MAD2: A Scalable High-Throughput Exact Deduplication Approach for Network Backup Services

Jiansheng Wei[†], Hong Jiang[‡], Ke Zhou[†]⊠, Dan Feng[†]

[†]School of Computer, Huazhong University of Science and Technology, Wuhan, China
Wuhan National Laboratory for Optoelectronics, Wuhan, China

[‡]Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, USA

jianshengwei@gmail.com, jiang@cse.unl.edu, k.zhou@hust.edu.cn, dfeng@hust.edu.cn

***Abstract*—Deduplication has been widely used in disk-based secondary storage systems to improve space efficiency. However, there are two challenges facing scalable high-throughput deduplication storage. The first is the *duplicate-lookup disk bottleneck* due to the large size of data index that usually exceeds the available RAM space, which limits the deduplication throughput. The second is the *storage node island effect* resulting from duplicate data among multiple storage nodes that are difficult to eliminate. Existing approaches fail to completely eliminate the duplicates while simultaneously addressing the challenges.**

This paper proposes MAD2, a scalable high-throughput exact deduplication approach for network backup services. MAD2 eliminates duplicate data both at the file level and at the chunk level by employing four techniques to accelerate the deduplication process and evenly distribute data. First, *M*AD2 organizes fingerprints into a Hash Bucket *M*atrix (HBM), whose rows can be used to preserve the data locality in backups. Second, M*A*D2 uses Bloom Filter *A*rray (BFA) as a quick index to quickly identify non-duplicate incoming data objects or indicate where to find a possible duplicate. Third, *D*ual Cache is integrated in MA*D*2 to effectively capture and exploit data locality. Finally, MAD*2* employs a *D*HT-based Load-Balance technique to evenly distribute data objects among multiple storage nodes in their backup sequences to further enhance performance with a well-balanced load.

We evaluate our MAD2 approach on the backend storage of B-Cloud, a research-oriented distributed system that provides network backup services. Experimental results show that MAD2 significantly outperforms the state-of-the-art approximate deduplication approaches in terms of deduplication efficiency, supporting a deduplication throughput of at least 100MB/s for each storage component.

I. Introduction

Although backup storage systems have been widely deployed in enterprises, there remains a great need for providing network backup services to individuals and organizations for whom maintaining their own backup systems is either too expensive or cost ineffective. Existing network backup services can be classified into two categories, centralized storage provided by storage service providers (SSPs) and decentralized peer-to-peer schemes based on peer-cooperation over distributed network. The former allows user to trade money for reliable backup and provides better quality-of-service (QoS), while the latter allows user to trade local resources for remote storage capacity. Because forming a cooperative storage network that is large enough to run security policy is difficult in practice [1], Internet backup sites operated by SSPs remain the main providers of network backup services.

One of the main challenges facing centralized backup services is scalability. To reduce the total cost of ownership (TCO), which is one of the key factors affecting a SSP's competitiveness, the system must be both scalable and cost-effective.

The deduplication technology has been widely applied in disk-based secondary storage systems to improve cost-effectiveness via space efficiency [2-4]. However, there are two technical challenges that limit its application. The first is the *duplicate-lookup disk bottleneck*. Traditional approaches keep a full index in RAM to determine if an incoming data object is a duplicate. As data volume grows, the index can become too large for RAM to hold in its entirety, forcing the deduplicate process to lookup fingerprints in an on-disk index and degrading the system performance. The second challenge, which we refer to as the *storage node island effect*, stems from the fact that most early approaches eliminate duplicates within individual storage nodes but not among multiple servers.

It is clearly important for the network backup services overcome these challenges as they directly impact their scalability and cost-effectiveness.

Two recent studies, DDFS [5] and Sparse Indexing [6], proposed novel schemes to effectively eliminate the duplicate-lookup disk bottleneck by exploiting chunk localities in disk-to-disk (D2D) backup streams. Their approaches are based on one basic observation that chunks tend to reappear in the same or very similar sequences in backup streams. They store the chunks in a locality-preserving manner and then exploit the locality to accelerate duplicate detection. However, none of these approaches is targeted at scalable storage systems such as network backup services where there are a potentially large number of storage server nodes.

HYDRAstor [4] utilizes distributed hash table (DHT) to distribute data blocks among multiple storage nodes according to their hash-keys. Global deduplication can be achieved by eliminating duplicate blocks inside each node. HYDRAstor attempts to avoid the duplicate-lookup disk bottleneck by adopting a block size of 64KB on average, among other constraints, to keep all the metadata in memory.

Extreme Binning [7] exploits file similarity to eliminate duplicates in non-traditional backup workloads that are composed of files with little or no locality. Similar files sharing the same signature, called the *representative fingerprint*, are compressed into a compact structure called *bin* that maintains only non-duplicate chunks. Representative fingerprints of all the bins are organized into primary index that is sufficiently small to be kept in RAM, thus requiring only one disk access for each incoming file to locate its corresponding bin to eliminate duplicates. Further, Extreme Binning distributes bins among multiple storage nodes according to their representative fingerprints to enable scalable parallel deduplication. However, it allows duplicates to exist in different bins, which results in *approximate deduplication* as opposed to *exact deduplication* that completely eliminates duplicates at all levels.

To overcome the shortcomings of the above state-of-the-art deduplication approaches, we propose in this paper MAD2, a new scalable high-throughput exact deduplication approach aimed at backend storage system of network backup service. MAD2 completely eliminates duplicate data both at the file level and at the chunk level. Specifically, it employs four key techniques to accelerate the deduplication process and evenly distribute data. First, *M*AD2 organizes fingerprints into Hash Bucket *M*atrix (HBM), whose rows can be used to preserve the data locality in backups. Second, M*A*D2 uses Bloom Filter *A*rray (BFA) as a quick index to quickly identify a non-duplicate incoming data object or indicate where to find a possible duplicate. Third, MA*D*2 integrates *D*ual Cache, one of which is implemented as a directly-mapped cache to buffer incoming non-duplicate fingerprints and capture locality while the other is implemented as a set-associative cache to reduce disk accesses by exploiting locality and maintaining a high hit rate for duplicate fingerprints. Finally, MAD*2* employs a *D*HT-based Load-Balance technique to evenly distribute data objects among multiple storage nodes in their backup sequences to further enhance performance with a well-balanced load.

The first three techniques are effective in avoiding the duplicate-lookup disk bottleneck, while the last technique eliminates the storage node island effect and enables scalable parallel deduplication. Most importantly, MAD2 eliminates all duplicates while simultaneously addressing the key deduplication challenges.

We evaluate our MAD2 approach on the backend storage of B-Cloud, a research-oriented distributed system that provides network backup services. Deduplication efficiency results show that MAD2 achieves compression ratios of 16.73 and 18.52 respectively for our two backup data sets with exploitable locality, which significantly outperforms the state-of-the-art Extreme Binning approach with the corresponding ratios of 14.10 and 9.71. Storage load is shown to be perfectly balanced while using 4 storage components. Performance evaluation shows that MAD2 is about 3.7 times faster than Extreme Binning in identifying duplicate fingerprints and supports a deduplication throughput of at least 100MB/s for each storage component. Further, we observed the existence of extremely hot *zero-chunks* that may be widely shared even among dissimilar files. This observation is very helpful in further improving our approach.

The remainder of this paper is organized as follows. In the next section, we provide the necessary background information to further motivate our MAD2 research. The MAD2 architecture and its detailed design are presented in Section 3. Section 4 evaluates MAD2 by comparing and analyzing the extensive experimental results obtained from our MAD2 prototype implementation. Section 5 reviews the research in the literature that is most relevant to MAD2 and Section 6 concludes the paper.

II. Background and Motivation

A. *Duplicate Detection Methods*

Deduplication is one of the main techniques to eliminate redundancy in datasets. Different from sequential compression and delta encoding, it usually works at the KB or larger granularity.

Traditional backup software usually detects possible duplicate transmission by examining the file system metadata. For example, incremental backup omits the files that reside in the same path and with a timestamp earlier than the last backup. Since the timestamp of a file may be modified by virus or special user command such as ‘touch’ and the clock of the user host can also be at risk of being illegally tampered, simply identifying duplicate files by the associated file system metadata is far from being reliable.

The emergence of highly reliable hash algorithms such as MD5 and SHA-1 has enabled many recent approaches to adopt content-based duplicate detection, which is more efficient than byte-to-byte comparison and more reliable than metadata-based duplicate identification. For a given data object, a hash algorithm is used in content-based duplicate detection to generate a unique identifier, also referred to as a fingerprint, to identify duplicate data objects.

In general, content-based deduplication can be carried out at one of the three levels of granularity, namely, whole files, fixed-size blocks, or variable-sized chunks generated by a content-defined chunking algorithm. Previous research shows that variable-sized chunk-level deduplication is more space efficient than the other two methods [8]. It is more sensitive to duplicates than the whole-file hashing approach and can detect duplicate data chunks among similar files. On the other hand, it is far more effective than the fixed-size blocking approach in solving the block-shifting problem [9].

B. *Duplicate Lookup Acceleration Methods*

Previous studies [8, 10] have shown that the storage efficiency of variable-sized chunk-level deduplication is highly dependent on the average chunk size, and smaller chunks usually detect more duplicate information. However, smaller average chunk size also means that more chunks will be generated for a given data set, resulting in a chunk index that may be too large for RAM to hold and leading to the duplicate-lookup disk bottleneck. For example, an average chunk size of 4KB implies 2.25×2^{30} unique chunks in a 10TB deduplicated dataset. Considering that each fingerprint consumes 40 bytes, an approximately 100GB memory will be needed to hold the whole index, which is prohibitively expensive for today's systems and thus forces the deduplication process to frequently access an on-disk index. In this case, fingerprint lookup acceleration becomes critically important.

Currently, there are generally two approaches to improving the fingerprint lookup efficiency. The first approach exploits data locality, which has been used in DDFS and Sparse Indexing. DDFS [5] exploits chunk locality in D2D backup streams. It utilizes Stream-Informed Segment Layout (SISL) to create spatial localities for both chunk fingerprints and chunk contents, and then employs Locality Preserved Caching (LPC) to exploit locality and accelerate duplicate chunk detection. DDFS also uses Bloom Filter to quickly identify incoming non-duplicate chunks.

Sparse Indexing [6] divides a data stream into 4KB average-sized chunks and then partitions chunk sequences into 10MB average-sized segments. For each segment, a few representative fingerprints are sampled and inserted into an in-memory structure called *sparse index*, which is used to estimate similarity between segments. Once a new segment arrives, it is sampled and deduplicated against several most similar existing segments. Although Sparse Indexing consumes smaller RAM space than DDFS, its duplication efficiency can be affected by the sample rate and a few other factors, which makes it an approximate deduplication approach.

Both DDFS and Sparse Indexing are targeted at D2D backup workloads, where scalability is not as important a concern as in a network backup-service environment.

The second approach exploits file similarity, which has been used in Extreme Binning [7], a deduplication approach targeting at non-traditional backup workloads that are composed of individual files with little or no locality. For each file, Extreme Binning chooses the smallest chunk hash as the representative fingerprint. Files sharing the same representative fingerprint are grouped into a bin, which is the basic scope of chunk level deduplication. Representative fingerprints of all the bins are organized into a primary index that is sufficiently small to remain in RAM, so that only one disk access to its corresponding bin is needed for each incoming file.

Since Extreme Binning samples only one fingerprint for each file, the probability of similar files being grouped into the same bin is highly dependent on their similarity degree. According to Broder's Theorem [11], the probability that different files share the same representative fingerprint will decrease as the number of files grows. For example, consider two chunk sets S_1 and S_2 generated from two different files respectively. Let $H(S_n)$ denote the corresponding fingerprint set of S_n and $\min(H(S_n))$ denote the smallest element of $H(S_n)$. Then:

$$\Pr[\min(H(S_1)) = \min(H(S_2))] = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Now, introducing a new chunk set S_3. We have:

$$\Pr[\min(H(S_1)) = \min(H(S_2)) = \min(H(S_3))]$$
$$= \frac{|S_1 \cap S_2 \cap S_3|}{|S_1 \cup S_2 \cup S_3|} \le \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

As a result, it can be speculated that the number of bins will increase rapidly as the number of files grows, resulting in more duplicate chunks to remain among bins.

In the final analysis, the key in fingerprint lookup acceleration lies in the fast membership determination of incoming data objects and the fast identification of the potential existing duplicates.

C. *Scalable Deduplication*

Existing approaches to scaling up the deduplication storage are generally based on the use of distributed hash table.

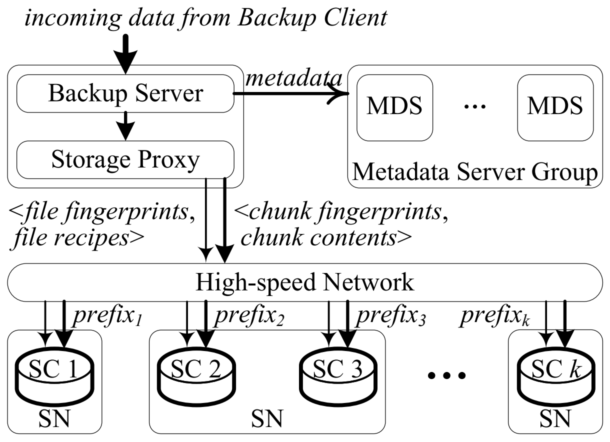
HYDRAstor [4] uses a modified version of the fixed prefix network (FPN) DHT to distribute data blocks according to their hash-keys among logical *supernodes*. Data blocks assigned to different supernodes have different hash prefixes and can be deduplicated in parallel. Different from HYDRAstor, Extreme Binning [7] distributes bins among storage nodes according to their representative fingerprints to enable scalable parallel deduplication. Each storage node is only responsible for files with specified representative fingerprints and files belonging to different bins can be deduplicated in parallel. However, both HYDRAstor and Extreme Binning are considered approximate deduplication schemes as they fail to completely eliminate duplicates.

In summary, the key for scalable deduplication lies in finding an effective way to partition data into dissimilar or less similar groups. Moreover, the redundancy degree between less similar groups must be acceptable and controllable.

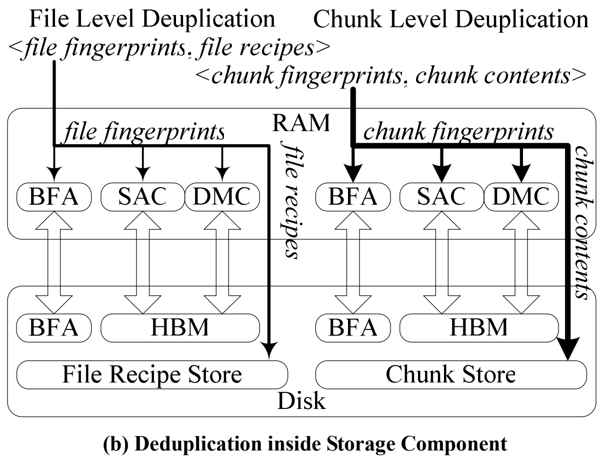
III. The MAD2 Architecture and Design

This section presents the architecture and design of MAD2, our approach to providing scalable high-throughput exact deduplication in the backend storage of network backup services.

To provide the necessary context for presenting MAD2, we first describe the architecture of its underlying storage system, the backend storage of network backup services. While MAD2 is designed to be applicable to general network backup services, we use the B-Cloud system as an example to illustrate the key architectural features of a typical backend storage of network backup services since our MAD2 is prototyped on B-Cloud. B-Cloud is a research-oriented distributed system that provides network backup services for user files and other binary data.



(a) Data Distribution among Multiple Storage Nodes



(b) Deduplication inside Storage Component

Figure 1: The Application of MAD2 in B-Cloud

As shown in Figure 1-(a), the B-Cloud front end consists of *backup servers* (BSs), *storage proxies* (SPs) and *metadata servers* (MDSs). BS and SP are independent software components, and they usually run in pairs on the same physical node to avoid unnecessary network traffic. When the user initiates a backup job, one BS is chosen to control the job status and receive incoming files. BS splits metadata from files and sends them to the responsible MDS along with the job information. On the other hand, SP receives file contents from BS and divides them into content-defined chunks using the Rabin fingerprinting algorithm [12]. Meanwhile, both file fingerprints and chunk fingerprints are calculated, and *file recipes* are generated for file reconstruction purposes. The chunking and fingerprinting processes can also be done by the *backup client* (BC), depending on the hardware environment of the user host and whether there is a bandwidth-saving requirement.

The backend of B-Cloud is built of clustered storage nodes. Based on the availability of physical resources, each storage node (SN) can run a single or multiple *storage components* (SCs) to deduplicate and maintain distributed backup data.

MAD2 employs DHT-based load balancing to distribute file recipes and chunk contents among multiple SCs in their backup sequences. As shown in Figure 1-(b), both file fingerprints and chunk fingerprints are organized into locality-preserved Hash Bucket Matrixes (HBMs) inside each SC. MAD2 utilizes Bloom Filter Array (BFA) to quickly identify incoming non-duplicate fingerprints and indicate in which row of HBM to find a possible duplicate. Because BFA has an unavoidable false positive probability and possible duplicates must be confirmed by checking HBM, MAD2 integrates Dual Cache to capture and exploit the fingerprint locality and further accelerate the duplicate confirmation process. In the remainder of this section, the design of MAD2 will be detailed by the descriptions of its main functional components.

A. Locality-Preserved Hash Bucket Matrix

As previous research shows, exploiting data locality is an effective approach to accelerating duplicate detection process, and there exists substantial exploitable locality in most backup workloads. For example, backup clients usually transfer target files in the same sequence in multiple backup jobs, and identical chunks tend to appear in approximately the same sequence between similar backup files. DDFS and Sparse Indexing preserve locality in a D2D backup environment by storing chunk fingerprints exactly in the order of their incoming sequences. To achieve the goal of preserving fingerprint locality in a distributed environment, we develop a novel data structure called Hash Bucket Matrix (HBM) that is scalable and conducive to parallel deduplication.

Figure 2 shows the structure of HBM. In this structure, the fingerprint space is divided into n equal parts called *super buckets* that are each further divided into *buckets* of equal capacity. Buckets that belong to different super buckets but are in the same logical row are grouped into a *tanker*, the basic unit on which the fingerprint locality is preserved. During backup, all the incoming non-duplicate fingerprints will be appended into HBM. If any super bucket is full, a new tanker will be created to extend the capacity of HBM.

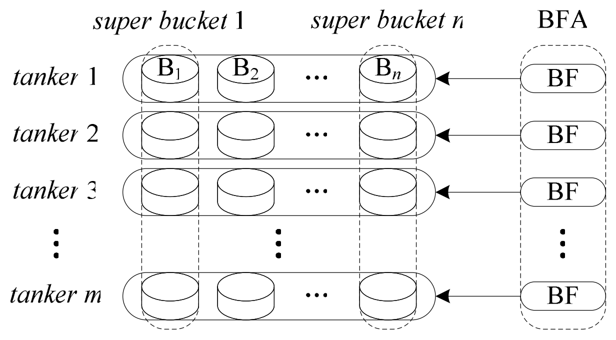


Figure 2: Structure of HBM and BFA

Now, we explain how HBM preserves fingerprint locality in backup jobs. Consider the fact that the fingerprint value of an incoming data object is totally random and could be assigned to any super bucket with the same probability, all the super buckets may hold approximately the same amount of fingerprints at any given time. This also implies that we can expect consecutive fingerprints belonging to the same backup job to have a high probability of being stored in the same tanker. If a group of files or a group of chunks tends to reappear together in backup jobs, their locality can be preserved by tankers of HBM.

B. Using Bloom Filter Array as Quick Index

While HBM can preserve fingerprint locality, it does not directly address the issue of the duplicate-lookup disk bottleneck. A fast index is needed to identify non-duplicate incoming fingerprints or determine in which tanker to find possible duplicates in an expeditious way. Bloom Filter (BF) [13] is a good tool for building such a fast index, due to its efficiency in recognizing unique fingerprints, and its controllable probability of identifying duplicates.

However, a number of problems will arise when recording the memberships of all the existing fingerprints using a single BF. First, if the potential number of fingerprints is underestimated, the false positive rate of BF will increase rapidly as the number of fingerprints exceeds the BF capacity, forcing a BF reconstruction. Second, a single BF is ineffective in locating possible duplicates. Third, every time a fingerprint is physically removed, resulting in the deletion of an element in BF, the whole BF must be rebuilt. While a Counting Bloom Filter presents a potential solution [14], its use will likely increase the RAM requirement and decrease the performance.

Instead, MAD2 employs a Bloom Filter Array (BFA), shown in Figure 2, where each tanker is associated with a Bloom Filter recording the membership information for the member fingerprints. All the Bloom Filters are isomorphic and share the same hash functions. Once a new fingerprint arrives, it can be quickly identified by a BFA query. If any member of the BFA returns a positive for the incoming fingerprint, there may be an existing duplicate in the corresponding tanker. And the fingerprint prefix can be used to determine the target bucket. Conversely, all negatives denote that the incoming fingerprint is definitively unique. If the HBM overflows, we only need to add a Bloom Filter along with a new tanker, which means the memory consumption will only increase linearly with the amount of data.

According to Broder's analysis [14], given Bloom Filter's bitwise m and total number of fingerprints n, the optimal number of hash functions k can be expressed as $k=(m/n)\times\ln 2$, and the minimal false positive rate f can be expressed as $f=(1/2)^{(m/n)\times\ln 2}$. If we bound false positive rate to ε, i.e., $f\leq\varepsilon$, then we can infer that the bitwise m should satisfy $m\geq n\times\log_2 e\times\log_2(1/\varepsilon)$. Consider the situation in Section 2.2, where 10TB data result in 2.5×2^{30} chunks with 4KB average size. The minimal space requirement of BFA can be calculated in bytes using $n\times\log_2 e\times\log_2(1/\varepsilon)\times(2.5\times2^{30}/n)\times(1/8)$. Detailed RAM consumption and memory saving methods will be discussed later in Section 4.6.

C. Dual Cache Mechanism

Since BFA has an unavoidable false positive probability, possible duplicates need to be confirmed by checking the on-disk HBM. *Dual Cache* is designed to improve disk access efficiency while locating duplicate fingerprints.

To describe the mechanism of Dual Cache, we first introduce the two possible states of a tanker, i.e., appendable state and reference-only state. An appendable tanker appends incoming non-duplicate fingerprints to the corresponding bucket and records the reference count for each duplicate fingerprint, while a reference-only tanker only maintains existing non-duplicate fingerprints. An appendable tanker will gracefully transition to the reference-only state when it is full. Before that, fingerprints inside each bucket will be sorted to enable binary search for future duplicate identification.

To capture the fingerprint locality in backup streams, MAD2 utilizes a *directly-mapped cache* (DMC, Figure 1-(b)) to hold several appendable tankers in RAM, as shown in Figure 3. All the incoming non-duplicate fingerprints will be directly appended to the bottom of the corresponding bucket. Once a tanker in DMC is full, it will be changed to the reference-only state and saved to the on-disk HBM. A new tanker will be created by DMC if there is no space left for incoming fingerprints.

However, our experiments indicate that the super buckets in HBM are not perfectly balanced in their actual occupations and this imbalance increases as the HBM capacity grows, which can decrease the locality-capturing capability of a tanker. To solve this problem, MAD2 introduces a periodic rebalancing policy. Specifically, if the current imbalance among super buckets exceeds a predefined threshold, all the tankers in DMC will be changed to the reference-only state and flushed to the on-disk HBM, meanwhile the imbalance will be reset to *zero* and a new appendable tanker will be created if necessary. Because of the periodic rebalancing policy, there may be a small number of empty cells in some reference-only tankers. Fortunately, the number of tankers with empty cells can be controlled by properly choosing the HBM rebalancing threshold, and it is easy to avoid the waste of disk space by compressing these empty cells.

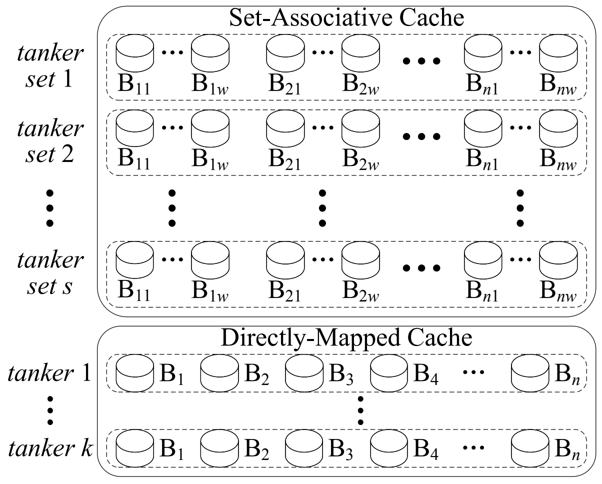


Figure 3: Dual Cache. The Directly-Mapped Cache is always mapped to the appendable tankers at the bottom of HBM, and the Set-Associative Cache is mapped to all the reference-only tankers inside HBM. Their mapping range will change accordingly as the HBM capacity changes.

To exploit the fingerprint locality in backup data, MAD2 uses a *set-associative cache* (SAC, Figure 1-(b)) to cache the buckets of reference-only tankers and maintain a high hit rate for duplicate fingerprints. As Figure 3 shows, each tanker set consists of *n* bucket sets that each further contains *w* buckets, where *n* is the number of buckets inside each tanker and *w* is the associativity (way count) of SAC. For a target bucket, MAD2 can determine the mapped bucket set according to its tanker ID and bucket ID.

MAD2 adopts two policies to improve the access efficiency of SAC. First, it takes a bucket as the basic I/O unit and executes LRU replacement policy inside each bucket set. This policy can effectively reduce the disk access cost caused by a BFA false positive, with only one small disk read required for the target bucket. Second, it executes a batch write-back policy to enhance the disk access locality. Since logical insert or delete operations can change the reference counts of the corresponding fingerprints, dirty buckets will be generated. Once a dirty bucket is going to be replaced, MAD2 flushes all the cached buckets belonging to the same tanker back to disk. Because of the locality-preserving capability of HBM, buckets in the same tanker usually have correlated access patterns and tend to be loaded or replaced in the same time window. By performing a batch write-back policy, both fingerprint locality and disk access locality are well exploited.

D. DHT-based Load Balancing

Benefitting from the randomness of SHA-1, DHT is a natural choice for partitioning data into dissimilar groups and distributing them among multiple storage components (SCs) with a well-balanced load.

In our approach, each SC is responsible for file recipes and chunks with the same specific fingerprint prefix, as shown in Figure 1. Because fingerprints with different prefixes are collision free, multiple SCs can deduplicate data in parallel and no duplicate files or chunks will remain among them. If each SC performs exact deduplication in its responsible hash sub-space, the entire backend storage can achieve global exact deduplication.

Moreover, both file recipes and chunk contents will be distributed in their backup sequences to preserve locality. Consider a sequence of fingerprints with two different prefixes (a_1, b_0, c_1, d_1, e_0, f_1, g_0). MAD2 divides them into two sub-sequences (a_1, c_1, d_1, f_1) and (b_0, e_0, g_0), and distributes each sub-sequence to one responsible SC. If the fingerprint sequence reappears in backup data, the locality of each sub-sequence will be captured and exploited.

E. Data Organization and Deletion Support

As shown in Figure 1-(b), there are four kinds of data maintained by SCs, file fingerprints along with file recipes and chunk fingerprints along with chunk contents. Both the file fingerprints and the chunk fingerprints are organized as on-disk HBMs that can be cached in RAM by Dual Caches. And each HBM is associated with a BFA that is kept in RAM in its entirety to accelerate non-duplicate identification and duplicate locating.

File recipes and chunk contents are generally stored on disk. In particular, all the chunk contents are kept in *chunk store*, which consists of *chunk tankers* corresponding to *tankers* in HBM. More specifically, inside each chunk tanker, chunks are grouped and packaged into *chunk containers* in a stream-locality-preserved manner to improve the disk access efficiency. Chunk containers can be further compressed by a variation of the Ziv-Lempel algorithm [15]. However, we do not have enough room in this paper to describe the detailed organization of chunk store.

Supporting data deletion is difficult but important for a deduplication storage targeting a network backup workload. Files or chunks may be referenced by multiple users after deduplication, and any associated user may request to delete his/her own logical copy.

MAD2 exposes only a file-level delete interface to SC clients and determines whether a file or chunk should be physically deleted by means of a special data structure called *counting fingerprint*. A counting fingerprint is structured as <*fingerprint*, *data length*, *reference count*>. *Fingerprint* is a 20-bytes SHA-1 hash of the target data object. Both *data length* and *reference count* are represented by 64-bit unsigned integers. A logical file delete operation will decrease the reference count of the corresponding file fingerprint, and a physical file delete operation decreases reference counts of all the chunk fingerprints belonging to that file. A file or a chunk will not be physically deleted until the associated reference count drops to *zero*.

Practically, a physical delete operation is executed in a batch mode. When there are too many unreferenced fingerprints in a tanker, the corresponding file recipes or chunk contents will be physically deleted and the associated Bloom Filter will be reconstructed. Adjacent tankers can be merged if they are sparse enough and all the involved Bloom Filters will be reconstructed. During the physical deletion period, all involved tankers must be changed to the read-only mode to maintain data consistency, suggesting that it is better to reclaim spaces in off-peak hours.

F. Workflow of the MAD2 Deduplication Approach

With the main functional components of MAD2 presented above, we now put them together to describe the overall workflow of the MAD2 deduplication process in the backend storage of B-Cloud.

SC (storage component) supports two inline deduplication modes, the exclusive mode and round-robin mode. The former targets at high-speed backup streams that can finish data transmission in short time windows, while the latter aims at low-speed backup streams that will be buffered by SP (storage proxy). The purpose of this is to avoid low-bandwidth concurrent write streams and make sure that the locality of each data stream can be captured and preserved. In general, deduplication can be performed concurrently among multiple SCs.

Specifically, the MAD2 deduplication process includes two phases. In the first phase, SP distributes file fingerprints along with file recipes to their responsible SCs to eliminate duplicate files. In particular, SC performs the following steps for each file:

- Use the BFA to quickly check if the incoming file fingerprint is unique.
- If the BFA indicates that a duplicate possibly exists in a certain tanker, then the fingerprint is handed to Dual Cache. According to the target tanker ID, SAC or DMC will be chosen to further lookup the potential duplicate.
- If a duplicate is found, then the corresponding reference count is increased and the associated file recipe will not be transferred.
- If the incoming file turns out to be unique, then the fingerprint is appended to the appendable tanker maintained by DMC and located at the bottom of HBM, and the file recipe is transferred and saved.

In the second phase, SP distributes chunk fingerprints along with chunk contents of non-duplicate files to multiple SCs to eliminate duplicate chunks. In particular, SC performs the following steps:

- Detect duplicate chunks in the same way as above.
- All identified non-duplicate chunk contents are written into chunk store sequentially with their fingerprints recorded.

Note that if a duplicate file is detected, then all the chunks belonging to that file can be directly skipped. Besides, SP can also work as a transparent proxy to enable bandwidth saving between backup clients and backup servers.

IV. Prototype Implementation and Evaluation

We evaluate MAD2 through a prototype of MAD2 running on a Windows environment that consists of 2 storage nodes that each in turn hosts 2 storage components. The hardware configuration includes a quad-core CPU running at 2 GHz, 16GB RAM, 2 gigabit network interface cards, and 16 1TB hard disks organized in a RAID5 system. The experimental front end of B-Cloud is composed of a single metadata server and two backup servers that is each coupled with a storage proxy.

Our data sets consist of files from two different groups of users. The first data set was collected from an engineering group consisting of 15 graduate students, which we refer to as the *Workgroup* set. Each student runs full or incremental backups independently using their desktop PCs and workstations in a span of 31 days. There are 12.1 million files that amount to a total of 6.0TB data in the *Workgroup* set.

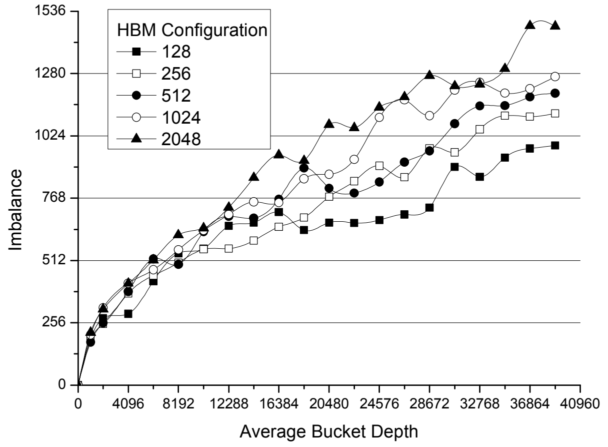
The second data set was collected from 26 users on a campus network, including personal website owners, small file transfer site managers and other individuals. Every user backs up their selected datasets in a span of 31 days independently, which is called the *Campus* set. There are 15.4 million files for a total size of 4.7TB in the *Campus* set.

A. Locality-Preserving Capability of HBM

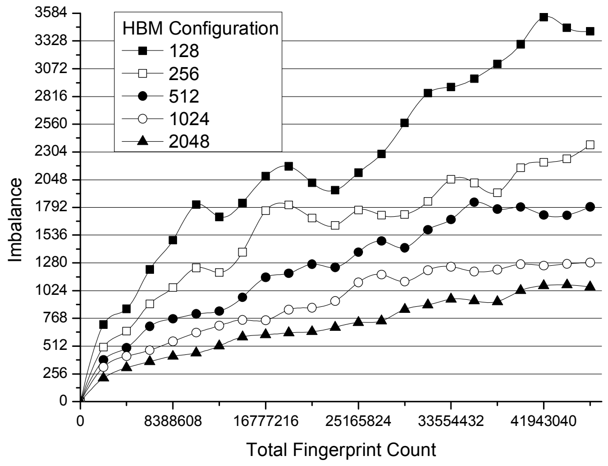
To evaluate the locality-preserving capability of HBM and its sensitivity to some key design parameters such as the number of super buckets (the number of columns in HBM) and the capacity of a physical bucket, we first measure the load-balancing capability of HBM. We define (super) bucket depth as the number of fingerprints contained in a (super) bucket, and load *imbalance* as the depth difference between the super bucket with the most fingerprints and the super bucket with the fewest fingerprints.

For the *Workgroup* set, by choosing an average chunk size of 4KB, with a minimal threshold of 1KB and a maximum threshold of 64KB, our chunking algorithm generated 83,733,597 unique chunks for a total size of 367.7GB. We use the distribution of the fingerprints of these chunks among the super buckets to measure the load imbalance.

By doubling the number of super buckets in HBM from 128 to 2048 and setting the physical bucket capacity as infinity, implying that each super bucket consists of only one bucket, we examine five different configurations of HBM (i.e., 128-, 256-, 512-, 1024-, and 2048-super-bucket HBM). As Figure 4-(a) shows, at the same average super bucket depth, a larger number of super buckets in HBM tends to induce greater imbalance. As the super bucket depth grows, the imbalance of a HBM with more super buckets increases faster than a HBM with fewer super buckets.



(a) Imbalance as a Function of Average Super Bucket Depth



(b) Imbalance as a Function of Total Fingerprint Count

Figure 4: Imbalance Measurement of HBM

On the other hand, a larger number of super buckets in HBM imply a smaller average super bucket depth for the same number of fingerprints. Figure 4-(b) compares the imbalances of five HBM configurations as a function of the number of fingerprints, counting from 2 million up to 44 million fingerprints. The HBM configuration with 2,048 super buckets is the least imbalanced with a maximum imbalance of 1,077 fingerprints, while the one with 128 super buckets is the most imbalanced with an imbalance of up to 3,550 fingerprints.

Additionally, we observe that all curves in Figure 4 show approximately logarithmic growth. Let *m* denote the average depth of super buckets, *n* the number of super buckets, and *k* an integer larger than *zero*. We can conclude from Figure 4-(a) the following relationships about imbalance:

$$\mathrm{Imb}(m,n) \le \mathrm{Imb}(m,kn) \qquad (1)$$

$$\mathrm{Imb}(km,n) \le k \cdot \mathrm{Imb}(m,n) \qquad (2)$$

And from Figure 4-(b) the following relationship:

$$\mathrm{Imb}(m,kn) \le \mathrm{Imb}(km,n) \qquad (3)$$

These relationships provide very useful insight into the configuration and design of HBM in general and the shaping of its tankers in particular.

According to Relationship (3), for a given capacity, the larger the number of super buckets is, the more effective it is for a tanker to balance load and capture fingerprint locality. Relationship (2) further indicates that it is beneficial to adopt a larger bucket depth for each tanker, for otherwise a sequence of fingerprints will be more likely to be dropped into neighbor tankers.

In our prototype implementation, we shape the tanker with 1,024 buckets plus 1,024 fingerprint cells inside each bucket. By using a 40-byte fingerprint structure, each tanker holds at most 2^{20} fingerprints for a total size of 40MB, and each bucket occupies only 40KB space. According to Figure 4-(a), we choose 1,024 as the rebalancing threshold, which means that HBM will be periodically rebalanced for approximately every 24 newly added tankers.

B. Extremely Hot Fingerprints

During the locality-preserving-capability evaluation of HBM, we also observed the existence of extremely hot *zero-chunks* that may be widely shared even among dissimilar files.

Specifically, we detected 84,876,504 duplicate chunks with the same content of 1,024-byte *zeros*. And we noticed that *zero* strings may exist in many files. By setting the minimum chunk size to 1KB, even dissimilar files share the same *zero-chunk*, which can disrupt the chunk locality and affect the efficiency of our cache mechanism.

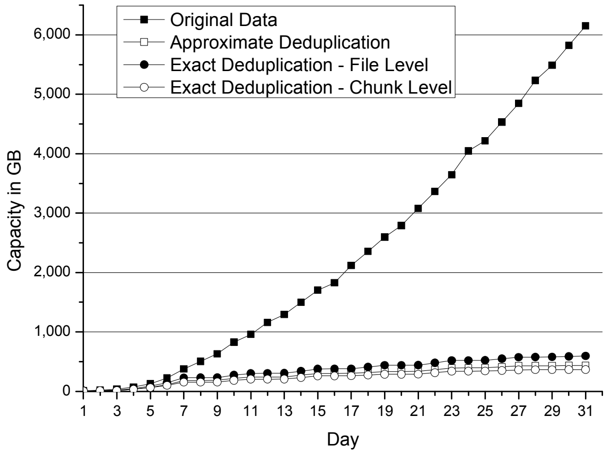
To further improve our approach, we pre-calculated the SHA-1 hash of 1KB *zero-chunk*, and define it as a built-in fingerprint. All incoming fingerprints matching the built-in fingerprint will be directly deduplicated, which will not incur any cache replacement operation.

C. Deduplication Efficiency

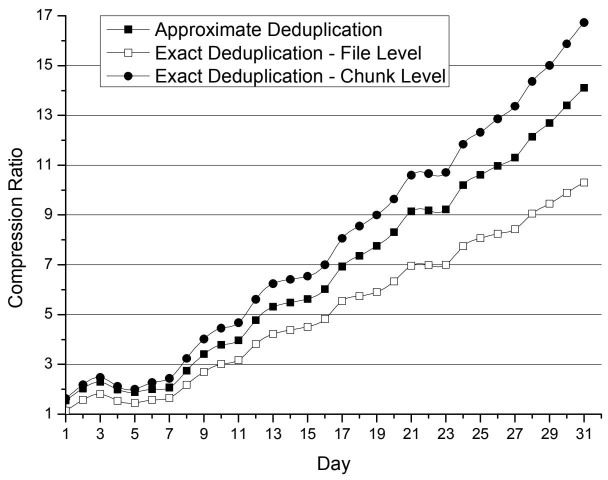
This subsection reports the deduplication efficiency of MAD2 for the two aforementioned data sets, *Workgroup* and *Campus*. For comparison, we have also implemented a simple version of Extreme Binning to represent *approximate deduplication*.

Figure 5-(a) shows the original cumulative capacities and deduplicated cumulative capacities over 31 days for the *Workgroup* set. During the beginning days, only a few students joined our experiment and backed up their data, so the data capacity grew slowly. As more users participated, the original data size grew steadily to 6,151.9GB by the last day. By using the MAD2 deduplication approach, there are logically 596.5GB data at the file level and physically 367.7GB data at the chunk level. On the other hand, Extreme Binning generates 436.3GB data, which lies in between the two data sizes of our approach.

The cumulative global compression ratios are shown in Figure 5-(b). At the end of the 31st day, the compression ratio of file-level exact deduplication reaches 10.31, while the compression ratio of chunk-level exact deduplication is much higher at 16.73. Extreme Binning also achieves very good compression ratio at 14.10, which is close to chunk-level exact deduplication. All of the three compression ratios represent similar trend during the 31-day period. Also, we notice that on some days (e.g., the 4th day and 5th day) the global compression ratio decreases slightly, which means that, during these days, most newly arrived files or chunks are non-duplicate, and the daily compression ratio is much lower than the usual.



(a) Logical/Physical Capacities for the *Workgroup* Set

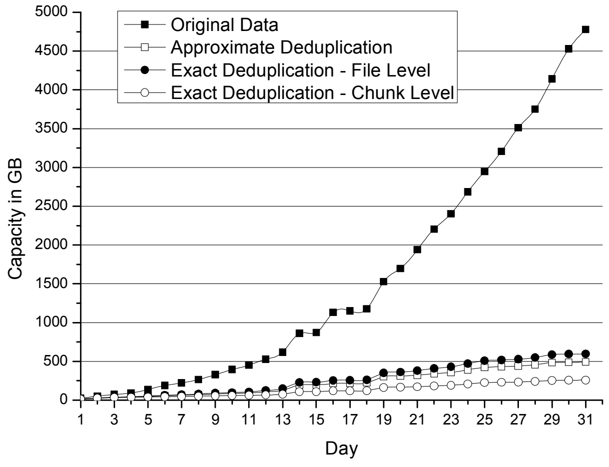


(b) Compression Ratios for the *Workgroup* Set

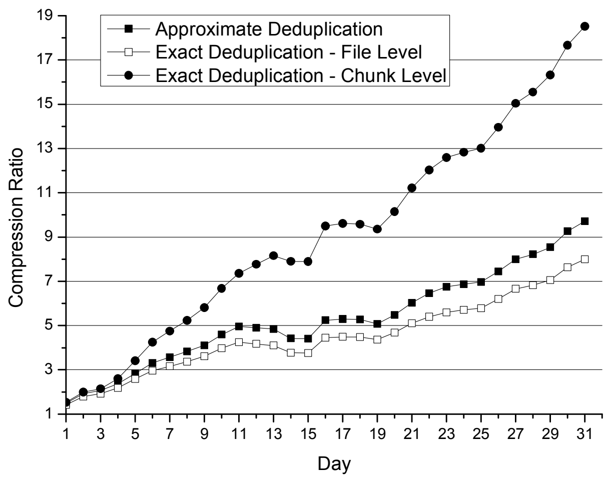
Figure 5: Deduplication Efficiency for the *Workgroup* Set

For the *Campus* set, since it takes more time for us to persuade the campus users to participate in our experiment, the original data capacity grew very slowly during the first few days. As Figure 6-(a) shows, the original cumulative data capacity did not reach 500GB until the 12th day. At the end of the 31st day, the original cumulative data amount reached 4,778.1GB. By using the MAD2 deduplication, there are 597.3GB logical data at the file level and 258.0GB physical data at the chunk level. The corresponding data size of Extreme Binning is 491.9GB, which is still between our two data sizes but much worse than the chunk-level exact deduplication.

Figure 6-(b) shows the cumulative global compression ratios over time for the *Campus* set. At the end of the 31st day, the compression ratio of the file-level exact deduplication reaches 8.00, while the compression ratio of the chunk-level exact deduplication is far better at 18.52. On the other hand, Extreme Binning achieves a compression ratio at 9.71 on the 31st day, which is slightly better than our file-level ratio but only at about half of our chunk-level ratio. This result also shows that there is an abundant amount of low-similarity real-world data for which Extreme Binning is much less efficient and effective in eliminating duplicates.



(a) Logical/Physical Capacities for the *Campus* Set



(b) Compression Ratios for the *Campus* Set

Figure 6: Deduplication Efficiency for the *Campus* Set

Similar to the results on the *Workgroup* set, the cumulative compression ratios for the *Campus* set also show an uneven rising trend, which is quite different from the results obtained from D2D backup [5]. We believe that this phenomenon reflects the important distinction between the internal D2D backup and the service oriented network backup. For network backup services, both backup policies and data compressibility may vary between users and over time. Further, we notice that most of duplicate information exists at the file level in our collected data sets, which suggests the whole file duplication may be one of the main causes of data redundancy.

D. Load Balancing

This subsection reports the storage load distribution results based on 4 SCs.

Figure 7 shows the load distribution for the *Workgroup* set. Since only chunk contents are physically stored on disk ultimately, the logical deduplicated file sizes in fact reflect the storage load of file recipes. At the file level, SC4 contains the most file recipes that correspond to 168.7GB deduplicated files referenced by 1,666.9GB original files. SC1 contains the fewest file recipes that correspond to 132.4GB deduplicated files referenced by 1,396.5GB original files. At the chunk level, SC1 physically stores the most chunk contents at 91.96GB, corresponding to 210.43GB logical chunks, while SC3 physically stores the least chunk contents at 91.91GB, corresponding to 128.31GB logical chunks. Note that the file recipes and chunk contents are distributed according to their associated fingerprints respectively, so the deduplicated file size and the logical chunk size may not be equal in specific SCs, but they are the same in global storage.

During the deduplication for the *Workgroup* set, 84,876,504 hot fingerprints were detected at the chunk level, which means that there are about 80.9GB *zero-chunks* being distributed among files. As the 1024-byte *zero-chunk* is distributed to SC0 in our experiment, the logical chunk size of SC0 is significantly larger than that of the other SCs. However, the physically stored chunk contents are well balanced among all the SCs.

Figure 8 shows the load distribution for the *Campus* set. At the file level, SC0 holds the most file recipes, corresponding to 153.8GB files that are referenced by 1,221.0GB original files. SC3 contains the fewest file recipes, corresponding to 145.3GB files that are referenced by 1157.5GB original files. At the chunk level, SC1 stores the most physical data, for a total of 64.51GB, and SC2 stores the least physical data, for a total of 64.47GB. The corresponding total amounts of logical chunks are 148.44GB and 148.17GB respectively.

A total of 3,953,486 hot fingerprints are detected in the *Campus* set, corresponding to approximately 3.8GB *zero-chunks*. We notice that more *zero-chunks* are detected in the *Workgroup* set, and we speculate that some of them may come from engineering data sets such as virtual disk files generated by virtual machine. Clearly, using hot fingerprints to eliminate *zero-chunks* is efficient in improving deduplication throughput.

Most importantly, the experimental results show that storage load can be perfectly balanced after deduplication at both the file level and the chunk level.

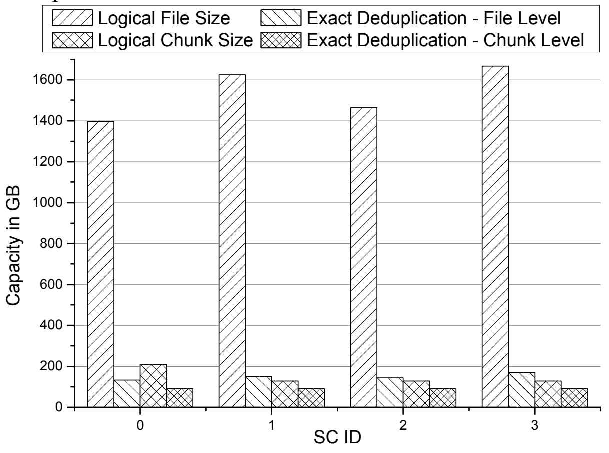


Figure 7: Data Distribution of the *Workgroup* Set among 4 SCs

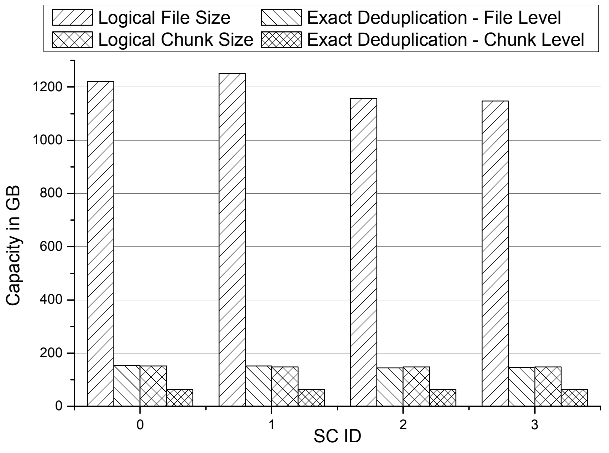


Figure 8: Data Distribution of the *Campus* Set among 4 SCs

E. Throughput

Since our approach target at network backup service applications and can be used to save bandwidth between backup servers and storage components, it is difficult to measure the raw deduplication throughput against data streams. Instead, we trace and report the fingerprint deduplication efficiency. Considering an average chunk size of 4KB, 25,600 chunk fingerprints must be deduplicated per second to achieve a 100MB/s raw deduplication throughput.

In tracing the throughput, we use a single BS (backup server) to send fingerprints to a single SC over a gigabit network without transferring the physical chunk content. Note that one duplicate fingerprint found at the file level means that all the chunk fingerprints belonging to that file can be directly skipped.

For the *Workgroup* set, 12,154,807 file fingerprints and 207,856,782 chunk fingerprints are actually transferred and deduplicated in a period of 982 seconds. For the *Campus* set, 15,391,112 file fingerprints and 132,110,642 chunk fingerprints are actually transferred and deduplicated in a period of 814 seconds. The average throughputs of chunk fingerprint deduplication are 211,667/sec and 162,298/sec respectively for the two data sets, which correspond to raw deduplication throughputs of 827MB/s and 634MB/s respectively for 4KB-sized chunks. Considering the acceleration effect of the file-level deduplication, MAD2 achieves much better efficiencies, corresponding to raw deduplication throughputs of 6,415MB/s and 6,011MB/s respectively for the two data sets.

Since deduplication throughput can be affected by chunk locality, compressibility and many other data characteristics, the actual throughput may vary depending on data sets, as evidenced by our results.

Further, our tracing of the disk access times during deduplication indicates that disk access times are negligible during the file-level deduplication because of the relatively small number of deduplicated file fingerprints. For the chunk-level deduplication for the *Workgroup* set, 82 tankers are generated by DMC, and 757,939 bucket load operations and 6,811 batch write-back operations are performed by SAC. For the chunk-level deduplication for the *Campus* set, 55 tankers are generated by DMC, and 531,328 bucket load operations and 3,237 batch write-back operations are performed by SAC. These results demonstrate that our cache policy is effective in preserving and exploiting data locality and disk access locality.

F. RAM Usage

Although the compression ratio may be different from one data set to another, RAM consumption of our deduplication approach is mainly related to the physical size of non-duplicate data.

For a 10TB deduplicated data set, there will be a total of approximately 40×2^{20} files and 2.5×2^{30} chunks if the average file size is reasonably assumed to be 256KB [16] and the average chunk size is set to 4KB. Assuming a capacity of 2^{20} fingerprints, 40 tankers are required to hold the file fingerprints and 2,560 tankers to hold the chunk fingerprints. By limiting the false positive rate of Bloom Filter Array (BFA) to an extremely low level of $1/2^{20}$, we need about 144MB to hold the file-level BFA and 9.2GB to hold the chunk-level BFA. If 800MB is used to construct the in-memory cache, the total RAM consumption will be approximately 10GB, which is a reasonable size that most modern storage servers can afford.

As the above estimate shows, BFA consumes most of the RAM capacity. To further reduce the RAM space requirement, we propose three possible solutions. The first solution is to increase the average chunk size. For example, doubling the average chunk size, the number of chunks and thus the required RAM space will be halved, at the expense of less detectable duplicate data. The second solution is to simply allow a much higher false positive rate. For example, by increasing the false positive rate from $1/2^{20}$ to $1/2^{9}$, the total RAM consumption by BFA will be reduced from 9.2GB to 4.2GB. Considering the fact that higher false positive rate can cause more cache replacement operations and affect the throughput, our third solution configures the chunk-level deduplication to run on a round-robin manner among multiple SCs on the same storage node, while keeping the file-level deduplication unchanged since it consumes much less memory. Consequently, with n SCs rotating to execute the chunk-level deduplication one at a time on a round-robin basis, the memory requirement will be reduced to approximate $1/n$, but at the cost of reduced chunk-level deduplication throughput.

G. Comparison between MAD2 and Extreme Binning

This subsection compares our MAD2 approach to Extreme Binning. MAD2 approach exploits data locality to enable scalable high-throughput exact deduplication in backend storage of network backup service, while Extreme Binning exploits file similarity to achieve scalable parallel approximate deduplication in backup workloads with little locality [7]. In general, data locality is the main factor that affects the throughput of MAD2, whereas file similarity is the main factor that influences the compression ratio of Extreme Binning.

Since detailed organization of file recipes and chunk contents of Extreme Binning are not reported in the literature, we instead implement the fingerprint deduplication mechanism of Extreme Binning, which enables us to measure its data deduplication efficiency (compression ratio) and fingerprint deduplication efficiency (throughput) that we believe are sufficient for the purpose of our comparison.

Table 1 compares the two approaches in terms of targeted applications, deduplication efficiency, load balancing, throughput, memory requirement, and deletion support, summarizing results reported in this section.

Table 1: Comparison between MAD2 and Extreme Binning

Approach	MAD2	Extreme Binning
Targeted application	Network backup services	Non-traditional backup workloads with little locality
Deduplication efficiency (compression ratio)	16.73 for the *Workgroup* set, 18.52 for the *Campus* set	14.10 for the *Workgroup* set, 9.71 for the *Campus* set
Load balancing	Yes	Yes
Fingerprint deduplication efficiency (throughput)	6,415MB/s for the *Workgroup* set, 6,011MB/s for the *Campus* set	1,722MB/s for the *Workgroup* set, 1,612MB/s for the *Campus* set
Memory requirement for 10TB exactly deduplicated data with an average chunk size of 4KB	Estimated 10GB at the most, can be reduced by memory-saving solutions of Section 4.6	Estimated between 1.2GB to 2.4GB
Deletion support	Yes	N/A

Experimental results show that the compression ratio of Extreme Binning lies between our exact deduplication results of the file level and the chunk level. And for low-similarity data sets, deduplication efficiency of MAD2 is far better than that of Extreme Binning.

Both approaches distribute data according to fingerprint prefixes and can achieve storage load balance. The difference is that our approach distributes file recipes and chunk contents respectively, while Extreme Binning distributes the whole bins according to their representative fingerprints.

Since Extreme Binning must choose the representative fingerprint for each file and can not eliminate all duplicates at the file level, more fingerprints need to be deduplicated at the chunk level. This is also evidenced by our experimental results showing that MAD2 is more efficient in deduplication than Extreme Binning for data sets with exploitable locality.

We estimate the RAM requirements of Extreme Binning for a dataset corresponding to 10TB exactly deduplicated data, assuming 60-byte as the record size of the primary index [7] and 512KB as the average size of the chunk data contained in a bin, which is twice the average file size. Assuming that Extreme Binning generates 10TB deduplicated data for the best case scenario and 20TB approximately deduplicated data for a near worst case scenario, the RAM requirements will be approximately 1.2GB and 2.4GB respectively, which is more memory efficient than MAD2. It must be noted that, by using memory saving solutions introduced in section 4.6, it is possible for the MAD2 approach to achieve similar memory efficiency to Extreme Binning.

V. Related Work

Several approaches have been previously proposed to avoid the duplicate-lookup disk bottleneck and enable efficient deduplication.

DDFS [5] exploits chunk locality to achieve high-throughput exact deduplication for D2D backup. It preserves locality by a Stream-Informed Segment Layout and exploits locality with Locality Preserved Cache. An in-memory Bloom Filter is also used to accelerate non-duplicate chunk identification.

Sparse Indexing [6] is an approximate deduplication technique designed for D2D backup. It divides data stream into variable-sized chunks to construct segments, which are then sampled and mapped to a compact in-memory sparse index. Incoming segments are only deduplicated against several existing similar segments selected according to the sparse index. Its deduplication quality is dependent on the sampling rate and a few other parameters.

Both DDFS and Sparse Indexing are designed for D2D backup workloads, and do not address the scalability issue in a distributed environment. The following two recent studies proposed scalable deduplication approaches.

HYDRAstor [4], a scalable secondary storage solution, constructs its backend using a grid of storage nodes built around a distributed hash table. The backend maintains large-scale variable-sized, content-addressed, immutable, and highly-resilient data blocks that are logically organized in a directed acyclic graph. Duplicate blocks are eliminated according to their hashes. However, there are some outlier cases for which duplicates can not be detected. HYDRAstor adopts an average block size of 64KB, among other constraints, to keep all the metadata in memory and avoid the duplicate-lookup disk bottleneck.

Extreme Binning [7] is a scalable parallel deduplication approach that targets at non-traditional backup workloads that consist of low-locality individual files. It groups highly similar files into bins, and eliminates duplicate chunks inside each bin. Duplicate chunks are allowed to exist among different bins, resulting in approximate deduplication. By keeping only the primary index in memory, Extreme Binning can reduce the RAM requirement while maintaining a reasonably high throughput.

There are also deduplication systems designed for personal storage and SAN cluster. Foundation [17] uses content-addressed storage to archive nightly snapshots of users' disks. Fixed-size duplicate blocks will be removed during the write operations. DEDE [18] is a fixed-size block-level deduplication system for SAN-cluster file systems. It uses out-of-band deduplication to minimize the impact on system performance. Hosts maintain their own recent writes to cluster file system in on-disk logs. Each host periodically updates a shared index to reflect these recorded writes and reclaim duplicate blocks.

Many earlier deduplication systems mainly focus on improving storage efficiency by eliminating duplicates at the file level, fixed-size block level, or variable-sized chunk level. Farsite [19] and EMC's Centera [20] identify and eliminate duplicate data by comparing the hash of the whole file or fixed content. Venti [2], a block-level archival storage, removes redundant fixed-size data blocks by comparing their secure hashes. Pastiche [21] utilizes chunk-level duplicate detection to construct a resource-saving peer-to-peer backup network. Deep Store [3], a large scale archival storage system, uses both variable-sized chunk-level deduplication and delta compression to save storage. REBL [10] uses content-defined chunking algorithm to divide data and identify duplicate chunks by their hash. It utilizes delta encoding and sequential compression to further improve the storage efficiency. Jumbo Store [22] organizes variable-sized chunks into Hash-Based Directed Acyclic Graphs to save both storage and bandwidth while performing incremental upload and versioning for a utility rendering service. All these systems are very different from our MAD2 approach in terms of target applications, deduplication granularity, throughput and scalability.

Previous works have evaluated deduplication efficiencies at different granularities [8]. Storage efficiencies of chunk-level deduplication, delta encoding and traditional sequential compression have also been compared [23].

Duplicate elimination has also been used in bandwidth-saving network protocols [24], low-bandwidth distributed file systems [9, 25], replica synchronization [26], multi-source download acceleration [27], and other network applications [28-30]. There are also techniques that can save disk storage space while providing timely recovery to any point-in-time [31].

Rabin Fingerprinting [12], a low computational complexity hash method, has been widely used in content-defined chunking and similarity detection [11, 32-34].

The idea of using *Bloom Filter Array* to accelerate duplicate locating is inspired by HBA [35, 36], a decentralized metadata lookup scheme. An excellent survey on network applications of Bloom Filters can be found in [14]. Hot *zero-chunks* have also been detected and discussed in [37] and [38].

MAD2 distributes file recipes and chunk contents according to the prefixes of corresponding fingerprints. Techniques such as RUSH [39], CAN [40] and LH* [41] can also be used to enhance scalability and reliability. DHT has been widely used in large-scale distributed storage systems and peer-to-peer systems to balance load [42, 43], distribute data and locate resources [44-46].

VI. CONCLUSIONS

This paper presents a scalable high-throughput exact duplication approach, called MAD2, to eliminate duplicates both at the file level and at the chunk level in backend storage of network backup services. MAD2 utilizes on-disk Hash Bucket Matrix to preserve fingerprint locality and integrates in-memory Dual Cache to capture and exploit locality. In addition, MAD2 employs Bloom Filter Array to efficiently identify unique incoming fingerprints and indicate where a duplicate may reside. By employing a DHT-based Load-Balance technique to distribute file recipes and chunk contents among multiple storage nodes in their backup sequences, MAD2 further enhances performance with a well balanced load.

Experimental results show that the storage efficiency of MAD2 is generally very good and sometimes far better than that of the approximate deduplication based on Extreme Binning. Both file recipes and chunk data can be well balanced among multiple storage components (SCs). The average throughputs of fingerprints deduplication measured in our experiments show that MAD2 is capable of supporting a raw deduplication throughput of at least 100MB/s for each SC. By adopting an average chunk size of 4KB, MAD2 requires about 10GB RAM for 10TB exactly deduplicated data in full functional mode, which is affordable for most modern storage servers. Also, we have proposed three possible solutions to minimize the RAM consumption. Moreover, we believe that the MAD2 ideas of using built-in fingerprint to eliminate hot *zero-chunks* and using Bloom Filter Arrays to accelerate duplicate detection are also applicable to D2D oriented deduplication such as DDFS.

ACKNOWLEDGMENTS

This work is supported in part by the National High Technology Research and Development Program (863 Program) of China under Grant No.2009AA01A402, the Program for New Century Excellent Talents in University NCET-06-0650, and the US NSF under Grants NSF-IIS-0916859, NSF-CCF-0937993 and NSF-CCF-0621526. The authors are grateful to the anonymous reviewers for their many helpful comments and suggestions. We also thank all the experiment participants for their continuous support.

REFERENCES

[1] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, “A Cooperative Internet Backup Scheme,” in *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, TX, USA, June 2003, pp. 29-42.

[2] S. Quinlan and S. Dorward, “Venti: a new approach to archival storage,” in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, Monterey, CA, USA, 2002, pp. 89-102.

[3] L. L. You, K. T. Pollack, and D. D. E. Long, “Deep Store: An Archival Storage System Architecture,” in *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, Washington, DC, USA, 2005, pp. 804-8015.

[4] C. Dubnicki, L. Gryz, L. Heldt, M. Kaczmarczyk, W. Kilian, P. Strzelczak, J. Szczepkowski, C. Ungureanu, and M. Welnicki, “HYDRAstor: a Scalable Secondary Storage,” in *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, CA, USA, Feb. 2009.

[5] 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)*, San Jose, CA, USA, Feb. 2008, pp. 269-282.

[6] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Campbell, “Sparse Indexing: Large Scale, Inline Deduplication Using Sampling and Locality,” in *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*,San Francisco, CA, USA, Feb. 2009, pp. 111-123.

[7] D. Bhagwat, K. Eshghi, D. D. E. Long, M. Lillibridge, “Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup,” in *Proceedings of the 17th IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, London, UK, Sept. 2009.

[8] C. Policroniades and I. Pratt, “Alternatives for Detecting Redundancy in Storage Systems Data,” in *Proceedings of the 2004 USENIX Annual Technical Conference*, Boston, MA, USA, June 2004.

[9] A. Muthitacharoen, B. Chen, and D. Mazieres, “A Low-bandwidth Network File System,” in *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*, Banff, Canada, Oct. 2001, pp. 174-187.

[10] P. Kulkarni, F. Douglis, J. LaVoie, J. M. Tracey, “Redundancy Elimination Within Large Collections of Files,” in *Proceedings of the 2004 USENIX Annual Technical Conference*, Boston, MA, USA, June 2004.

[11] A. Z. Broder, “On the resemblance and containment of

documents," in *Proceedings of the Compression and Complexity of Sequences (SEQUENCES)*, Washington, DC, USA, June 1997, pp. 21–29.

[12] M. O. Rabin, "Fingerprinting by random polynomials," *Report TR-15-81*, Center for Research in Computing Technology, Harvard University, 1981.

[13] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422-426, July 1970.

[14] A. Z. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, pp. 485-509, 2005.

[15] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. IT-23, pp. 337-343, May 1977.

[16] N. Agrawal, W. J. Bolosky, J. R. Douceur, J. R. Lorch, "A Five-Year Study of File-System Metadata," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, Berkeley, CA, USA, 2007.

[17] S. Rhea, R. Cox, and A. Pesterev, "Fast, Inexpensive Content-Addressed Storage in Foundation," in *Proceedings of the 2008 USENIX Annual Technical Conference*, Boston, MA, USA, June 2008, pp. 143-156.

[18] A. T. Clements, I. Ahmad, M. Vilayannur, and J. Li, "Decentralized Deduplication in SAN Cluster File Systems," in *Proceedings of the 2009 USENIX Annual Technical Conference*, Jan. 2009.

[19] A. Adya, W. J. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, Dec. 2002.

[20] EMC Corp., "EMC Centera: Content-Addressed Storage System," http://www.emc.com/products/detail/hardware/centera.htm.

[21] L. P. Cox and B. D. Noble, "Pastiche: Making Backup Cheap and Easy," in *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Boston, MA, USA, Dec. 2002.

[22] K. Eshghi, M. Lillibridge, L. Wilcock, G. Belrose, and R. Hawkes, "Jumbo Store: Providing Efficient Incremental Upload and Versioning for a Utility Rendering Service," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, USA, Feb. 2007, pp. 123-138.

[23] L. L. You and C. Karamanolis, "Evaluation of Efficient Archival Storage Techniques," in *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST)*, College Park, MD, USA, Apr. 2004.

[24] N. T. Spring and D. Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic," in *Proceedings of ACM SIGCOMM*, Aug. 2000, pp. 87-95.

[25] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, A. Perrig, and T. Bressoud, "Opportunistic Use of Content Addressable Storage for Distributed File Systems," in *Proceedings of the 2003 USENIX Annual Technical Conference*, San Antonio, TX, June 2003, pp. 127-140.

[26] N. Jain, M. Dahlin, and R. Tewari, "TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization," in *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST)*, Dec. 2005.

[27] H. Pucha, D. G. Andersen, and M. Kaminsky, "Exploiting Similarity for Multi-Source Downloads Using File Handprints," in *Proceedings of the 4th Symposium on Networked System Design and Implementation (NSDI)*, Cambridge, MA, Apr. 2007.

[28] J. C. Mogul, Y.-M. Chan, and T. Kelly, "Design, Implementation, and Evaluation of Duplicate Transfer Detection in HTTP," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, Mar. 2004, pp. 43-56.

[29] S. C. Rhea, K. Liang, and E. Brewer, "Value-Based Web Caching," in *Proceedings of the 12th International Conference on World Wide Web (WWW)*, Budapest, Hungary, May 2003, pp. 619-628.

[30] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum, "Optimizing the Migration of Virtual Computers," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, Dec. 2002.

[31] Q. Yang, W. Xiao, and J. Ren, "TRAP-Array: A Disk Array Architecture Providing Timely Recovery to Any Point-in-time," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*, 2006, pp. 289-301.

[32] A. Z. Broder, "Some applications of Rabin's fingerprinting method," in *Sequences II: Methods in Communications, Security, and Computer Science*, R. Capocelli, A. De Santis, and U. Vaccaro, Eds. New York, NY: Springer-Verlag, 1993, pp. 143–152.

[33] U. Manber, "Finding similar files in a large file system," in *Proceedings of the Winter 1994 USENIX Technical Conference*, San Fransisco, CA, USA, Jan. 1994, pp. 1-10.

[34] G. Forman, K. Eshghi, and S. Chiocchetti, "Finding Similar Files in Large Document Repositories," in *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Chicago, Illinois, USA, Aug. 2005, pp. 394–400.

[35] Y. Zhu, H. Jiang, and J. Wang, "Hierarchical Bloom Filter Arrays (HBA): A Novel, Scalable Metadata Management System for Large Cluster-based Storage," in *Proceedings of International Conference on Cluster Computing (CLUSTER)*, Sept. 2004, pp. 165-174.

[36] Y. Hua, Y. Zhu, H. Jiang, D. Feng, and L. Tian, "Scalable and Adaptive Metadata Management in Ultra Large-scale File Systems," in *Proceedings of the 28th International Conference on Distributed Computing Systems (ICDCS)*, Beijing, China, June 2008.

[37] K. Jin and E. L. Miller, "The Effectiveness of Deduplication on Virtual Machine Disk Images," in *Proceedings of SYSTOR*, 2009.

[38] D. Meister and A. Brinkmann, "Multi-Level Comparison of Data Deduplication in a Backup Scenario," in *Proceedings of SYSTOR*, 2009.

[39] R. J. Honicky and E. L. Miller, "Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS)*, Santa Fe, NM, Apr. 2004.

[40] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, San Diego, CA, Aug. 2001, pp. 161-172.

[41] W. Litwin, M.-A. Neimat, and D. A. Schneider, "LH*—A Scalable, Distributed Data Structure," *ACM Transactions on Database Systems*, vol. 21, no. 4, pp. 480-525, 1996.

[42] M. Raab and A. Steger, "Balls into Bins - A Simple and Tight Analysis," in *Proceedings of 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, 1998, pp. 159-170.

[43] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 4, pp. 349-361, 2005.

[44] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, San Diego, CA, Aug. 2001, pp. 149-160.

[45] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329-350.

[46] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A Resilient Global-scale Overlay for Service Deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41-53, Jan. 2004.