EFFICIENT CLOUD BACKUP USING CHUNKING OF DATA

Amal Thankachan[1] | R.Sujitha[2]
[1](UG Student, Christ the King Engineering College, amalthankachan001@gmail.com)
[2](Assistant Professor, Christ the King Engineering College, srisuji14@gmail.com)

Abstract— *Deduplication has become a widely deployed technology in cloud data centers to improve IT resources efficiency. However, traditional techniques face a great challenge in big data deduplication to strike a sensible tradeoff between the conflicting goals of scalable deduplication throughput and high duplicate elimination ratio. We propose AppDedupe, an application-aware scalable inline distributed deduplication framework in cloud environment, to meet this challenge by exploiting application awareness, data similarity and locality to optimize distributed deduplication with inter-node two-tiered data routing and intra-node application-aware deduplication. It first dispenses application data at file level with an application-aware routing to keep application locality, then assigns similar application data to the same storage node at the super-chunk granularity using a hand printing-based stateful data routing scheme to maintain high global deduplication efficiency, meanwhile balances the workload across nodes. AppDedupe builds application-aware similarity indices with super-chunk handprints to speedup the intra-node deduplication process with high efficiency. Our experimental evaluation of AppDedupe against state-of-the-art, driven by real-world datasets, demonstrates that AppDedupe achieves the highest global deduplication efficiency with a higher global deduplication effectiveness than the high-overhead and poorly scalable traditional scheme, but at an overhead only slightly higher than that of the scalable but low duplicate-elimination-ratio approaches*

Keywords— Big Data Deduplication, Application Awareness, Data Routing, Handprinting, Similarity Index

1. INTRODUCTION

Recent technological advancements in cloud computing, internet of things and social network, have led to a deluge of data from distinctive domains over the past two decades. Cloud data centers are awash in digital data, easily amassing petabytes and even exabytes of information, and the complexity of data management escalates in big data. However, IDC data shows that nearly 75% of our digital world is a copy [1]. Data deduplication [2], a specialized data reduction technique widely deployed in disk- based storage systems, not only saves data storage space, power and cooling in data centers, also decreases significant administration time, operational complexity and risk of human error. It partitions large data objects into smaller parts, called chunks, represents these chunks by their fingerprints, replaces the duplicate chunks with their fingerprints after chunk fingerprint index lookup, and only transfers or stores the unique chunks for the purpose of improving communication and storage effi-cie ncy. Data deduplication has been successfully used in various application scenarios, such as backup system [1], virtual machine storage[3], primary storage [4], and WAN replicationBig data deduplication is a highly scalable distrideduplication technique to manage the data deluge under the changes in storage architecture to meet the service level agreement requirements of cloud storage. It is gen-erally in favor of source inline deduplication design, be-cause it can immediately identify and eliminate dupli-cates in datasets at the source of data generation, and hence significantly reduce physical storage capacity re-quirements and save network bandwidth during data transfer. It performs in a typical distributed deduplication [6], [7], [8], [9], [10], [11], [12] framework to satisfy scala-ble capacity and performance requirements in massive data. The framework includes inter-node data assignment from clients to multiple deduplication storage nodes by a data routing scheme, and independent intra-node redun-dancy suppression in individual storage nodes..

2. RELATED WORKS

2.1 CHUNKING OF DATA

Unfortunately, this chunk-based inline distributed de-duplication framework at large scales faces challenges in both inter-node and intra-node scenarios. First, for the inter- node scenario, different from those distributed de-duplication with high overhead in global match query [37], [43], there is a challenge called deduplication node in-formation island. It means that deduplication is only per-formed within individual nodes due to the communication overhead considerations, and leaves the cross-node redundancy untouched. Second, for the intra-node sce-nario, it suffers from the chunk index lookup disk bottleneck. There is a chunk index of a large dataset, which maps each chunk's fingerprint to where that chunk is stored on disk in order to identify the replicated data. It is generally too big to fit into the limited memory of a deduplication node [3], and causes the parallel deduplication perfor-mance of multiple data streams to degrade significantly due to the frequent and random disk index I/Os. There are several existing solutions that aim to tackle the above two challenges of distributed deduplication by exploiting data similarity or locality. Locality means that the chunks of a data stream will appear in approximately the same order again with a high probability. Locality-only based approaches [7], [8], [9] distribute data across deduplication servers at coarse granularity to achieve scalable deduplication throughput across the nodes by exploiting locality in data streams, but they suffer low duplicate elimination ratio due to high cross- node redundancy. Similarity in this context means that two segments of a data stream or two files of a dataset

share many chunks even though they arrive in a random order.

The most similar stored segments or files are prefetched to deduplicate the processing segment or file in low-locality workloads by exploiting a property called logical localitySimilarity-only based methods [6], [20], [30] leverage data similarity to distribute data among deduplication nodes to reduce cross-node duplication, while they also often fail to obtain good load balance and high intra-node deduplication ratio by fingerprint based mapping and allowing some duplicate chunks to be stored. In recent years, researchers [32], [33] exploitbothdata similarity and locality to strike a sensible tradeoff between the con-flicting goals of high deduplication effectiveness and high performance scalability for distributed deduplication

2.2 CONNECTED CAR ENVIRONMENT

• The connected car is receiving much attention as the next- generation Vehicle-IT convergence technology due to the rapid development of mobile communication technology and the ex- pansion of the smart device and application services. Many auto manufactures have been independently developing connected car technologies such as OnStar of GM or Connected Drive of BMW. In addition, with the popularity of a Pay-as-You-Drive insurance, a variety of electronic devices are being sold that connect to the car's OBD2 (On-Board Diagnostics) port and can be used by smartphone applications. In general, a connected car is a vehicle that is always connected to external networks while driving. As in [16] and [18], Our attack model is designed based on an environment where a driver uses a self-diagnostic app to monitor status information after installing an OBD2 scan tool on the vehicle and then pairing it with his/her smartphone by Bluetooth. When the driver installs on his/her smartphone the malicious self- diagnostic app distributed by an attacker, the attacker can launch the actual attack. The attacker can obtain status informa-tion of the vehicle from the malicious self-diagnostic app and use it to inject malicious data into the in-vehicle network. Since the malicio us self-diagnostic app and attacker's server commu-nicate using the mobile communication network (e.g., 3G, 4G, or LTE), the attack is unconstrained by distance. Furthermore, as our attack model uses ECU forced control data commonly used for the same model (more precisely, vehicles with the same configuration of automotive electronic subsystems), it is not necessary to physically occupy the target vehicle in advance. The victim of the target vehicle downloads the malicious self-diagnostic app to his/her smart-phone through an app market. The victim does not recognize that the app is performing malicious acts such as eavesdropping or replay attack on the in-vehicle CAN. In our proposed attack model, we do not consider an attack to compromise the ECU installed on the vehicle inside or an attack to manipulate the firmware of ECU, as these requirements An adversary has access to an automotive diagnostic tool to acquire a CAN data frame to force control of an ECU before launching an actual attack. The attacker can eavesdrop and inject the CAN data frame using a malicious self-diagnostic app into the in-vehicle CAN in the connected car environment. Thus, the attacker does not have to attack the target from a short range. The app may be widely spread through the app markets by masquerading as a legitimate self-diagnostic app for a vehicle.

3 TWO-TIERED DATA ROUTING SCHEME

As a new contribution of this paper, we present the two-tiered data routing scheme including: the file-level appli-cation aware routing decision in director and the super-chunk level similarity aware data routing in clientThe

application aware routing decision is inspired by our application difference redundancy analysis in Section 2.2. It can distinguish from different types of application data by exploiting application awareness with filename extension, and selects a group of dedupe storage nodes as the corresponding application storage nodes, whichhave stored the same type of application data with the file in routing. This operation depends on an application route table structure that builds a mapping between application type and storage node ID. The application aware routing algorithm is shown in Algorithm 1, which performs in the application aware routing decision module of directo

Algorithm 1. Application Aware Routing Algorithm

Input: the full name of a file, fullname, and a list of all dedupe storage nodes {S1, S2, …, SN}
Output: a ID list of application storage node, ID_list={A1, A2, … , Am}

1. Extract the filename extension as the application type from the file full name fullname, sent from client side;
2. Query the application route table in director, and find the dedupe storage node Ai that have stored the same type of application data; We get the corresponding application storage nodes ID_list={A1, A2, … , Am} □□ {S1, S2, ⋯, SN};
3. Check the node list: if ID_list= □□ or all ID_list are overloaded, then add the dedupe storage node SL with lightest workload into the list ID_list={SL};
4. Return the result ID_list to the client.

Algorithm 2. Handprinting Based Stateful Data Routing

Input: a chunk fingerprint list of super-chunk S in a file, {fp1, fp2, … , fpc}, and the corresponding application stor-age node ID list of the file, ID_list={A1, A2, … , Am}
Output: a target node ID, i

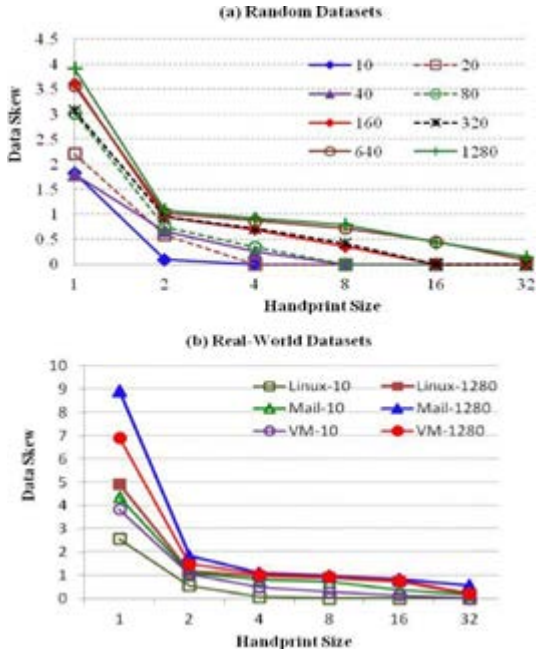
1. Select the k smallest chunk fingerprints {rfp1, rfp2, …, rfpk} as a handprint for the super-chunk S by sorting the chunk fingerprint list {fp1, fp2, …, fpc}, and sent the

handprint to k candidate nodes with IDs mapped by consistent hashing in the m corresponding application storage nodes;
2. Obtain the count of the existing representative finger-prints of the super-chunk S in the k candidate nodes by comparing the representative fingerprints of the previously stored super-chunks in the application-aware similarity index, are denoted as {r1, r2, …, rk};
3. Calculate the relative storage usage, which is a node storage usage value divided by the average storage usage value, to balance the capacity load in the k candidate nodes, are denoted as {w1, w2,
…, wk};
Choose the dedupe storage node with ID i that satisfies ri/wi
= max{r1/w1, r2/w2, …, rk/wk} as the target node.

3.1 Application-aware Deduplication Efficiency
In our design, the client performs data partitioning and chunk fingerprinting in parallel before data routing decision. It can divide the files into small chunks with fix-sized SC or variable-sized CDC chunking methods for each kind of application files, and calculates the chunk fingerprints with cryptographic hash function. Then, hundreds of or thousands of consecutive smaller chunks are grouped together as a super-chunk for data routing. The implementation of the hash fingerprinting is based the OpenSSL library. According to the study in [19], we select SHA-1 to reduce the probability of hash collision for fix-sized SC chunking, while we choose MD5 for variable sized CDC chunking for high hashing throughputwith almost the same hash collision possibility.To exploit the multi-core or many-core resource of the dedupe storage node, we also develop parallel applica-tion-aware similarity index lookup in individual dedupe servers. For our multiple-data-stream based parallel de-duplication, each data stream has a deduplication thread, but all data streams share a common hash-table based application-aware similarity index in each dedupe server. We lock the hash-table based application- aware similarity index by partitioning the index at the application granu-larity to support concurrent lookup. As we demonstrated in [33], the single-node parallel deduplication perform the best in application-aware- similarity-index lookup when the number of data streams equals to that of supported CPU logical cores, while the performance of more streams drops when the number of locks is larger than the num-ber of data stream because of the overhead of thread con-text switching that causes data swapping between cache and memory.
We compare our application aware similarity index with the traditional similarity index in [33] for parallel deduplication throughput in single dedupe storage node with multiple data streams. The results in Fig. 6 show the parallel deduplication throughput using the VM dataset, with data input from RAMFS to eliminate the performance interference of the disk I/O bottleneck. We test the throughput of both traditional similarity index (Naive) and application aware similarity index (Application aware) with cold cache or warm cache, respectively. Here, “cold cache” means the chunk fingerprint cache is empty when we first perform parallel deduplication with multi-ple streams on the VM dataset. While “warm cache”

means the duplicate chunk fingerprint had already been stored in the cache, when we perform parallel deduplication with multiple streams again on the same dataset. We observe that the parallel deduplication schemes with application-aware similarity index perform much better

3.2 Load Balance in Super-Chunk Assignment
Load balance is an important issue in distributed storage technique [41]. It can help improve system scalability by ensuring that client I/O requests are distributed equitably across a group of storage servers. The implementation of consistent hashing based DHTs in traditional distributed deduplication [6], [9], [10] are considerable load
imbalance due to its stateless assignment design. In particular



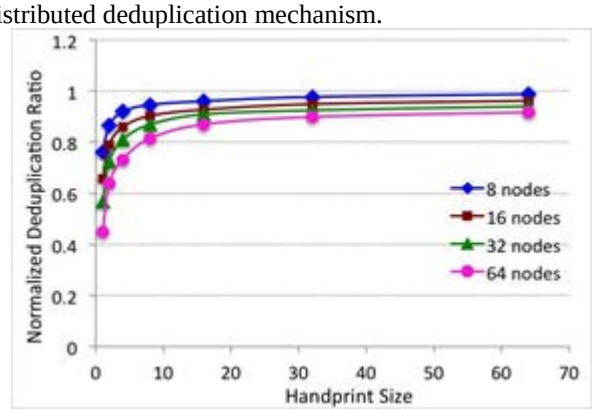
a storage node that happens to be responsible for a larger segment of the keyspace will tend to be assigned a greater number of data objects. In subsection 4.2, we defined DS as a metric for data skew in the storage server cluster. Data assignment method can achieve global load balance when DS=0. The larger DS value we measured inthe test, the more serious load imbalance happened in the storage cluster.

To make our conclusionmore general, we not only consider an ideal scenario that each super-chunk is filled with random data in Fig. 8(a), but also perform the tests on real world datasets: VM images, Linux source code, and mail datasets with 10 and 1280 nodes, respectively, as shown in Fig. 8(b). Super- chunk chooses the least loaded node in

the k storage nodes that are mapped independently and uniformly at random by consistent hashing with the k representative fingerprints in its handprint. The effectiveness of load balancing has been tested with different handprint size and node number in Fig. 8. The average load of each node is 65536 super-chunks with 1 MB size. The important implication of the results is that even a small amount of representative fingerprints in handprint can lead to drastically different results in load balancing. It shows that traditional schemes, like DHT [9] or hash-bucket [10], with only one representative fingerprint for choice are hard to keep a good load balance in the super- chunk assignment. When the cluster scales to hundreds or thousands of nodes, our handprint technique can keep a good load balance (the metric of data skew is less than 1) by routing the super-chunks with k random choices. We select handprint size from 4 to 16 to make a tradeoff between load balance and communica-tion overhead for large-scale distributed deduplication in big data.

5 DISTRIBUTED DEDUPLICATION EFFICIENCY

We route data at the super-chunk granularity to preserve data locality for high performance of distributed deduplication, while performing deduplication at the chunk granularity to achieve high deduplication ratio in each server locally. Since the size of the super-chunk is very sensitive to the tradeoff between the index lookup performance and the distributed deduplication effectiveness, as demonstrated by the sensitivity analysis on super- chunk size in [7], we also choose the super-chunk size of 1MB to reasonably balance the conflicting objectives of cluster-wide system performance and capacity saving, and to fairly compare our design with the previous EMC distributed deduplication mechanism.



ful scheme for a clusterof 128 server nodes on the seven datasets, while this performance margin narrows to 103.6% when averaging over all cluster sizes, from 1 through 128. Stateless routing performs worse than AppDedupe, Produck and Stateful routing due to its low cluster-wide data reduction ratio and unbalanced capaci- ty distribution. Extreme Binning underperforms Stateless routing on the workloads because ofthe large file size and skewed file size distribution in the datasets, work-load properties that tend to render Extreme Binning's similarity detection ineffective. Produck achieves higher normalized EDR than Stateless routing and Extreme Binning due to its stateful routing design, but it underperforms AppDedupe on all datasets for its low deduplication ratio and unbalanced load distribution. AppDedupe outperforms Extreme Binning in EDR by up to 72.7% for a cluster of 128 nodes on the five datasets containing file-level information. For the seven datasets, AppDedupe is better than Stateless routing in EDR by up to 50.5% for a cluster of 128 nodes. Our AppDedupe achieves an improvement of 28.2% and 11.8% in EDR with respect to Produck and -Dedupe on all datasets in a cluster of 128 nodes, respectively. As can be seen from the trend of curves, these improvements will likely be more pronounced with cluster sizes larger than 128

14

plication due to their stateless designs. Stateful routing, on the other hand, must send the fingerprint lookup requests to all nodes, resulting in 1-to-all communication that causes the system overhead to grow linearly with the node number even though it can reduce the overhead in each node by using a sampling scheme. Produck has high communication overhead due to its fine-grained chunk size with 1KB, while other deduplication methods adapt 4KB or 8KB. The number of fingerprint-lookup messages in Produck is about four times that of AppDedupe, Ex- treme Binning and Stateless routing, and it grows as slow as these three low-overhead schemes. As described in Algorithm 1, the main reason for the low system over-head in AppDedupe is that the pre- routing fingerprint-lookup requests for each super- chunk only need to be sent to at most 8 candidate nodes, and only for the lookup of representative fingerprints, which is 1/32 of the num-ber of chunk fingerprints, in these candidate nodes. The message overhead of AppDedupe in fingerprint lookup is about 1.25 times that of Stateless routing and Extreme Binning in all scales. -Dedupe is the preliminary version of our AppDedupe, and they have almost the same communication overhead due to their consistent interconnect protocol.

6. CONCLUSION

In this paper, we describe AppDedupe, an application-aware scalable inline distributed deduplication frame-work for big data management, which achieves a tradeoff between scalable performance and distributed deduplication effectiveness by exploiting application awareness, data similarity and locality. It adopts a two-tiered data routing scheme to route data at the super-chunk granularity to reduce cross-node data redundancy with good load balance and low communication overhead, and employs application-aware similarity index based optimization to improve deduplication efficiency in each node with very low RAM usage. Our real-world trace-driven evaluation clearly demonstrates AppDedupe's significant adventages over the state-of-the-art distributed deduplication schemes for large clusters in the following important two ways. First, it outperforms the extremely costly and poor 7

REFERENCES

[1] Gantz, D. Reinsel, "The Digital Universe Decade-Are You Ready?" White Paper, IDC, May 2010.

[2] Biggar, "Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements," White Paper, the Enterprise Strategy Group, Feb. 2007.

[3] R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, H. Lei. "An Empirical Analysis of Similarity in Virtual Machine Images," Proc. Of the ACM/IFIP/USENIX Middleware Industry Track Workshop (Middleware'11), Dec. 2011.

[4] Srinivasan, T. Bisson, G. Goodson, and K. Voruganti. "iDedup: Latency-aware, inline data deduplication for prima- ry storage," Proc. of the 10th USENIX Conference on File and Storage Technologies (FAST'12). Feb. 2012.

[5] Shilane, M. Huang, G. Wallace, and W. Hsu. "WAN opti- mized replication of backup datasets using stream-informed delta compression," ACM Transactions on Storage (TOS), 8(4): 915-921, Nov. 2012.