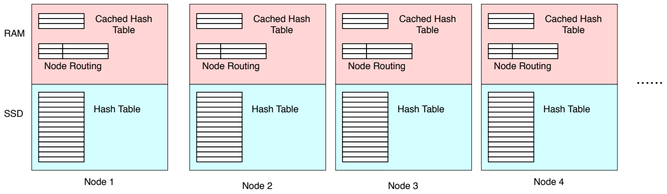

Figure 3. Hash node memory layout

Like the Chord [11] system, the SHHC is made up of a set of connected hash nodes, which hold a range of hash values. However, unlike Chord which was designed for a highly unstructured environment, a signature cluster used in cloud backup service runs in a reasonably structured and relatively static environment and each node is more stable than a node in peer-to-peer systems. Each hybrid node in the cluster is composed of RAM and Solid State Drives (SSD) to take advantage of the fast random access inherent in SSDs (Figure 3). The hash table is stored on the SSD as a Berkeley DB [8] and a bloom filter is used to represent the hash values in the database. A <bloom, Berkeley DB filepath> entry is maintained in RAM and in addition, RAM serves as the cache for SSDs to absorb requests for frequent queries and hide the latency of SSD accesses.

Figure 4 illustrates a typical work flow of SHHC:

1) The client (via the Web Front-end Cluster) sends a fingerprint (SHA-1) to a corresponding hash node N.
2) Node N attempts to locate this fingerprint in main memory. If it exists, node N informs the client that the data block indentified by this fingerprint has already been stored.
3) Otherwise, a read miss is triggered. Node N then tries to locate this fingerprint in the hash table on SSD. If this fingerprint exists on SSD, then node N loads it

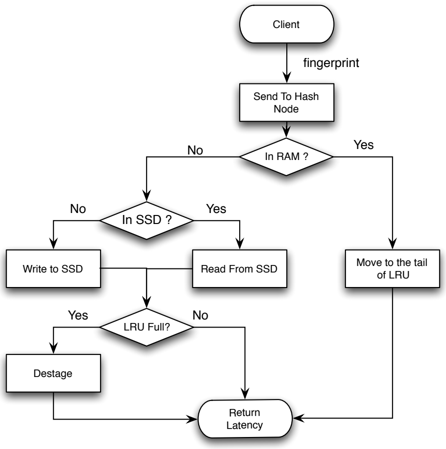

Figure 4. Flow chart of SHHC lookup operation

into RAM and replies to the client. Otherwise node N writes a new entry in the hash table on SSD and notifies the client that the corresponding data does not exist in the cloud and asks the client to transmit the data.

4) Node N maintains a least recently used (LRU) cache list in RAM. If the LRU is full, it discards the least recently used fingerprints.

IV. Preliminary Evaluation

To evaluate SHHC, we developed and deployed a cluster and compared the throughput of different cluster sizes by injecting fingerprints of several real-world I/O workloads.

A. Experimental Setup

All our experiments were conducted on a cluster of 6 machines, each node with an Intel Xeon 2.53 GHz X3440 Quad-core Processor, 4-16GB RAM, SATA II 64GB SSD and 1GB NIC. All the nodes were running GNU/Linux Ubuntu Server 10.10 and were connected via a 1Gb/s Ethernet switch using CAT5e UTP cables. In addition, two machines (different from the cluster nodes) were used as clients to generate and send workload traffic to the cluster.

Using traces of fingerprints from real-world workloads, we issued hash value queries to the distributed hash cluster for different numbers of cluster nodes. Table I shows the fingerprint characteristics of each workload using 8KB chunk size for the Time machine and 4KB for the others. The Time machine workload was collected from an OSX user's data backed up over a period of 6 months. In this table, the bigger the percentage of redundant content, the more duplicated data the workload has. Distance indicates the average distance between similar fingerprints in the list of hash values. Workloads with shorter distances exhibit spatial locality of data blocks.

Table I
Workload characteristics

Workload	Fingerprints	% Redundant	Distance
Web Server [9]	2,094,832	18%	10,781
Home Dir [9]	2,501,186	37%	26,326
Mail Server [9]	24,122,047	85%	246,253
Time machine	13,146,417	17%	1,004,899

B. Scalability and Performance

To assess the scalability of the proposed solution, we fed the aforementioned 4 mixed workloads to different sizes of the hybrid hash cluster (from $1 \sim 4$ nodes) in order to evaluate the throughput of the entire cluster for each particular configuration. For each evaluation, two separate clients running on different machines from the cluster nodes were used to send workloads to the cluster. Moreover, due to the size limitations of each workload, we used cold machines that did not contain any previous data to respond to all hash query requests. It is worth noting that in this setup, each client holds a buffer to aggregate hash queries and send them as a batch to the cluster. Using this batch mode to group hash queries has a two-fold advantage: firstly, it improves network bandwidth utilization and secondly, it preserves the spatial locality of incoming hash requests. The later is usually helpful in improving in-line deduplication throughput [3]. For our experiments, the batch sizes were set to $1/128/2048$ per request.

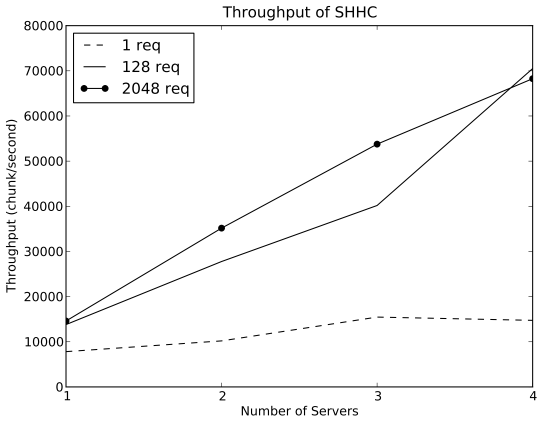

Figure 5. Scalable Throughput

As shown in Figure 5, the overall throughput for larger batch sizes (128 and 2048 requests) is significantly superior (1 order of magnitude) to the case without batch mode (e.g. batch size = 1) for all configurations. The results show good scalability of the cluster-based hash index design. However, when the system scales up, the performance of the 128 and 2048 batch sizes are similar. This is caused by the influences between the multiple I/O streams from clients. We leave it as our future work.

C. Evaluation of Load Balancing

We also evaluated load balancing in our proposed scheme by primarily analyzing the hash table entries stored in each hash node. Figure 6 shows the percentage of the number of hash entries stored in each hash table of each node when the number of server nodes in the cluster is 4. We can see that the number of hash values stored in each server node is roughly 25%, which indicates that our proposed approach is load balanced.

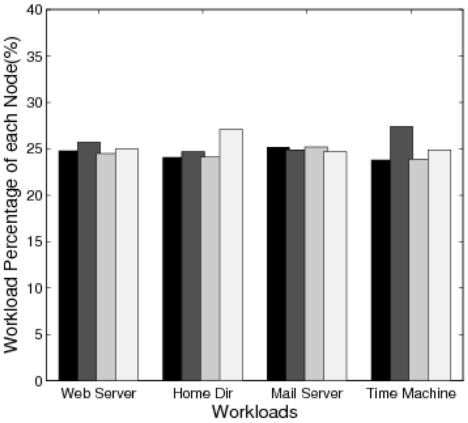

Figure 6. Hash value storage distribution

V. Conclusion and Future Work

Scalability (in various forms) is a key requirement for any cloud service. We identified scalability of the fingerprint store and lookup mechanism as the key issue facing hash based in-line deduplication systems in cloud data centers that have to handle numerous number of concurrent backup requests. To this end, we designed SHHC, a distributed fingerprint store and lookup service which can scale to handle a very large number of concurrent backup clients while maintaining high fingerprint lookup throughput. We also proposed separating this service into a different tier from the data store to allow for flexibility, high optimization and scalability of the hash lookup mechanism.

We evaluated the design with real-world workloads and the evaluation shows that the distributed hybrid hash cluster is consistently scalable as the number of nodes increases. We also evaluated the load balance property of the proposed scheme and the results show that the cluster exhibits good access load balance.

Our future work will, among other things, focus on supporting in-line deduplication for primary storage, dynamic resource scaling and fault tolerance, and energy efficiency of hash operations in cloud deduplication storage systems. Further, in our solution, the hash queries are batched before being sent to the hash cluster. While this aggregation of queries helps us exploit spatial locality of incoming hash requests and also improves throughput on the server side, it introduces latency in the lookup operations. A tradeoff can be obtained as evidenced in our experiments for some batch sizes. We intend to further explore this issue to find a tradeoff between query latency and optimal batch size.

References

[1] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in p2p systems. *Commun. ACM*, 46:43–48, February 2003.

[2] D. Bhagwat, K. Eshghi, D. D. E. Long, and M. Lillibridge. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. Sept. 2009.

[3] 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*, USENIXATC'10, pages 16–16, Berkeley, CA, USA, 2010. USENIX Association.

[4] IDC. The digital universe decade are you ready? http://www.emc.com/collateral/demos/microsites/ idc-digital-universe/iview.htm.

[5] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble. Sparse indexing: Large scale, inline deduplication using sampling and locality. In *FAST'09: Proceedings of the 7th USENIX Conference on File and Storage Technologies*, pages 1–13, Berkeley, CA, USA, 2009. USENIX Association.

[6] D. Meister and A. Brinkmann. dedupv1: Improving deduplication throughput using solid state drives (ssd). In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1 –6, May 2010.

[7] Memcached.org. Memcached: A distributed memory object cache system. http://memcached.org/.

[8] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley db. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 43–43, Berkeley, CA, USA, 1999. USENIX Association.

[9] R. Rangaswami. http://users.cis.fiu.edu/~raju/WWW/.

[10] P. Russell and L. Jinyang. Piccolo: Building fast, distributed programs with partitioned tables. In *OSDI '10: Proceedings of the 9st USENIX conference on Operating Systems Design and Implementation*, pages 1–14, Berkeley, CA, USA, 2010. USENIX Association.

[11] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[12] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *FAST'08: Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.