Improving Multi-Node Deduplication Performance for Interleaved Data via Sticky-Auction Routing

Kave Eshghi, Mark Lillibridge, Deepavali Bhagwat, Mark Watkins

HP Laboratories
HPL-2015-77

Keyword(s):

deduplication; routing; load-balancing

Abstract:

High capacity, high throughput, chunk-based inline deduplication systems for backup have been commercially successful, but scaling them out has proved challenging. In such multi-node systems, the data needs to be routed at a large enough granularity to sustain locality at the back ends. Two routing algorithms, Min Hash and Auction, have been put forth for this purpose. We demonstrate that these algorithms perform poorly on interleaved data. Interleaved data occurs when multiple streams are multiplexed into a single high-speed stream to speed up backups. Of particular commercial importance, database backup procedures produce such interleaved data, where multiple threads read database files in parallel. We present a new routing algorithm, Sticky Auction routing, that, unlike existing algorithms, handles interleaved data with little deduplication loss. It also achieves comparable or better deduplication performance for non-interleaved data and good load balancing, especially when multiple streams are used, the typical case.

External Posting Date: September 6, 2015 [Fulltext] Approved for External Publication
Internal Posting Date: September 6, 2015 [Fulltext]

Improving Multi-Node Deduplication Performance for Interleaved Data via Sticky-Auction Routing

Kave Eshghi[†], Mark Lillibridge[†], Deepavali Bhagwat[♯,*] and Mark Watkins[♯]
[†]*HP Labs* [♯]*HP Storage*

Abstract

High capacity, high throughput, chunk-based inline deduplication systems for backup have been commercially successful, but scaling them out has proved challenging. In such multi-node systems, the data needs to be routed at a large enough granularity to sustain locality at the back ends. Two routing algorithms, Min Hash and Auction, have been put forth for this purpose. We demonstrate that these algorithms perform poorly on *interleaved data*. Interleaved data occurs when multiple streams are multiplexed into a single high-speed stream to speed up backups. Of particular commercial importance, database backup procedures produce such interleaved data, where multiple threads read database files in parallel.

We present a new routing algorithm, *Sticky Auction* routing, that, unlike existing algorithms, handles interleaved data with little deduplication loss. It also achieves comparable or better deduplication performance for non-interleaved data and good load balancing, especially when multiple streams are used, the typical case.

1 Introduction

High capacity, high throughput, chunk-based inline deduplication systems, primarily aimed at deduplicating backup streams, have been successfully introduced to the market by multiple vendors. These systems achieve high scale (e.g., 1 GB/s per input stream, 100s of TBs of raw capacity) throughput economically by holding only a fraction of chunk hashes in RAM and selectively retrieving hashes from disk based on chunk locality [7, 10].

Scaling out such deduplication systems to multiple nodes with global deduplication (*i. e.*, all data remains available to deduplicate against) while keeping their cost advantage has proved challenging. To maintain chunk locality upon which their cost advantage depends, each incoming stream needs to be routed between nodes in such a way that locality is sufficiently preserved for each node. The only known solution [3, 5] is to break each stream of chunks into much larger *segments*, and then route each segment in its entirety to a single node such that similar segments (*i. e.*, segments that have many chunks in common) are routed to the same node with high probability. Then each node can use its locality-based deduplication scheme locally to economically find identical chunks. A small amount of deduplication is expected to be lost due to routing inefficiencies because nodes do not deduplicate against data mistakenly sent to a different node.

Two routing algorithms have been proposed for this purpose: *Min Hash* [3], which routes segments based solely on their contents, and *Auction* [5], which routes segments based on which nodes currently have data most similar to the current segment. They have been demonstrated to work well on many types of backup data.

However, as we show, they perform poorly on *interleaved data*, which occurs when multiple streams are multiplexed to produce a single high speed stream. Interleaving is recommended as a 'best practice' technique for speeding up database backups by major database vendors [1, 2]: backup windows can be shortened by having multiple threads read tables in parallel, multiplexing the results into a single stream. Databases are of particular commercial importance in the backup market since protecting database applications is seen as top priority [9].

We present a new routing algorithm, *Sticky Auction* routing, which, unlike the existing algorithms, handles interleaved data with little deduplication loss and achieves comparable or better deduplication performance for non-interleaved data. It also achieves good load balancing, especially when multiple streams are used, the typical case.

The rest of the paper is organized as follows: the next section provides the background. Section 3 demonstrates the problem caused by interleaved data. Section 4 describes our approach, Sticky-Auction routing. Section 5

[*] Author is now at PernixData.

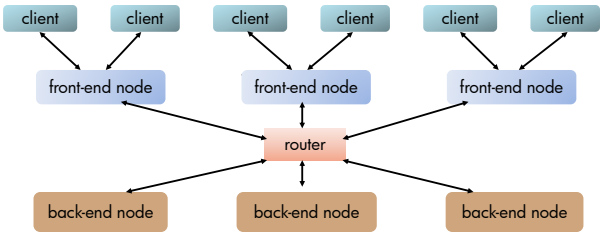

Figure 1: **A multi-node deduplication architecture.**

demonstrates the effectiveness of Sticky-Auction using simulation experiments with real and synthetic data. We conclude in Section 6.

2 Background

There are a number of ways to architect a system that uses these routing algorithms to scale out economically. For illustration, we show a simple one in Figure 1. Here, there are a number of clients of the system connected to front-end nodes. Communication is stream-based, so clients stream backup data to be stored to a front-end node. The front-end nodes break up the incoming streams into small (≈4–8 KB) variable-size chunks. Each stream of chunks is then grouped into ≈1 MB segments using a variable-size segmentation algorithm [5, 7]. Similar to variable-sized chunking, variable-size segmentation breaks segments using content landmarks (chunk hashes in this case) so that local changes tend not to disturb segment boundaries.

A router running one of the routing algorithms is used to route each segment to one of the back-end nodes. Alternatively, each front-end node could run a separate router instance; this is possible because these routing algorithms maintain no or only stream-local state and each stream passes through only one front-end node.

Each back-end node is responsible for performing local deduplication of and storing the segments it receives. It deduplicates every segment locally, against only the data that it stores, not data stored at any other back-end node. The system as a whole, nonetheless, performs global deduplication because similar segments with high probability do get deduplicated against each other. Each back-end node operates autonomously, with local indexing, storage, garbage collection, and management of data.

2.1 The Min-Hash algorithm

Min Hash routes a segment based on the smallest hash of any of its chunks. By Broder's theorem [4], two segments that are highly similar—*i. e.*, have many chunks and hence chunk hashes in common—have the same minimum hash with high probability [3]. Taking the maximum hash instead or the only chunk meeting a landmark condition (e.g., the first hash in Dong *et al.* [5]) works similarly. Given m, the minimal hash of a segment's chunks, a destination back-end node for that segment can be determined by numbering the back-end nodes $0 \ldots N-1$ and choosing the back-end node numbered $m \bmod N$.

Min Hash is a *stateless* routing algorithm. That is, where a segment gets routed depends only on the contents of that segment and not on the state of the back-end nodes. Auction, by contrast, is *stateful*, because where a segment gets routed depends on the current contents of the back-end nodes. The same segment could get routed to different nodes at different times with Auction in some corner cases. This means that routing results must be remembered with Auction but not Min Hash.

2.2 The Auction algorithm

In its simplest form, the Auction algorithm broadcasts (a sample of) the hashes of the segment to be routed to all of the back ends. Each back end returns a *bid* indicating how well it expects that segment to deduplicate against its data. The segment is then routed to the back end with the highest bid. If no back end bids high enough, the segment is routed to a random node. Ties are broken randomly.

The idea is that the first time we see some data, we put it somewhere convenient for load-balancing, and thereafter when we see it again we send it to the same place. For speed reasons, bid generation is usually done via an approximation method that does not require accessing disk. For example, with a system based on Zhu *et al.* [10], the number of hashes that hit in a node's Bloom filter can be used, whereas with a system based on sparse indexing [7], the number of hashes that hit in a node's sparse index can be used.

We consider a somewhat more sophisticated version of Auction in this paper, where the random decisions above are replaced by choosing first in favor of the least-loaded node, defined as the node currently using the least of its raw capacity (*i. e.*, size of deduplicated data). Any remaining ties are broken randomly. Preferring the least-loaded node improves how evenly data is distributed across the nodes (load balancing). Dong *et al.* [5] suggest a number of additional modifications (e.g., weighing bids, ignoring relatively overloaded nodes, rebalancing nodes) to improve load balancing still further that we do not consider in this paper.

In general, Auction has been found to perform better than Min Hash on deduplication quality and load balancing at the cost of requiring more communication and

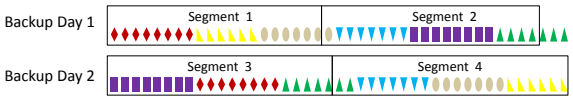

Figure 2: **Effect of interleaving.**

computation [5].

3 The Problem

Interleaving, a process in which multiple streams of data are multiplexed into one stream, is done to create a high-speed stream from multiple low-speed streams. For example, multiple slow disks may be read in parallel and their data merged to into a single high speed backup stream. Interleaving is not a deterministic process: the order in which data pages from different streams are merged depends on factors that differ moment to moment (e.g., relative thread/disk speeds, I/O budget of the RAID array).

Interleaving is commonly used to speed up backing up databases in order to honor limited backup windows [1, 2]. Here, one backup stream is created from multiple database files stored on separate disks. Rather than reading files sequentially, one thread is assigned to each disk. Each thread reads the files on its disk, filling up a private buffer of *interleave-page size*. When its buffer fills up, a thread empties it into the single output stream as a unit. The resulting interleaved data is thus a series of interleave-page–size units, each potentially from a different disk.

Interleaving plays havoc with segment routing. Consider what happens when interleaved data is received for deduplication. As an example, we show in Figure 2 two different interleavings of six underlying database streams and how they might be segmented. The different colors/shapes correspond to six different underlying streams of data. Each colored shape is a chunk and each continuous sequence of chunks of the same color/shape is an interleave page. For illustration purposes, we show dozens rather than hundreds of chunks per segment and only the first two segments per backup.

Because of the nondeterminism involved, even if the underlying streams have not changed (e.g., the database has not really changed much), very different interleavings and hence grouping of pages into segments can result. In our example, on the first day the red/diamond and purple/square pages were in different segments, whereas on the second day they were in the same segment. It is this inconsistency that causes problems. If on the first day, segment 1 and segment 2 are sent to different nodes because they are new data and hence there is no good bid, then on the second day there is no way to route segment

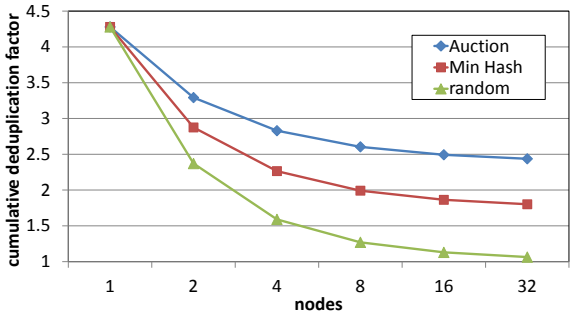

Figure 3: **Deduplication results for interleaved data** (Customer data set).

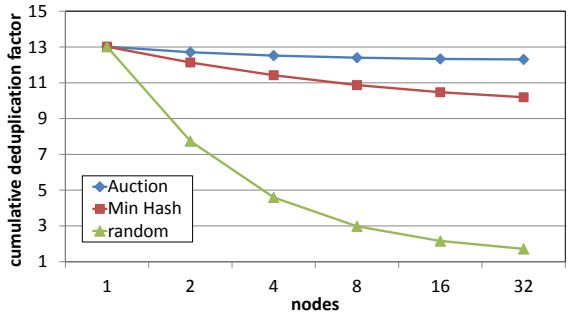

Figure 4: **Deduplication results for non-interleaved data** (Workgroup data set).

3 without duplicating either its square or diamond data.

This problem is not just theoretical: Figure 3 shows how in practice deduplication drops off rapidly in the presence of interleaved data as the number of nodes increase. This is true for both Min Hash and Auction. Contrast this with their behavior of non-interleaved data as shown by Figure 4, where—especially for Auction—there is little drop-off in deduplication. Results of randomly routing each segment are shown for comparison; experiment details may be found in Section 5.

In an ideal world, we would simply de-interleave the data and segment each underlying stream separately. Unfortunately, however, this would require us to be able to detect and correctly parse the format of the incoming data, which is dependent on the nature of the data (databases have their own proprietary storage formats) and the backup agent (backup agents put extra formatting information in the data stream). This would be a maintenance nightmare for real-world commercial systems so we strongly prefer an alternative solution that does not require correctly parsing stream formats.

In Figure 2 we showed an interleave-page size about one third of our segment size. Although the interleaving problem would be less severe if the interleave-page size was orders of magnitude larger than our segment size, we cannot make our segment size much smaller and

still maintain locality. Attempting to make the interleave-page size much larger runs into the conservatism of administrators, who are understandably reluctant to muck with hundreds of time-tested backup scripts. Accordingly, a system that can handle interleaving data with any interleave-page size will be much easier to sell and deploy.

4 Our Approach

We take advantage of a property of interleaved backup streams that we have discovered, which we call *extended locality*: if data has not changed much from one backup to another then two interleave pages of data that are close to each other in the first backup stream are highly likely to be close to each other in the second backup stream. How close is close depends on the system producing the streams and the overall length of the streams, but seems to be in the small number of GB's range in practice.

This follows from the fact that if the hardware and software have not changed between the two backups then the average relative speeds of reading the streams being interleaved are the same. This means that when the data does not change too greatly, each page ends up in roughly the same spot in the interleaved backup. How much its location varies depends on the read variances and the square root of the backup length,[1] which are limited in practice. Because pages tend to end up in the same locations, close pages tend to remain close between backups.

We modify Auction to take advantage of extended locality as follows: when there is no useful bid from any of the back-end nodes—*i. e.*, when the data is new—we try to send large swaths (GBs) of the incoming stream to the same node. Thus, the first backup of new data gets placed in large swaths. Now consider what happens during the second and later backups of that data. Each incoming segment will typically be composed of at least two partial interleave pages. Clearly these pages are close to each other in the current backup's stream, so by extended locality their distance in the first backup stream is not large. Thus, there is a good chance that they were routed during the first backup as part of the same swath and thus to the same back-end node, causing the auction process to find that back-end node and send the segment to it, successfully deduplicating those pages. We call our routing algorithm the Sticky-Auction routing algorithm.

4.1 The Sticky-Auction algorithm

The Sticky-Auction algorithm takes a global parameter, the *sticky threshold*, which determines the size of our swaths. For each incoming stream, its front-end node maintains two values: the current sticky node, n, and a current sticky counter, c. When the stream begins, c is initialized to 0 and n is set to the back-end node with the smallest load (defined per Section 2.2). In the event of a tie, one of the least-loaded nodes is chosen at random.

When a segment needs to be routed, the broadcast and bidding are done the same way as Auction. If at least one adequate bid is received (e.g., one exceeding a minimum bid threshold), the segment is sent to the node that Auction would have sent it to—one of the winning bidders.

If no adequate bid is received, the segment is sent to the current sticky node for this stream, n, and the sticky counter for this stream, c, is increased by the size of the segment. If c exceeds the sticky threshold, a new sticky node is chosen and assigned to n, and c is reset to 0. A new sticky node is chosen using the same procedure as was used to initialize n.

Higher values of sticky threshold handle more unbalanced interleaving processes but produce less overall system balance. Experimentally, we find values in the tens of gigabytes work well (see Section 5.5).

5 Results

To demonstrate the efficacy of our approach we simulate Sticky-Auction and measure deduplication and load-balancing using several data sets.

5.1 Data sets

We test against four primary data sets, two of which contain interleaved data, as well as a composite data set, *Combined*, which alternates backups from the four data sets simulating multiple stream ingest (see Section 5.4). Each backup in these data sets was chunked into either 4 KB (Workgroup, Homer) or 3850 byte variable-size chunks using the TTTD chunking algorithm [6].

Both of the interleaved data sets are a series of backups of a (different) Microsoft SQL Server database done using the SQL BACKUP command. The *Customer* data set contains nine backups, each 300 GB, of a database belonging to HP storage customer. This trace was originally provided by the customer using their usual backup parameters; we do not know the exact parameters used or the details of the database, but inspection of the trace shows the interleaving-page size is 1 MB.

The *Clickstream* data set consists of 6 backups of anonymized clickstream data provided by Nielsen. The first backup here holds the tables for January 2008. Before each succeeding backup we added the tables for another month. The final backup thus contains data from January 2008 through June 2008. Each month's tables occupy roughly 130 GB of space. The backups were

[1] The relative motion of the underlying streams is a random walk, with expected divergence $O(\sqrt{n})$.

done using the default settings, which means that the MAXTRANSFERSIZE parameter—the interleave-page size—is 1 MB. The database was set up using 8 mount points so 8 underlying streams were interleaved to produce each backup.

The first non-interleaved data set, *Workgroup*, is created from a semi-regular series of backups of the desktop PCs of a group of 20 engineers taken using uncompressed tar over a period of four months. These backups comprise 3.8 TB of data. We generate the Workgroup data set from these backups by grouping backups taken on the same day into a single "system-wide" backup, resulting in 91 backups with mean size 42 GB. We believe this data set is representative of a small corporate workgroup being backed up via tar directly to a NAS interface.

The second non-interleaved data set, *Homer*, is intended, by contrast, to be representative of a small or medium business server backed up to virtual tape. It contains two weeks (3 fulls, 12 incrementals) of Oracle & Exchange data backed up via Symantec's NetBackup to virtual tape with the fulls averaging 169 GB. The Exchange data was synthetic data generated by the Microsoft Exchange Server 2003 Load Simulator (LoadSim) tool [8], while the Oracle data was created by inserting rows from a real 1+ TB Oracle database belonging to a compliance test group combined with a small number of random deletes and updates.

5.2 Simulator

The simulator is a 9,000 line C++ program capable of simulating several modes and styles of deduplication. Here, it is set to simulate a multi-node system a la Figure 1 where each back-end node locally performs so-called perfect deduplication where no chunk is ever duplicated. We use variable-size segments with mean size 1 MB (same as Doug *et al.* [5]).

When bidding, we use 1:8 sampling also per Doug *et al.*, broadcasting only hashes with three particular bits zero. Using 1:1 sampling did not produce better results. A bid is the number of sampled chunks found at the given back end (*i. e.*, no approximation is done, which somewhat improves results) and ties are broken as described in Section 2.2. Bids below 2 are considered insufficient and ignored. When using Sticky Auction, we use a sticky threshold of 64 GB unless we say otherwise.

Though the simulator produces extensive statistics, here, we are concerned with the system-wide *(estimated) cumulative deduplication factor* (total raw/total deduplicated = sum of every non-deleted input chunk's length / sum of every currently-stored chunk copy's length) and a measure of load-balancing, *max/mean node size* = the maximum node size over the mean node size where a node's size is measured as the total deduplicated

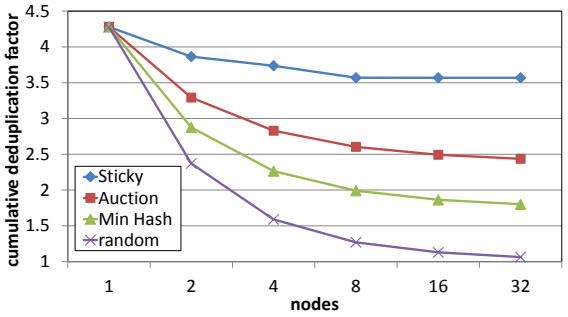

Figure 5: **Deduplication results for interleaved data including sticky routing** (Customer data set).

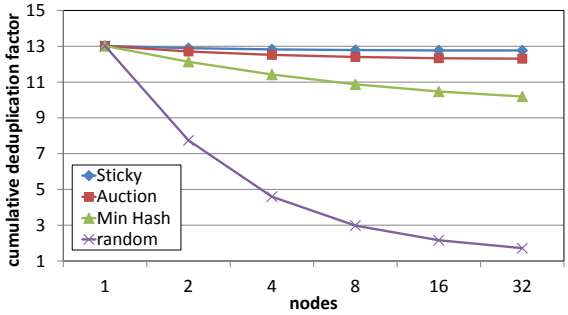

Figure 6: **Deduplication results for non-interleaved data including sticky routing** (Workgroup data set).

stored on that node. The estimated cumulative deduplication factor does not take into account metadata overhead (causing it to overestimate) or local compression of chunks (causing it to underestimate total compaction). We ignore local compression for this paper, measuring segment sizes and the sticky threshold in terms of uncompressed chunk lengths. Except where we say otherwise, displayed data is for the last backup of the given data set; cumulative thus here refers to all the backups of the data set.

5.3 Deduplication

Figure 5 updates Figure 3, showing Sticky-Auction routing's superior deduplication performance on interleaved data. Figure 6 shows that Sticky Auction's deduplication quality for non-interleaved data is somewhat better than Auction, which in turn is noticeably better than Min Hash. These results are typical, with similar results seen for the other data sets (not shown).

5.4 Load balancing

The typical use case for multi-node deduplication systems is where hundreds of streams are being ingested simultaneously. Sticky Auction has worse load balancing than Min Hash and Auction on individual streams be-

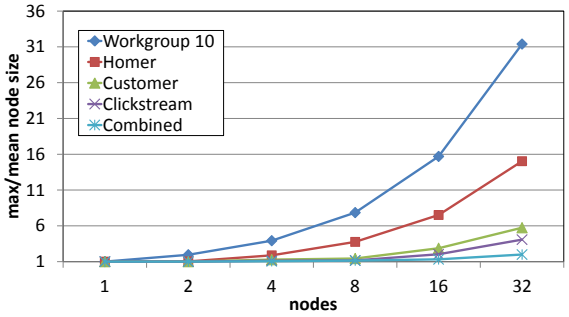

Figure 7: **Load balancing with multiple streams for Sticky Auction.** Clickstream data set derived from Nielsen data.

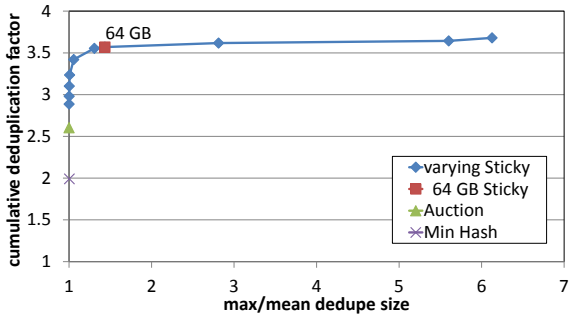

Figure 8: **Effects of varying the sticky threshold using 8 nodes** (Customer data set). Sticky threshold for varying Sticky is varied from 1 GB to 512 GB in powers of 2 from left to right, with 64 GB called out.

cause it switches nodes less often. But when it processes multiple independent streams, the overall load balancing significantly improves because unrelated streams are routed independently.

Our simulator does not directly support simultaneous ingestion of multiple streams, but we can simulate this by interleaving the daily backups from different data sets in a round-robin manner. We do just this with the Combined data set, which alternates the first 10 daily backups from the other four data sets when available.

Figure 7 shows the effect of combining streams. The first four results are single stream results, showing the load balancing of the system after only the first 10 days of that data set has been ingested. Workgroup is about as unbalanced as it is possible to get, with essentially all data located on one node. The other data sets and Workgroup over time—it falls at 32 nodes from 31.4 after backup 10 to 5.2—exhibit better load balancing.

As can be seen, combining the data sets significantly improves load-balancing: 2.0 for 32 nodes at the end of the Combined dataset. For perspective, at 32 nodes Min Hash is at 5.0 for Homer and averages 1.06 for the other 3 uncombined data sets. Auction is at 2.0 for Homer and averages 1.03 for the other 3. While we show the results of combining four streams here, with larger numbers of streams load balancing will only get better.

We expect that incorporating some of the suggestions from Dong *et al.* [5] for improving Auction's load balancing would improve Sticky Auction's numbers here. As expected, Sticky Auction deduplicates better than the other algorithms on Combined because it contains interleaved data: on this data set, Sticky Auction's cumulative deduplication factor at 32 nodes is 3.1 while Min Hash's is 1.7 and Auction's is 2.1.

5.5 The sticky threshold

Changing the sticky threshold trades off load-balancing against deduplication. Figure 8 shows this trade-off for Customer using 8 nodes. A similar trade-off is seen for the other data sets and node numbers: there is a sweet spot around our default of 64 GB where increasing the sticky threshold will show little improvement in deduplication while impairing load balancing.

6 Conclusions

We have demonstrated that the existing routing algorithms, Min Hash and Auction, have a major flaw: they perform poorly on interleaved data, which is of commercial importance because most database backups are interleaved. Our new routing algorithm, Sticky-Auction routing, is similar to Auction but does not suffer from this flaw. Although it has somewhat worse load-balancing on individual streams, when a large number of independent streams are ingested simultaneously—the typical use case for this type of system—the load balancing is completely acceptable. Even on non-interleaved data its deduplication performance is somewhat better than the existing methods, and it does not require understanding stream formats or require the interleaved-page size to lie in a particular range. We believe this makes it a promising replacement for the existing algorithms.

References

[1] SQLBackupRestore: Backup reads and writes. http://www.sqlbackuprestore.com/backupreadsandwrites.htm. Viewed September 24, 2012.

[2] Recovery manager (RMAN) performance tuning practices, an Oracle white paper. http://www.oracle.com/technetwork/database/availability/rman-perf-tuning-bp-452204.pdf, June 2011. Viewed September 24, 2012.

[3] BHAGWAT, D., ESHGHI, K., LONG, D. D. E., AND LILLIBRIDGE, M. Extreme binning: Scalable, parallel deduplication for chunk-based file backup. In *Proceedings of the 17th IEEE International Symposium on Modeling, Analysis, and Simulation*

of Computer and Telecommunication Systems (MASCOTS 2009) (Sept. 2009), pp. 1–9.

[4] BRODER, A. Z. On the resemblance and containment of documents. In *SEQUENCES '97: Proceedings of the Compression and Complexity of Sequences 1997* (Washington, DC, USA, 1997), IEEE Computer Society, pp. 21–29.

[5] DONG, W., DOUGLIS, F., LI, K., PATTERSON, H., REDDY, S., AND SHILANE, P. Tradeoffs in scalable data routing for deduplication clusters. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)* (San Jose, CA, USA, Feb. 2011), USENIX Association, pp. 15–30.

[6] ESHGHI, K. A framework for analyzing and improving content-based chunking algorithms. Tech. Rep. HPL-2005-30(R.1), Hewlett Packard Laboratories, Palo Alto, 2005.

[7] LILLIBRIDGE, M., ESHGHI, K., BHAGWAT, D., DEOLALIKAR, V., TREZISE, G., AND CAMPBELL, P. Sparse Indexing: Large scale, inline deduplication using sampling and locality. In *Proceedings of the Eighth USENIX Conference on File and Storage Technologies (FAST '09)* (Feb. 2009), pp. 111–123.

[8] Microsoft exchange server 2003 load simulator. Download available at http://www.microsoft.com/downloads, February 2006.

[9] PETERS, M. Implementing the Right High Availability and Disaster Recovery Plan of Your Business. *The Enterprise Strategy Group* (Aug. 2010).

[10] ZHU, B., LI, K., AND PATTERSON, H. 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)* (San Jose, CA, USA, February 2008), USENIX Association, pp. 269–282.