APPLICATION-AWARE BIG DATA DEDUPLICATION IN CLOUD ENVIRONMENT

Miss Namitha A R[1], Mahesh R[2], Nayana L V[3], Nischitha L[4], Pavana Shree K M[5]
[1]Assistant Professor, [2,3,4,5]UG Students
Dept of Information Science and Engineering, BGS Institute of Technology, BG Nagar, Mandya-571448

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 handprinting-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**

I. INTRODUCTION

Recent technological advancements in cloud compu-ting, 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 informa-tion, 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-ciency. Data deduplication has been successfully used in various application scenarios, such as backup system [1], virtual machine storage[3], primary storage [4], and WAN replication [5]. Big data deduplication is a highly scalable distributed deduplication 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.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. 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 communica-tion 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..

II. APPDEDUPE DESIGN

In this section, we use the following three design princi -ples to govern our AppDedupe system design:

─ Throughput. The deduplication throughput should scale with the number of nodes by parallel dedupli-cation across the storage nodes.

─ Capacity. Similar data should be forwarded to the same deduplication node to achieve high duplicate elimination ratio.

─ Scalability. The distributed deduplication system should easily scale out to handle massive data vol-umes with balanced workload among nodes.

To achieve high deduplication throughput and good scalability with negligible capacity loss, we design a scal-able inline distributed deduplication framework in this section. In what follows, we first show the architecture of our AppDedupe system. Then we present our two-tiered data routing scheme to achieve scalable performance with high deduplication efficiency.

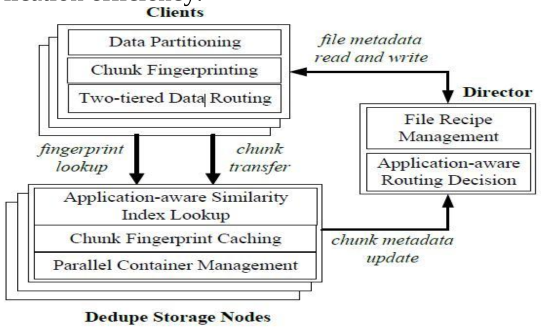

Fig 1:The architectural overview of AppDedupe

A. Interconnect communication

The interconnect communication is critical for the design of AppDedupe. We detail the operations carried out when storing and retrieving a file. A file store request is processed as shown in the Fig. 3: a client sends a PutFileReq message to the director after file partitioning and chunk fingerprinting. The message includes file metadata like: file ID (the SHA-1 value of file content), file size, file name, timestamp, the number of super -chunk in the file and their checksums. The director stores the file metadata as a file recipe [34], and makes sure that there has enough space in the distributed stor-age systems for the file. It also performs the applicationaware routing decision to select a group of corresponding application storage nodes for each file. The director re-plies to the client with the file ID and a corresponding application storage node list in PutFileResp message. After received the PutFileResp, the client sends k LookupSCReq requests to the k candidate dedupe storage nodes for each super-chunk in the file, respectively, to lookup the appli-cation-aware similarity index in dedupe storage nodes for the representative fingerprints of the super-chunk. These candidate nodes reply to the client with a weighted re-semblance value for the super-chunk. The client selects a candidate node as the target route node to store the su-per-chunk, and notifies the director its node ID by PutSC Req message. Then, the client sends all chunk fingerprints of the super-chunk in batch to the target node to identify whether a chunk is duplicated or not.

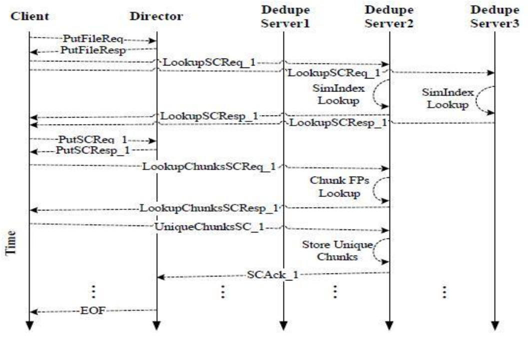

Fig 2:Message Exchange for Store Operation.

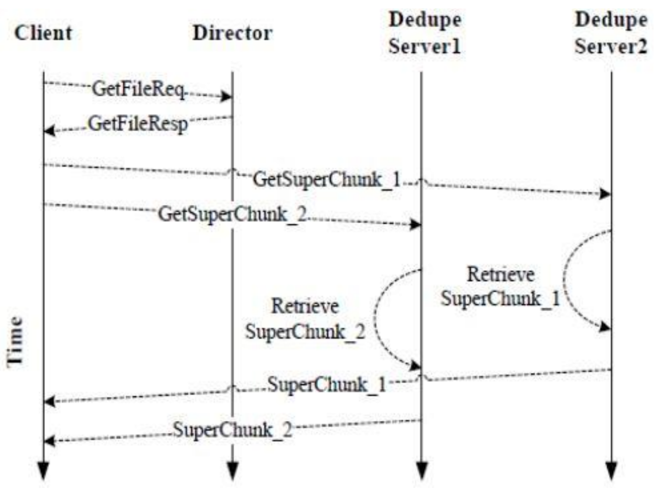

Fig3: Message Exchange for retrieve operation.

The process of retrieving a file is also initiated by a cli-ent request GetFileReq to the director, as depicted in Fig. 4. The director reacts to this request by querying the file recipe, and forwards the GetFileResp message to the client. The GetFileResp contains the super-chunk list in the file and the mapping from super-chunk to the dedupe stor-age node where it is stored. Then, the client requests each super-chunk in the file from the corresponding dedupe storage node with GetSuperChunk message. The dedupe server can retrieve super-chunk from data containers, and the performance of restore process can be accelerated, like [35] [36]. Finally, the client downloads each super-chunk and uses the checksums of super-chunks and file ID to verify the data integrity.

III. PERFORMANCE EVALUATION

We have implemented a prototype of AppDedupe in user space using C++ and pthreads, on the Linux platform. We evaluate the parallel deduplication efficiency in the multi-core deduplication server with real system implementa-tion, while use trace-driven simulation to demonstrate how AppDedupe outperforms the state-of-the-art distrib-uted deduplication techniques in terms of deduplication efficiency and system scalability. In addition, we conduct sensitivity studies on chunking strategy, chunk size, su-per-chunk size, handprint size and cluster size.

A. Evaluation Platform and Workload

In our prototype dedup-lication system, 7 desktops serve as the clients, one server serves as the director and the other three servers for dedupe storage nodes. It uses Huawei S5700 Gigabit Ethernet switch for internal communication. To achieve high throughput, our client component is based on an event-driven, pipelined design, which utilizes an asyn-chronous RPC implementation via message passing over TCP streams. All RPC requests are batched in order to minimize the round-trip overheads. We also perform event-driven simulation on one of the four servers to evaluate the distributed deduplication techniques in terms of deduplication ratio, load distribution, memory usage and communication overhead.

B. Evaluation Metrics

The following evaluation metrics are used in our evalua-tion to comprehensively assess the performance of our prototype implementation of AppDedupe against the state-of-the-art distributed deduplication schemes.

Deduplication efficiency(DE): It is first defined in [22], to measure the efficiency of different dedupe schemes in the same platform by feeding a given dataset. It is calculated by the difference between the logical size L and the physical size P of the dataset divided by the deduplication process time T. So, deduplication efficiency can be expressed in (7).

Normalized deduplication ratio(NDR): It is equal to the distributed deduplication ratio (DDR) divided by the single-node deduplication ratio (SDR) achieved by a sin-gle-node, exact deduplication system, and can be ex-pressed in (8). This is an indication of how close the deduplication ratio achieved by a distributed deduplica-tion method is to the ideal distributed deduplication ratio.

Normalized effective deduplication ratio(NEDR): It is equivalent to normalized deduplication ratio divided by the similar to the metric used in [7]. Normalized ef-fective deduplication ratio can be expressed in (9). It indi-cates how effective the data routing schemes are in elimi-nating the deduplication node information island.

Number of fingerprint index lookup messages: It in-cludes that of inter-node messages and intra-node mes-sages for chunk fingerprint lookup, both of which can be easily obtained in our simulation to estimate communica-tion overhead.

RAM usage for intra-node deduplication: It is an es-sential system overhead related to chunk index lookup in dedupe server. And it indicates how efficient the chunk index lookup optimization is to improve the performance of intra-node deduplication.

IV. CONCLUSIONS

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 deduplica-tion effectiveness by exploiting application awareness, data similarity and locality. It adopts a two-tiered data routing scheme to route data at the super-chunk granular-ity 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 adven-tages 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-ly scalable stateful tight coupling scheme in the cluster-wide deduplication ratio but only at a slightly higher system overhead than the highly scalable loose coupling schemes. Second, it significantly improves the stateless loose coupling schemes in the cluster-wide effective de-duplication ratio while retaining the latter's high system scalability with low overhead.

V. ACKNOWLEDGMENTS

This research was partially supported by the National Key Research and Development Program of China under Grant 2016YFB1000302.

REFERENCES

[1] W. Xia, H. Jiang, D. Feng, Y. Hua "SiLo: A Similarity-Locality based Near-Exact Deduplication Scheme with Low RAM Overhead and High Throughput", 2011.
[2] P. Shilane, M. Huang, G. Wallace, and W. Hsu "WAN optimized replication of backup datasets using stream-informed delta compression", 2012.
[3] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti "iDedup: Latency-aware, inline data deduplication for primary storage",2012.
[4] Y. Fu, H. Jiang, N. Xiao "A scalable inline cluster deduplication framework for big data protection",2012.
[5] D. Frey, A-M. Kermarrec, K. Kloudas "Probabilistic deduplication for cluster-based storage systems",2012.
[6] H. Kaiser, D. Meister, A. Brinkmann, S. Effert "Design of an Exact Data Deduplication Cluster",2012.
[7] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, L. Xu "Application-Aware Local-Global Source Deduplication for Cloud Backup Services of Personal Storage",2012.
[8] Y. Fu, H. Jiang, N. Xiao, L. Tian, F. Liu, "AA-Dedupe: An Application-Aware Source Deduplication Approach for Cloud Backup Services in the Personal Computing Environment",2012.
[9] D. Meister, A. Brinkmann, T. Sub "File Recipe Compression in Data Deduplication Systems",2103.
[10] M. Lillibridge, K. Eshghi, D. Bhagwat "Improving Restore Speed for Backup Systems that Use Inline Chunk-Based Deduplication",2013.