2012 ACM/IEEE 13th International Conference on Grid Computing

Droplet: a Distributed Solution of Data Deduplication

Yang Zhang*, Yongwei Wu† and Guangwen Yang‡
*†‡Department of Computer Science and Technology,
Tsinghua National Laboratory for Information Science and Technology (TNLIST)
Tsinghua University, Beijing 100084, China
† Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
‡Ministry of Education Key Laboratory for Earth System Modeling,
Center for Earth System Science, Institute for Global Change Studies,
Tsinghua University, Beijing 100084, China
Email: y@yzhang.net*, wuyw@tsinghua.edu.cn†, ygw@tsinghua.edu.cn‡

***Abstract*—Creating backup copies is the most commonly used technique to protect from data loss. In order to increase reliability, doing routinely backup is a best practice. Such backup activities will create multiple redundant data streams which is not economic to be directly stored on disk. Similarly, enterprise archival systems usually deal with redundant data, which needs to be stored for later accessing. Deduplication is an essential technique used under these situations, which could avoid storing identical data segments, and thus saves a significant portion of disk usage. Also, recent studies [1], [2] have shown that deduplication could also effectively reduce the disk space used to store virtual machine (VM) disk images.**

We present droplet, a distributed deduplication storage system that has been designed for high throughput and scalability. Droplet strips input data streams onto multiple storage nodes, thus limits number of stored data segments on each node and ensures the fingerprint index could be fitted into memory. The in-memory finger index avoids the disk bottleneck discussed in [3], [4] and provides excellent lookup performance. The buffering layer in droplet provides good write performance for small data segments. Compression on date segments reduces disk usage one step further.

***Index Terms*—deduplication; storage system; cluster;**

I. Introduction

The primary purpose of backup activities is to recover from data loss. It is a common practice to create multiple copies of important files and store them in peripheral storage devices. Up on data loss, a recent version of lost or corrupted files could be retrieved from backup storage to aid recovering procedure.

Because backup systems contain multiple copies of important data, their storage consumption is considerably high. In order to reduce disk storage consumption, two techniques are used frequently: compression and deduplication. They increase the effective disk storage space and improve backup efficiency. Also, for the same amount of data, less network bandwidth is required to transfer them, which reduces loads on networks.

Compression works by encoding original data segments using fewer bits. Repeating data patterns *within* the data segment are detected and processed. Compression applications such as `zip`, `gzip`, and `bzip2` are frequently used in Linux distributions. They provide good compression ratio within reasonable time frame [5].

Deduplication works by eliminating duplicate data segments, leaving only a single copy to be stored. Deduplication could work at file level, which guarantees no duplicate file exists; or work at finer granularity of block level, which ensures that duplicate data segments within a file could be detected. There are primarily two block level deduplication methods: fixed-length block, which divides data into fixed length data segments; and variable-length block, which locates certain split marks inside data and divides block boundary at those locations. Fixed-length block approach has better performance because of its simple design, while variable-length block approach is more resilient to insertions inside data.

The biggest challenge in deduplication systems, as pointed out in [3], is to identify duplicate data segments quickly. Byte-by-byte comparison is infeasible, as it requires too much IO operation. So most deduplication systems detect duplicate data segments with “fingerprints”. The fingerprints need to satisfy the property that two fingerprints are the same *if and only if* corresponding two data segments are the same. Cryptographic hash functions such as MD5 [6], SHA-1 [7] are best fitted for this purpose. Though collisions are possible for these hash functions, the probability is orders of magnitude smaller than probability of disk corruption [8], [9], thus they are safely ignored.

To check for data duplication quickly, an index over all existing fingerprint is necessary. Because of the properties of cryptographic hash functions, index access pattern is random. If the index is stored on disk, this random access pattern will result in too much random disk IO and will degrade whole system performance significantly. At the time of [3], RAM storage is relatively expensive and has limited size, so they used Bloom filters [10] to store a compressed index in RAM, which could reduce around 99% unnecessary disk IO. A more recent study [4] shows that even with this optimization, the average index lookup is still slower than RAM lookup by a factor of 1000. As RAM gets cheaper and bigger in size, it is possible to store all index into RAM, providing maximum possible deduplication speed. Also, for large data sets where the index cannot fit into one storage node’s RAM, this single storage node would have already become a bottleneck. Under this situation, multiple storage nodes could be deployed to reduce index size and deduplication load on a single node. This architecture of RAM storage is also presented in [11].

Recent findings [12] have shown that deduplication outperforms compression. This is because compression works at a limited range within a file, while deduplication generally works at global scale and has abundant data segment samples. Several other research works [1], [2] also presented the effectiveness of deduplication on VM disk images, and have shown that fixed-length block deduplication reaches similar space saving as variable-length block approach. As virtual machine is considered as an essential component in cloud computing, it indicates that deduplication will continue to be a hot research topic in the following years.

Inspired by all these considerations, we designed droplet, a distributed deduplication storage system that aims for high throughput and scalability. It scatters data blocks to multiple storage nodes, and uses fast compression after data deduplication to achieve maximum disk storage saving. Droplet combines write requests of small data blocks into large lumps to improve disk IO. Experiments have shown that droplet achieves excellent deduplication performance.

The rest of this paper is organized as follows. Section 2 provides system architecture of droplet. Section 3 presents considerations on droplet's design in detail. Section 4 shows evaluation results and discussions. Related work is given in section 5. Finally we conclude this paper in section 6.

II. System Design

Droplet consists of 3 components: a single meta server that monitors whole system status, multiple fingerprinting servers that run deduplication on input data stream, and multiple storage nodes that store fingerprint index and deduplicated data blocks.

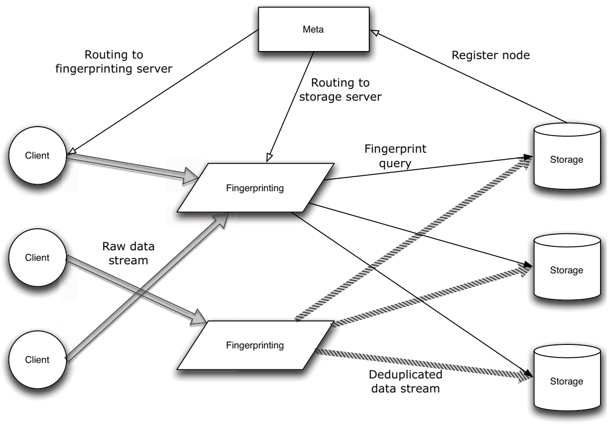

Fig. 1: System architecture

A. Communication Between Components

Meta server maintains information of fingerprinting and storage servers in the system. When new nodes are added into the system, they need to be registered on Meta server first. Meta server provides routing service with this information.

Droplet itself does not provide a backup service. Instead, it provides storage to backup services. A backup server acts as client to droplet. The client first connects to Meta server and queries for list of fingerprinting servers, and then connects to one of them. After that, raw data stream containing backup content will be sent to this fingerprinting server, which calculates data block fingerprints and replies results to the client. After achieving the fingerprints, the backup server could store these fingerprints as reference to deduplicated data.

Fingerprinting servers, when they receive raw data stream from clients, will split data into blocks and calculate their fingerprints. Each data block is tagged with its fingerprint, compressed to a smaller size, and then pushed to a deduplication queue. A daemon process periodically collects all fingerprints in the queue, and checks duplicated fingerprint by querying storage servers. New data blocks not found in storage servers are then flushed to storage servers, while duplicated data blocks are discarded immediately.

The whole fingerprint space is split into 256 buckets according to their first byte. Those 256 buckets are scattered to all the storage nodes. Meta server maintains this information and provides storage server routing to fingerprinting servers. When listing of fingerprints is sent to a storage server, it checks existence of these fingerprints, and replies listing of new fingerprints only, which allows fingerprinting server to discard duplicated data blocks. The storage server is also in charge of storing data blocks sent to it.

B. Client API

Droplet provides a clean interface to client side. The client library caches routing information from Meta server, in order to reduce the CPU load on Meta server and speedup communication process. It automatically switches fingerprinting server after a certain amount of data has been sent, which distributes deduplication load on each fingerprinting server. The demo client code in Python is given below.

```python
dp = droplet.connect(svr_host, svr_port) # establish connection to droplet
f = open(fpath, "rb") # prepare the file to be stored and deduplicated
try:
    result = dp.store(f) # store and deduplicate file contect
    for offst, finger in result:
        # result is a list of data block offset and its fingerprint
        print "%s, %d" % (finger, offst)
finally:
    f.close()
```

Fig. 2: Demo client API usage

III. Implementation Details

A. Fixed Length Block

Droplet deduplication runs at block level. Fixed-length block is used instead of variable-length block, based on a few considerations:

1) Variable-length block needs to locate split marks in data by Rabin fingerprint or similar techniques, which incurs additional calculation overhead. Droplet needs to provide high throughput, thus it need to avoid such additional calculation overhead.
2) The metadata of processed data block on client side is easier to manage when droplet uses fixed-length block. Given an offset in archived data, client side could easily locate corresponding data block by simple calculation. For variable-length block, a more complex algorithm will be needed to locate data blocks from an offset.
3) Deduplication on VM disk images have been studied in [1]. It has shown that for VM disk images, variable-length block and fixed-length block provides nearly the same deduplication ratio. As VM disk images contains all files in a whole system, similar result is expected on system backup data sets. A possible explanation is, data insertion in large file is not common, thus fixed-length block could provide the same deduplication ratio as variable-length block, even though it is not resilient to data insertion.

The deduplication ratio in this paper is defined as:

$$\text{deduplication ratio} = 1 - \frac{\text{total size after deduplication}}{\text{total size of orginal data set}}.$$

B. Block Size Choice

Deduplication block size is a balancing factor, which is very difficult to choose, as it has great impact on both deduplication ratio and IO throughput. Smaller block size will lead to higher deduplication ratio, since duplicate data segment at a finer granularity could be detected. Bigger block size result in fewer interrupts in IO activities and less meta data, and is able to provide indexing service for larger data set with limited RAM capacity.

In order to choose a proper block size, we need to study the properties of input data set. We analyzed all files in a photo gallery website serving around 140,000 pictures, treating them as backup data streams. The distribution of file size is given in Figure 3. It shows that file size in range 64KB$\sim$1MB is most common.

The relationship between block size, deduplication ratio and IO throughput is given in Figure 7 and 8. Based on our experience, droplet uses a block size of 64KB, which provides excellent IO performance and good deduplication ratio.

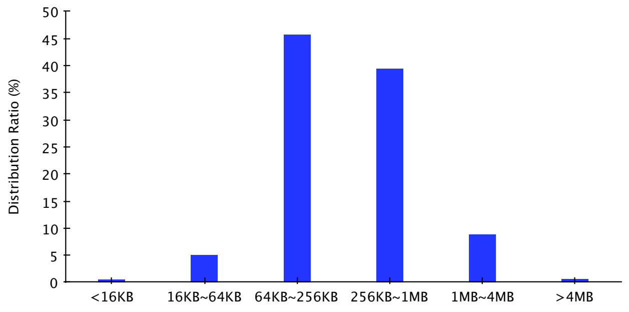

Fig. 3: File size distribution in backup data set

C. Fingerprint Calculation

Fingerprint calculation is the most important procedure in deduplication. Among all well known cryptographic hash functions, MD5 and SHA-1 are frequently used for fingerprint calculation. MD5 provides faster calculation speed, while SHA-1 has smaller collision probability. The output of MD5 and SHA-1 hash is 128 bits and 160 bits.

The fingerprint of each input data block need to be calculated to detect duplication, thus the performance of this procedure has direct impact on overall system throughput. Also, smaller fingerprint means more entries could be fitted into RAM index. Based on these considerations, droplet uses MD5 as its fingerprinting function.

In order to speedup fingerprinting procedure, droplet runs MD5 calculation algorithm in multiple threads to exploit the power of multicore CPUs. As shown in Figure 4, droplet MD5 procedure achieves linear speedup as number of calculation threads increase, until memory bandwidth has been satirized. The speed of SHA-1 hash is also provided for comparison.

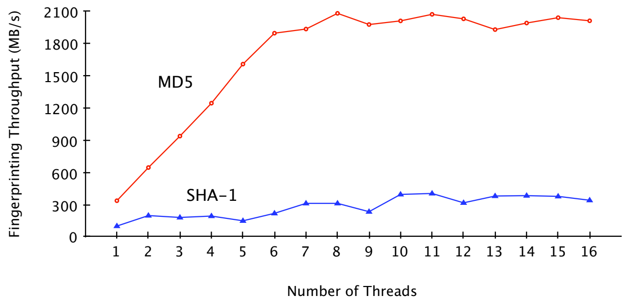

Fig. 4: Multithread fingerprint calculation speed

Similar efforts to speedup MD5 calculation speed have been made. The work in [13] tried to calculate MD5 hash with GPU. Because GPU is relatively expensive and not common on commodity servers, droplet did not adopt such techniques.

D. Fingerprint Index

The fingerprint index is the most important component in droplet. Its performance has direct impact on overall system throughput. In order to achieve maximum querying speed, the index is located inside RAM to avoid any disk IO.

The index is implemented as a hash table, with fingerprint as key. Droplet uses cuckoo hash [14], because it has a much better worst case complexity than other hash table implementations, and uses memory efficiently.

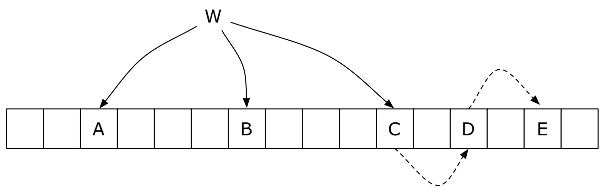

Fig. 5: Cuckoo hash

Figure 5 explains how cuckoo hash works. Cuckoo hash uses a number of hash functions to calculate a set of alternative slots of a given key. When inserting a new element `w`, all of its alternative slots will be checked, and `w` will be put in the first empty slot. If no slot is available, the current element in one of the slots (like `C`) will be "kicked" and replaced by `w`. The kicked element will again be inserted, kicking another element (like `D`) if necessary. If an insertion results in too many collisions, the hash table will be rehashed.

Data blocks are sealed into large lumps of "containers", each has a size of several megabytes. The data blocks are located by its container ID and offset inside the container file. Items in the index has the format `fingerprint` $\rightarrow$ `(container ID, offset)`. Each item is 24 bytes (16 byte MD5 fingerprint, 4 byte container ID and offset). To avoid frequent hash collisions, we ensure the cuckoo hash to work with a load factor below 50%. Even with a small block size of 64KB, a storage server with 24GB memory could easily hold

$$\frac{24\text{GB}}{24\text{B}} \times 50\% \times 64\text{KB} = 32\text{TB}$$

deduplicated data blocks, which is big enough for most use cases. The price for 32TB hard disk (16×2TB) is around \$2000, while the price for 24GB memory (3×8GB) is around \$250, which means the major cost of such a storage server will be on hard disk rather than memory, thus storing the index in memory is cost efficient.

E. Storage for Data Blocks

The design of data blocks storage is given in Figure 6. Small data blocks are packed into large lumps of containers, and flushed to disk in the scale of a whole container. Each container has a size of multiple megabytes, in order to exploit the high sequential IO throughput of hard disk. Each container is referenced by a unique ID, and blocks inside the container are referenced by their offset.

For each block, its size and fingerprint is stored together with data content. The block size is not stored in RAM index,

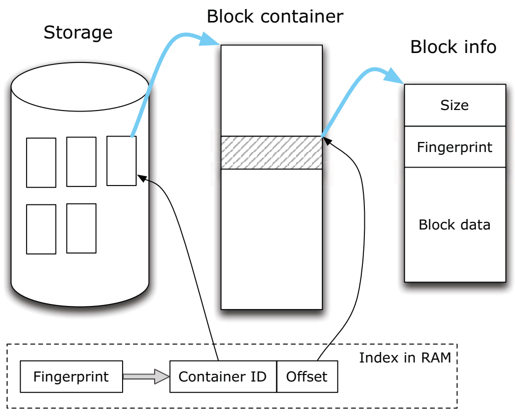

Fig. 6: Data block storage

because it is not necessary when detecting duplication. If a block needs to be retrieved, the storage server could locate the block info with `(container ID, offset)` pair in RAM index, and extract block size before fetching block data. Because disk access cannot be avoided when retrieving data block content, accessing block size info on disk will not introduce additional overhead.

In order to check for disk corruption, the block's fingerprint is also included in the container file. This information could be used when doing integrity check on container files, thus corrupted data blocks could be immediately detected.

F. Compression on Data Blocks

In order to reduce disk storage and network bandwidth consumption at maximum rate, droplet compresses data blocks on fingerprinting servers before sending them to data servers. As modern commodity servers have relatively powerful multicore CPU, the fingerprinting server is usually network saturated first rather than CPU saturated first. It is beneficial to compress data before sending it over network, alleviating burden on network by doing more work on CPU.

Droplet uses Google's snappy compression library [15]. It does not aim for maximum compression ratio; instead, it aims for high speeds and reasonable compression. Snappy is an order of magnitude faster for most inputs than other compression algorithms, but with 20%-100% larger compressed data. This behavior is acceptable for droplet.

TABLE I: Compression algorithm comparison

	Google snappy	**gzip**	**bzip2**
Space saving	34.0%	59.3%	64.2%
Compression speed	237M/s	19.4M/s	2.7M/s
Decompression speed	413M/s	61M/s	13.2M/s

Table I provides performance of Google snappy, `gzip`, and `bzip2` on a 16MB text file.

IV. Evaluation

A. Experiment Environment

To evaluate droplet's design, we conducted a few experiments on a rack of 8 blades connected by 1Gb Ethernet. Each blade has 4 Xeon X5660 CPUs, 24GB DDR3 memory, and a 500GB hard drive (Hitachi HDS721050CLA362). The blades are running Ubuntu 11.04 with Linux kernel version 2.6.38.

B. Block Size Choice

Data block size has a direct impact on disk IO performance and deduplication ratio. A proper choice on data block size need to balance these two factors in order to achieve best result.

The relationship between block size, deduplication ratio and IO performance is given in Figure 7 and 8. As block size increases, the deduplication ratio gradually degrades. In Figure 8, the label "raw" denotes the crude performance of writing deduplicated data blocks to hard disk, while the label "deduplicated" takes the original input data size into account.

It is shown that deduplication speed stabilizes after block size reaches 256KB, which is a good choice for data block size. But in practice we have chosen 64KB as data block size, since it already provides relatively good IO performance, and higher deduplication ratio than 256KB data blocks.

C. Index performance

The fingerprint index, implemented as a cuckoo hash, is the most important component in the system. Performance statics of the fingerprint index is given in Figure 9. As the cuckoo hash holds more and more keys, both insertion and query speed will be affected. The insertion speed drops very quickly after load factor reaches 0.7, because each insertion is likely to result in multiple key relocations. The query speed also drops, since it is more unlikely to find queried key in the first alternative slot, thus more than one memory access will be necessary.

In practice, we limit the load factor below 0.5 to achieve best performance on the fingerprint index.

D. Block Storage Comparison

Disk storage is the bottleneck in droplet. Droplet optimizes disk access by combining small write requests together. This significantly reduces fragmented IO request sent to disk.

Figure 10 presents performance comparison between droplet container, Berkeley DB and Tokyo Cabinet. In this experiment, Berkeley DB and Tokyo Cabinet uses fingerprint as key, and block content as value. Tokyo Cabinet is using B+ tree based API. Droplet's block storage provides highest IO throughput.

V. Related Work

A. Deduplication Systems

Deduplication techniques generally detect redundant objects by comparing calculated fingerprints rather than comparing byte by byte. Cryptographic hash functions such as MD5 and SHA-1 are frequently used for fingerprint calculation. For file content deduplication, the unit chosen is usually a section of the file, and finer-grain unit generally leads to higher deduplication ratio. Fixed size chunking is straightforward, but fails to handle content insertions gracefully. Variable size chunking solves this problem at the cost of complexity and IO performance overhead.

Venti [8] is one of the first papers that have discussed data deduplication in detail. It provides block-based deduplication with variable-length blocks, and uses SHA-1 hash to calculate the fingerprint of data blocks. Venti is a centralized deduplication system, with all data blocks inside one archival server. The index in Venti requires one disk access to test if a given fingerprint exists, which brings performance penalty. Venti mitigates this problem by striping the index across disk arrays, achieving a linear speedup.

Data Domain [3] avoided unnecessary disk index lookups with the help of a Bloom filter in main memory, which represents the set of all existing fingerprints. When inserting a new data block, its fingerprint is checked against the Bloom filter. If the fingerprint is missing, the data block will be directly added without disk index lookup. This optimization is reported to have avoided 99% unnecessary disk access.

ChunkStash [4] goes one step further by storing index on SSD instead of magnetic disk. An in-memory cuckoo hash table is used to index the fingerprint index on SSD. The cuckoo hash table stores compact fingerprint signatures (2 bytes) instead of full fingerprint (20 bytes SHA-1 hash), which tradeoffs between RAM usage and false SSD reads. Compared with magnetic disk index based inline deduplication systems, a speedup of 7x-60x is observed.

Both Data Domain and ChunkStash are centralized deduplication, which means the deduplication server itself will become a performance bottleneck when large amount of data is to be archived.

B. Deduplication for Virtual Machines

One of the most important components in cloud computing technology is virtual machine. Most data centers need to deal with virtual machine images. As a result, deduplication on virtual machines has become a focus in both research and industrial circles.

Previous work of [1], [2] has conducted a few deduplication experiments on common VM images. It is stated that for VM images, the deduplication ratio could be considerably high, and could be better than conventional compression algorithms. Also, deduplication ratio of variable size chunking is nearly the

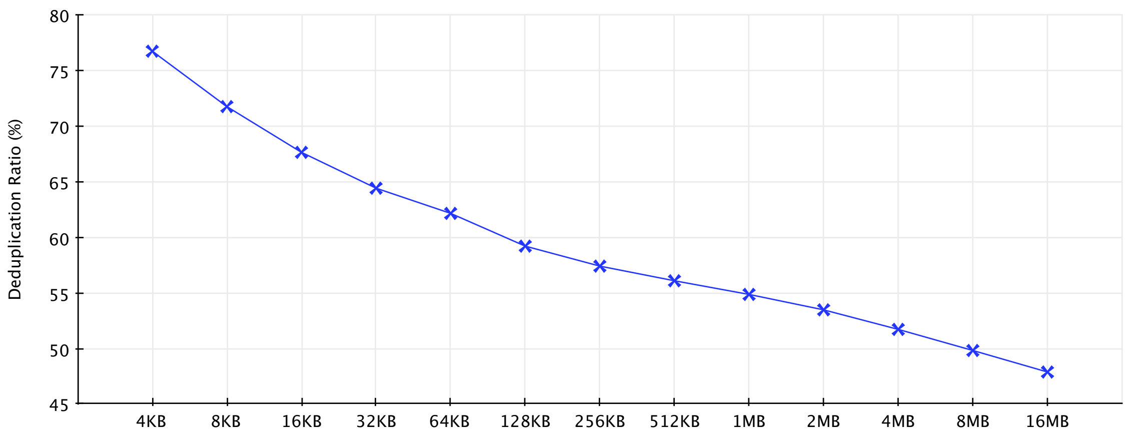

Fig. 7: Deduplication ratio

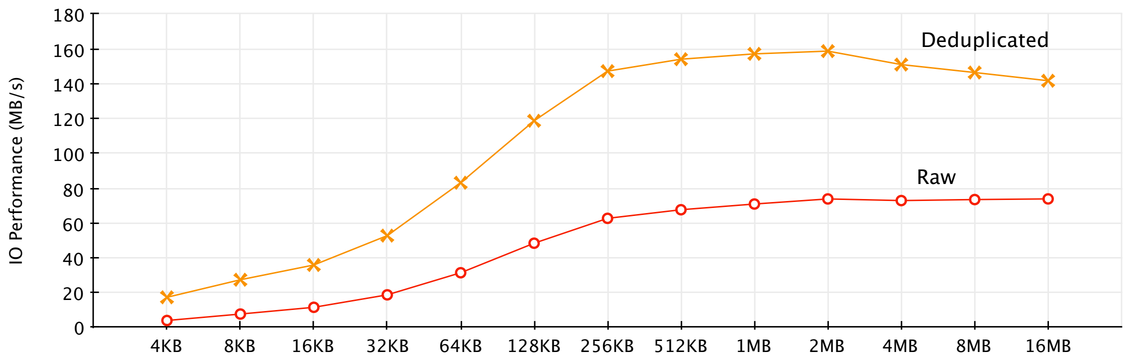

Fig. 8: IO performance

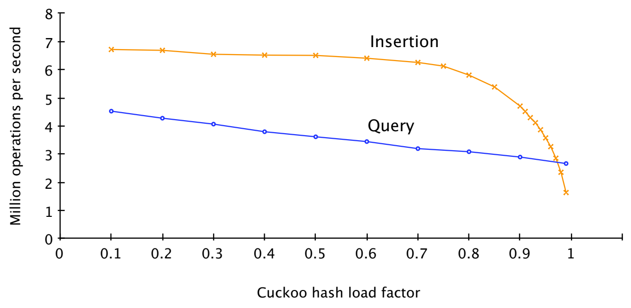

Fig. 9: Fingerprint index performance

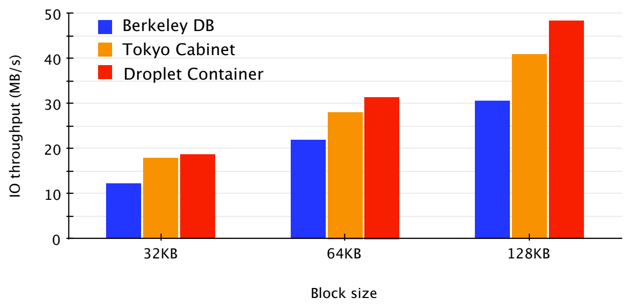

Fig. 10: Block storage comparison

same with fixed size chunking, which means that the simple fixed size chucking scheme is good enough for deduplication on VM images. However, no IO performance result was evaluated in these studies.

Deduplication for VM is not limited to VM images. VMware ESX server adopted content-based paged sharing technique to eliminate redundancy, enabling a more efficient use of hardware memory [16]. Difference engine [17] moved one step further by applying page patching and compression techniques to memory pages, achieving an even higher compression ratio.

As a complete deduplication solution for VM images in a cluster setting, DeDe [18] achieves a high IO speed at around 150MB/s, and supports mixed block size of 1MB or 4KB. DeDe reduces burden on server side by running deduplication on client side, which enlightened the design of droplet However, the data blocks are stored on a centralized SAN, which is not highly scalable when exposed to the growing needs of storing numerous VM images.

C. Network Data Transfer

The benefit of deduplication is not only limited to disk space savings. LBFS [19] reduces amount of data sent over network by means of variable-length deduplication, transferring only the blocks not present on target node. Source code management systems such as `git` [20] also adopt this approach, only transferring data not existing on target machine. These optimizations greatly reduced network bandwidth consumption. Droplet borrows this approach to reduce network traffic between deduplication nodes and storage nodes.

D. Storage for Data Blocks

Haystack [21] is an object storage system designed for Facebook's Photo application. By storing multiple small files into one big file, it avoids additional disk operations incurred by file systems. Haystack is append-only, delete images by special marks, and need to be rewritten to compact spaces occupied by deleted images. The Haystack approach enlightened the design of droplet storage nodes, which combines write requests of small data block into large lumps, and flush them to disk in similar format like Haystack.

E. Distributed Storage Systems

The master node in GFS [22] provides routing service to all client nodes. This results in a high CPU burden on the master node, even though individual request rate from each client node is relatively low. Droplet reduces CPU burden on Meta server by caching the routing table to each fingerprinting server. The fingerprinting servers find corresponding storage server from its cached routing table, and will fall back to querying Meta server when failed to find or connect to the storage servers.

Amazon's Dynamo [23] used DHT [24] to organize its content. Data on the DHT is split into several virtual nodes, and being migrated for load balance in unit of virtual nodes. Droplet follows this approach by splitting data blocks into shards according to their fingerprints, and split them across all storage nodes.

VI. Conclusion

We presented droplet, which is a distributed deduplication system with good IO performance and high scalability. Droplet provides good IO performance by combining multiple write requests of small data blocks into large lumps of data, which exploits the high throughput of sequential IO on magnetic disks. The data blocks are scattered among all storage nodes by their fingerprints. Adding new storage nodes will alleviate RAM consumption of fingerprint index on all storage nodes, and bring linear speedup on throughput.

Droplet achieves maximum deduplication throughput by storing whole fingerprint index inside RAM, completely avoid any disk access when doing fingerprint lookup. Such exclusive usage of RAM storage has also been advocated by [11]. This approach could achieve best performance with reasonable cost, as RAM price is becoming cheaper. The index in RAM contains only minimum necessary information, efficiently utilizing the RAM storage.

Droplet shows that it is worthy to place whole fingerprint index into RAM storage. The system provides best performance with a reasonable cost.

Acknowledgment

This Work is co-supported by National Basic Research (973) Program of China (2011CB302505), Natural Science Foundation of China (61170210, 61073165, 60911130371), National High-Tech R&D (863) Program of China (2011AA01A203, 2011AA040505, 2012AA012600).

References

[1] K. Jin and E. L. Miller, "The effectiveness of deduplication on virtual machine disk images," in *SYSTOR '09: Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*. New York, NY, USA: ACM, 2009, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1145/1534530.1534540

[2] A. Liguori and E. Van Hensbergen, "Experiences with content addressable storage and virtual disks," *Proceedings of the First conference on I/O virtualization*, p. 5, 2008.

[3] 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'08)*. Berkeley, CA, USA: USENIX Association, 2008, pp. 269–282. [Online]. Available: http://www.usenix.org/events/fast08/tech/zhu.html

[4] 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*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 16–16. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855840.1855856

[5] L. Collin, "A quick benchmark: Gzip vs. bzip2 vs. lzma," http://tukaani.org/lzma/benchmarks.html, 2005.

[6] R. Rivest, "The md5 message-digest algorithm," http://tools.ietf.org/html/rfc1321, April 1992.

[7] D. Eastlake, "Us secure hash algorithm 1 (sha1)," http://tools.ietf.org/html/rfc3174, Sep 2001.

[8] S. Quinlan and S. Dorward, "Venti: A New Approach to Archival Data Storage," in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02. Berkeley, CA, USA: USENIX Association, 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=1083333

[9] eXdupe.com, "Risk of hash collisions in data deduplication," http://www.exdupe.com/collision.pdf, 2010.

[10] "Bloom filter," http://en.wikipedia.org/wiki/Bloom_filter, Sep 2011.

[11] J. K. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, "The case for ramclouds: Scalable high-performance storage entirely in dram," in *SIGOPS OSR*. Stanford InfoLab, 2009. [Online]. Available: http://ilpubs.stanford.edu:8090/942/

[12] P. Nath, M. A. Kozuch, D. R. O'Hallaron, J. Harkes, M. Satyanarayanan, N. Tolia, and M. Toups, "Design tradeoffs in applying content addressable storage to enterprise-scale systems based on virtual machines," in *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, ser. ATEC '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267359.1267365

[13] M. Juric, "Notes: Cuda md5 hashing experiments," http://majuric.org/software/cudamd5/, May 2008.

[14] R. Pagh and F. F. Rodler, "Cuckoo hashing," *Journal of Algorithms*, vol. 51, no. 2, pp. 122 – 144, 2004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677403001925

[15] Google, “snappy - a fast compressor/decompressor,” http://code.google.com/p/snappy/.

[16] C. A. Waldspurger, “Memory resource management in vmware esx server,” *SIGOPS Oper. Syst. Rev.*, vol. 36, pp. 181–194, December 2002. [Online]. Available: http://doi.acm.org/10.1145/844128.844146

[17] D. Gupta, S. Lee, M. Vrable, S. Savage, A. C. Snoeren, G. Varghese, G. M. Voelker, and A. Vahdat, “Difference engine: harnessing memory redundancy in virtual machines,” *Commun. ACM*, vol. 53, pp. 85–93, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1145/1831407.1831429

[18] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, “Decentralized deduplication in SAN cluster file systems,” in *Proceedings of the 2009 conference on USENIX Annual technical conference*, ser. USENIX’09. Berkeley, CA, USA: USENIX Association, 2009, p. 8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855815

[19] A. Muthitacharoen, B. Chen, and D. Mazières, “A low-bandwidth network file system,” in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP ’01. New York, NY, USA: ACM, 2001, pp. 174–187. [Online]. Available: http://dx.doi.org/10.1145/502034.502052

[20] “Git (software),” http://en.wikipedia.org/wiki/Git_(software), Sep 2011.

[21] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, “Finding a needle in haystack: facebook’s photo storage,” in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI’10. Berkeley, CA, USA: USENIX Association, 2010, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924943.1924947

[22] S. Ghemawat, H. Gobioff, and S. T. Leung, “The Google file system,” in *Proceedings of the nineteenth ACM symposium on Operating systems principles*, ser. SOSP ’03, vol. 37, no. 5. New York, NY, USA: ACM, Oct. 2003, pp. 29–43. [Online]. Available: http://dx.doi.org/10.1145/945445.945450

[23] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, “Dynamo: amazon’s highly available key-value store,” in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP ’07, vol. 41, no. 6. New York, NY, USA: ACM, 2007, pp. 205–220. [Online]. Available: http://dx.doi.org/10.1145/1294261.1294281

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications,” in *SIGCOMM ’01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 31, no. 4. New York, NY, USA: ACM, Oct. 2001, pp. 149–160. [Online]. Available: http://dx.doi.org/10.1145/383059.383071