Elastic Data Routing in Cluster-based Deduplication Systems

Yufeng Wang Shaojie Tang Chiu C. Tan
Department of Computer & Information Sciences
Temple University
Philadelphia, PA, USA
Email: {Y.F.Wang,shaojie.tang,cctan}@temple.edu

***Abstract*—As a space-efficient approach to data archive and backup, data deduplication is becoming increasingly popular in storage systems. However, as the data growing rapidly in data centers, single-node storage node is no longer be able to provide the corresponding throughput and capacities as expected. Building deduplication clusters is considered as a promising strategy to leverage such bottle-neck on single-node system. However, deduplication relies on how much the system knows about information of previous stored data. The single-node system obviously obtains all such information and is able to detect duplicate data there; however storage nodes in cluster-based system cannot know information on other nodes. It is nontrivial to route data intelligently enough so that the system could support deduplication performance comparable to that of a single-node system, while also at a trivial cost. In this paper, we propose an elastic data routing strategy, aiming to achieve deduplication performance comparable to state-of-the-art, while require much less computation resources.**

I. Introduction

To mitigate storage costs associated with backing up such huge volumes of data, data deduplication is used. Data deduplication identies and eliminates duplicate data. Storage space requirements can be reduced by a factor of 10 to 20 or more when backup data is deduplicated [1].

Generally, deduplication is put into two categories [2]: In-line deduplication and Post-process deduplication. In the in-line deduplication, hash calculations happen on the target device as the data enters the device in real time. If the device spots a block that is already stored on the system, it marks such a block as duplicate and only saves its metadata. The benefit of in-line deduplication over post-process deduplication is that it requires less storage, as well as lighter networking occupation. Therefore the mainstream of deduplication is in-line deduplication and we only concern this situation.

To step further, deduplication is well understood for storage systems installed in centralized single-node environments. However, as the data grows rapidly, cluster-based storage systems [3] become as the solution to remedy the capacity limitation on single-node systems.

As known that in a centralized single-node environment, all the incoming data arrives in a single storage point, which certainly offers it the ability of detecting duplicate data globally by keeping a single index table. However, when the deduplication system is set in a cluster-based environment, storage breaks up into different locations, incoming data has to choose one or several of them as the storage host. Inappropriate selection will result in poor deduplication performance.

Therefore the challenge relies on the strategy of data routing for deduplication. Simple interactions among all storage nodes might bring unexpected overhead; also it's unlikely that clients can know storage information on nodes in advance. Work from [4] proposed a brute-force data routing strategy which is able to achieve a comparable deduplication performance to single-node system, however with extremely high consumption of computation resources. A 32-node system requires 96 GB amount of memory on the master node for Bloom Filter. The looks-up for 5-TB dataset can reach more than 20 billion. We propose to derive an elastic data routing strategy, which is able to provide comparable deduplication performance to the state-of-the-art [4], however at a much lower cost.

II. Elastic Data Routing Strategy

We hereby propose an elastic data routing strategy based on the idea of exploration and exploitation [5]. Suppose there are k storage nodes in the system, we distribute incoming data to every storage node i ($i = 1, 2, ...k$) for initial m routing operations. (We tentatively consider incoming data as unencrypted and uncompressed [6]). Such process will lead storage nodes to receive more data than expected, but it's supposed to help identify proper routing destination in the future process, also help identify more duplication due to extra stored data. We're still working on validation and optimization of such process, currently we simply apply such m operations at the start of data processing. After m^{th} operation, we choose the node with the highest value of entry $\zeta(t)$ in *Data Routing Index* as the data receiver, if it is not overloaded. When a node finishes processing incoming data, its statistics of amount of total data received $\sum_{j=1}^{t} T_i(j)$, duplication detected $R_i(t)$ as well as storage usage $U_i(t)$ on each node till t^{th} operation will be used to update the *Data Routing Index* on master node.

At the meantime, the load balance needs to be considered, a few nodes are overloaded while the others stay idle will lead to a high data skew. Using static threshold to simply exclude nodes that are above such threshold value is effective [4]. We are still exploring a more adaptive way for dealing with overloaded nodes in our approach, though currently we apply this simple way to identify and exclude overloaded nodes.

We define the average storage usage of all nodes as $\mu(t)$,

$$\mu(t) = \frac{\sum_{i=1}^{k} \sum_{j=1}^{t} T_i(j)}{k}$$

And use λ to denote the overload threshold. The content of entry slots in *Data Routing Index* as

$$\zeta_i(t) = \frac{R_i(t)}{\sum_{j=1}^{t} T_i(j)}$$

The details are also illustrated in Alg. 1. The main advantage of such a technique is the opportunity to incorporate past deduplication performance and capacity balancing while assigning segments to nodes. On the other hand, computation or communication overhead is trivial, there are only k entry slots in *Data Routing Index*!

Algorithm 1 Data Routing Strategy

for Incoming data segment Seg_t **do**
  if $t \leq m$ **then**
    Send Seg_t to all storage nodes
  else
    Check *Data Routing Index*
    if $U_x(t) < \lambda \cdot \mu(t)$ **then**
      Send Seg_t to the node x, where $x = \arg\max_{x \in 1...k}(\zeta_x(t))$
  After storage node(s) finish processing Seg_t
  $U_i(t)$ and $\zeta_i(t)$ are updated in *Data Routing Index*

III. Evaluation

Our dataset is 77.35GB in total, consisting of 4 virtual machines, each of them contains wiki dump files [7], fMRI dataset [8], Internet browsing history and system updates at different amounts. We assume that each storage node has enough physical space. We evaluate our solution, denoted as *Elastic*, against the other two alternative approaches, *Stateless* and *Statefull*, representing the state-of-the-art [4].

We use the RabinHashFunction64 library [9] to perform the Rabin fingerprint used in the variable block deduplication. The average chunk size is set to be 8 KB. We set the size of segment, which is the basic data unit to be routed, as 16 MB. We deploy *Stateless* approach by simply using the result of 160-bit hash value of the first chunk compared to number of nodes. But implement *Statefull* as described in [4]. Fig. III shows the Normalized Effective Deduplication Ratio of above three strategies. Effective Deduplication is defined as the ratio of overall deduplication ratio divided by data skew, as a measurement that encompasses both deduplication effectiveness and storage balance. It reflects the overall deduplication performance while every node consumes the same amount of storage space. This is important because the whole cluster performance degrades while data skew becomes large [10].

We can tell that our approach indeed over-performs *Stateless* approach, though cannot be always comparable to the performance of *Statefull*. We argue that our approach is much lightweight on the computation. In above example, *EAD* conducts 58,168 times of *Data Routing Index* searching, however

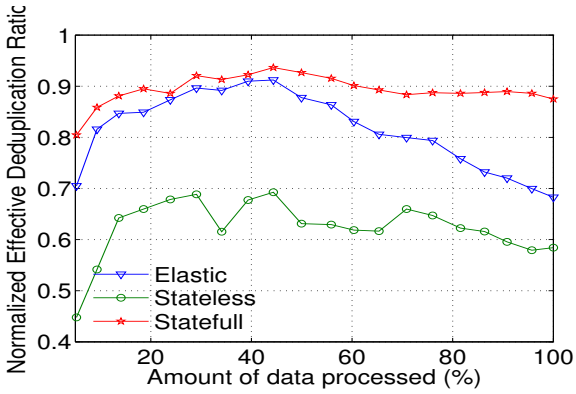

Fig. 1. Normalized Effective Deduplication Ratio Comparison for a 32-node system. Overload threshold $\lambda = 1.05$ and exploration times $m = 1$

the *Statefull* requires 260 million times of FP looks ups from Bloom filter. Though such huge overhead could be relieved by applying sampling technique, it may harm the performance, especially for data which is lack of locality. Compared with *Stateless*, which needs calculation on hash values, at most 32 times of index searching in *EAD* will not claim much computational difference.

IV. Conclusion and Future Work

In this work, we present the preliminary work for an elastic data routing strategy, which is able to achieve higher deduplication ratio than simple stateless data routing approach, with almost the same overhead. However the formulation of our strategy is till not complete, as we discussed in section II. Besides, a wider scope of implementation on more scalable amount of data is needed to validate its effectiveness.

Acknowledgements

This research was supported in part by NSF grants CNS 1156574. We would like to thank Jake Roemer and Reeve Groman for their work on the simulation software.

References

[1] H. Biggar, "Experiencing data de-duplication: Improving efficiency and reducing capacity requirements," *The Enterprise Strategy Group*, 2007.
[2] "Inline vs. post-processing deduplication appliances," Accessed in 03/2013, http://Searchdatabackup.techtarget.com.
[3] G.-w. You, S.-w. Hwang, and N. Jain, "Scalable Load Balancing in Cluster Storage Systems," in *Proceedings of the 12th International Middleware Conference*, 2011.
[4] W. Dong, F. Douglis, K. Li, R. H. Patterson, S. Reddy, and P. Shilane, "Tradeoffs in Scalable Data Routing for Deduplication Clusters." in *FAST*, 2011.
[5] J. G. March, "Exploration and exploitation in organizational learning," *Organization science*, 1991.
[6] Y. Wang and C. C. Tan, "Looking at the overheads of transmitting encrypted data to the cloud," in *Communications and Network Security (CNS)*, 2013.
[7] (Accessed in 09/2013) Database dump progress. [Online]. Available: http://dumps.wikimedia.org/backup-index.html
[8] "OpenfMRI," Accessed in 08/2013, https://openfmri.org/data-sets.
[9] (Accessed in 02/2013) RabinHashFunction64. [Online]. Available: http://rabinhash.sourceforge.net/com/planetj/math/rabinhash/RabinHashFunction64.html
[10] M. A. Alsaih, R. Latip, A. Abdullah, and S. K. Subramaniam, "A Taxonomy of Load Balancing Techniques in Cloud Computing." *International Review on Computers & Software*, 2013.