Send Orders for Reprints to reprints@benthamscience.net

Improving Read Performance with BP-DAGs for Storage-Efficient File Backup

Tianming Yang*, Jing Zhang and Ningbo Hao

International College, Huanghuai University, Henan, 463000, China

Abstract: The continued growth of data and high-continuity of application have raised a critical and mounting demand on storage-efficient and high-performance data protection. New technologies, especially the D2D (Disk-to-Disk) de-duplication storage are therefore getting wide attention both in academic and industry in the recent years. Existing de-duplication systems mainly rely on duplicate locality inside the backup workload to achieve high throughput but suffer from read performance degrading under conditions of poor duplicate locality. This paper presents the design and performance evaluation of a D2D-based de-duplication file backup system, which employs caching techniques to improve write throughput while encoding files as graphs called BP-DAGs (Bi-pointer-based Directed Acyclic Graphs). BP-DAGs not only satisfy the 'unique' chunk storing policy of de-duplication, but also help improve file read performance in case of poor duplicate locality workloads. Evaluation results show that the system can achieve comparable read performance than non de-duplication backup systems such as Bacula under representative workloads, and the metadata storage overhead for BP-DAGs are reasonably low.

Keywords: Data De-duplication, File Backup, Storage-Efficient, Read Performance.

1. INTRODUCTION

Data explosion [1] has been forcing backups to expand storage capacity, which makes modern enterprises face significant cost pressures and data management challenges. Studies showed that storage cost share of the enterprise information resource planning is rising, which has reached more than 50% [2]. With the expansion of system capacity, the storage management overhead can increase several times more than that by the storage hardware devices [3-5]. In addition, the high-continuity of application requires that data should be backed up and failure recovered as quickly as possible. So, the key challenge for modern enterprises data protection is to construct storage-efficient backup systems with high performance on both data write and read throughputs.

Traditional backup systems used to rely on magnetic tapes to archive data due to their cheap, large capacity and removability for off-site backup. With the increase of disk capacity and reliability [6], more and more backup solutions are built on top of hundreds or thousands of hard drives, which are known as Disk-to-Disk (D2D) technique in storage industry [7]. As the disk seek and rotation time is 2 to 3 orders of magnitude shorter than the tape rewinding time, a D2D-based backup system has the advantage of fast backup/recovery compared with Disk-to-Tape (D2T) solutions. More importantly, D2D backup can support global-scale data de-duplication that in turn dramatically improves the effective capacity of D2D device [8]. Now, D2D-based de-duplication storage is gaining popularity and new schemes are emerging to provide more storage-efficient and high performance data protection for enterprises [9-13].

In de-duplication, files or streams are divided into chunks and then duplicate chunks are eliminated in the global system [14]. Each chunk is stored and addressed by its fingerprint (the cryptographic hash such as SHA-1 [15] value of its content) to ensure that only 'unique' chunk is stored. A disk index is used to establish mapping between fingerprint and the location of its corresponding chunk on disk. The key challenge regarding performance is how to reduce the significant random disk I/O overhead to search for chunks on disk index [9]. Most of the existing de-duplication systems use caching technique, which judiciously exploits duplicate locality within the backup stream to avoid the disk index bottleneck, and hence achieves high de-duplication throughput [9, 10]. Duplicate locality refers to the tendency for chunks in backup streams to reoccur together. That is, when a backup stream contains a chunk A, it is surrounded by chunks B, C, and D, then in a different backup if chunk A appears it is very likely that chunks B, C, and D will also appear nearby.

Apart from improving de-duplication write throughput, another important issue for de-duplication backup is how to read files from the system more efficiently. Although restore is not much more frequent event than backup, read throughput is still very important especially in high-continuity application environments that require very short RTO (Recovery Time Objective). In existing de-duplication systems file chunks are indexed by their fingerprints (i.e., hash pointers), which are called Content-Addressed Storage (CAS) [14]. However, reading a file from the system using hash pointers involves time-consuming disk index seeks, and one disk index seek per file chunk is far too slow. The use of caching

*Address correspondence to this author at the International College, Huanghuai University, Zhumadian, Henan, 463000, China;
Tel: +8615290193519; E-mails: ytmzqyy@163.com, 2:282353724@qq.com

1874-1290/13

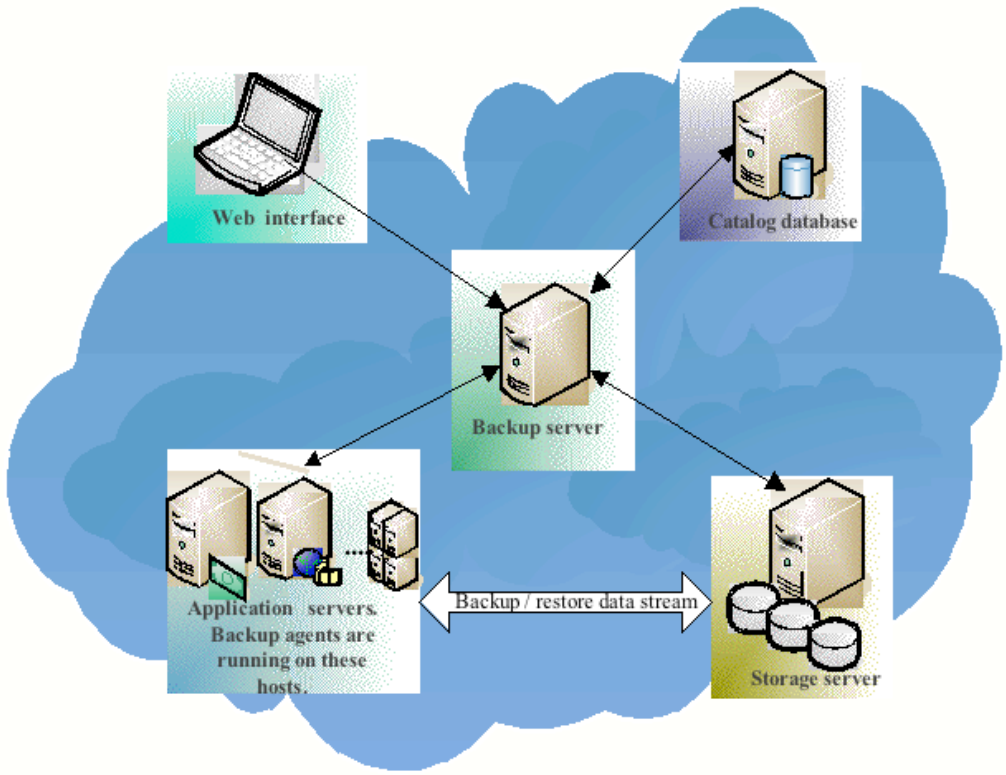

Fig. (1). The file backup architecture.

techniques mentioned above can reduce the disk index seeks for reading chunks, but their read performance is heavily degraded under conditions of poor duplicate locality. Different from large data streams that have good duplicate locality, poor duplicate locality workloads consist of random files from disparate sources such as file backup requests made by Network-attached Storage (NAS) clients, Continuous Data Protection (CDP) where files are backed up as soon as they are changed, and electronic emails are backed up as soon as they are received [16]. In order to maintain high read throughput under various workloads, files were encoded as graphs called Bi-Pointer-based Directed Acyclic Graphs (BP-DAGs) whose nodes had variable-sized chunks of data and whose edges were hash plus address pointers. The proposed BP-DAGs not only supported automatic structure sharing to satisfy the 'unique' chunk storing policy of de-duplication system but also directed access to chunks to completely avoid disk index seeks when restoring a file.

The rest of this paper is organized as follows. Section 2 gives the system architecture related to de-duplication backup and storage. Section 3 describes the structure of BP-DAGs. Section 4 presents the evaluation results in terms of de-duplication ratio, file write\read throughputs and the metadata storage overhead of BP-DAGs. Finally, section 5 contains conclusion.

2. THE STORAGE-EFFICIENT FILE BACKUP

The storage-efficient backup system is designed to provide a flexible data protection solution for companies whose application servers may be dispersed across the Internet. In this section, we introduce the architecture, de-duplication process and disk storage policy of this system.

2.1. System Architecture

The architecture of this system is shown in Fig. (**1**); it consists of the backup server, backup agent, storage server, catalog database, and web interface. Backup server controls the entire system; it supervises the backup, restore, verification and resource management jobs. Users can access the system to make their data protection plans through Web Interface with no time/location constraints. Backup agent runs as a background daemon on the application server where recently generated data needs to be backed up periodically. When doing a backup/restore job, backup agent sends/receives data to/from the storage server. In the backup process, data is de-duplicated, files are encoded into BP-DAGs and metadata associated with the backup job such as job ID, session time, root chunks of the BP-DAGs, etc. is sent to a catalog database to ensure that data can be recovered if necessary. This paper mainly focuses on data de-duplication and BP-DAGs, and not on the backup job management, so, the rest of the section is dedicated to workflow of backup agent and storage server.

2.2. De-duplication Backup Process

Fig. (**2**) shows the de-duplication backup process. The backup agent divides the input backup stream (it may consist of general files /directories or large tar files) into variable-sized chunks using the content-defined chunking algorithm [17] with an expected chunk size of 8KB, computes the SHA-1 hash [15] (20 bytes in size) of each chunk as its fingerprint, and sends the fingerprints in the order that they appear in the backup stream to the storage server. The storage server performs data de-duplication on the received fingerprints to identify new fingerprints and then informs the

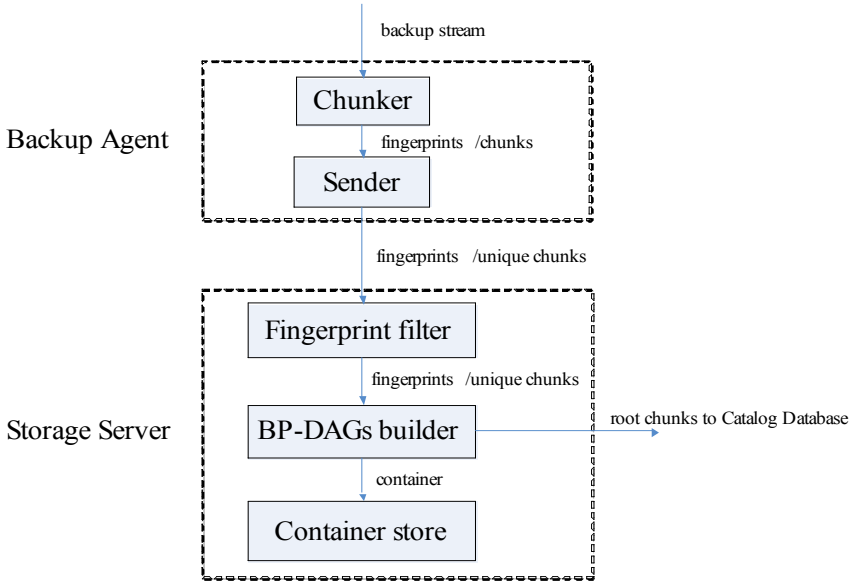

Fig. (2). Block diagram of the de-duplication backup.

backup agent to send the corresponding new chunks (i.e., unique chunks in the scope of the entire system). Since only new chunks are transmitted over the network, the network bandwidth overhead of the backup can be reduced.

The de-duplication process is performed by the *fingerprint filter* module of the storage server, which uses caching techniques to improve de-duplication throughput similar to most of the existing de-duplication systems such as DDFS [9] and Foundation [18]. Specifically, an in-memory Bloom filter [19] and a fingerprint cache have been used. The Bloom filter represents an in-memory conservative summary of the disk index, which can identify most of the new fingerprints without needing query to the disk index, and the fingerprint cache exploits duplicate locality to prefetch groups of chunk fingerprints to improve cache hit rate. If an incoming fingerprint is identified as new (not duplicate) by the Bloom filter then it is indeed a new fingerprint in the entire system, since the Bloom filter does not give a false negative, otherwise it is duplicate with very high probability (depending on the false positive rate of the Bloom filter that can be maintained at very low levels such as lower than 2% by using an adequately large Bloom filter as per the system capacity). If an incoming fingerprint is determined as duplicate by the Bloom filter, then it can be searched in the in-memory fingerprint cache; if it is there, it will definitely be a duplicate; otherwise it can be searched on the disk index. If an incoming fingerprint is found on the disk index, all the fingerprints are read on the same container with the incoming fingerprint from the *container store* to the in-memory fingerprint cache since these fingerprints are more likely to be accessed together in the near future due to duplicate locality. It should be noted that in order to preserve duplicate locality, new chunks and their fingerprints were stored in the order that they appeared in the backup stream to fixed-sized (e.g. 8MB) containers. The *container store* module is responsible for container management such as reading from or writing to the disk storage a container by its container ID. By using caching techniques, the disk index I/O bottleneck for de-duplication backup can be avoided under well duplicate locality workloads. The *BP-DAGs builder* module of the storage server encodes files into BP-DAGs based on the results of the *fingerprint filter*; details of the BP-DAGs are given in the next section.

2.3. Write-Once Storage Policy

For container storage, a write-once policy was imposed to retain backup data in perpetuity, that is, containers were stored on a write-once disk RAID (Redundant Arrays of Inexpensive Disks) [6]. This changes the concept of traditional backup which keeps data for a limited period of time and reclaims storage space from outdate backup data. Traditional archival storage may retain data in perpetuity, but it usually uses off-site tapes or optical jukeboxes as storage media and suffers from expensive administration cost. Write-once disk storage has been used due to the following reasons: first, the rapid advances in disk storage technology witnessed in recent years have notably improved the disk storage capacity while reducing its cost. Second, the de-duplication technique dramatically reduces the disk storage requirement for data protection [9], making it feasible to impose a write-once policy to the back-end storage, actually, there has been successful case to do so [14]. Third, one more attractive benefit of this approach is that it simplifies data management, since all the backup data are retained abidingly, users can access any history versions of information once backed up in the system, not worrying about accidental deletion of data, nor needing to decide a policy for ILM (Information Lifecycle Management)[20]. Finally, the write-once policy makes it possible to encode files as BP-DAGs and to append chunks to disk densely without fragmentation, thus giving rise to high data recovery throughput.

3. BI-POINTER-BASED DIRECTED ACYCLIC GRAPHS

From the perspective of the storage server, all the data is stored as chunks. A chunk can be shared by multiple files. On the other hand, a file may consist of many chunks and a file index should be built to access file chunks. Traditional

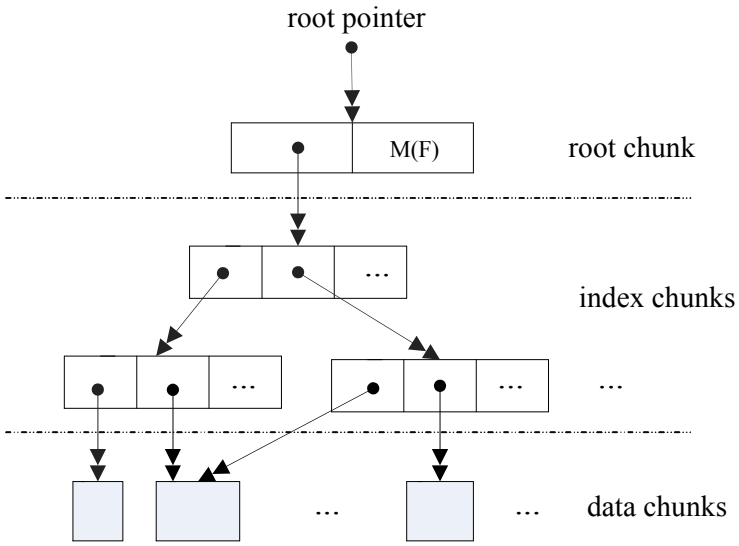

Fig. (3). A BP-DAG representation of a file, M(F) represents the file metadata.

file systems build file index by address pointers to physical blocks. For example, UNIX file systems use an inode structure, and indirect blocks if necessary, to map logical block addresses to the corresponding physical addresses on disk. This address-based index can provide quick access to file data, but cannot support duplicate block elimination and 'unique' block sharing since address pointers are completely block content agnostics. In contrast, the hash-based index such as the Venti hash tree [14] and HDAGs [21] refers to child nodes by their hash rather by their location on disk, hence can well support 'unique' chunk storing and sharing, but its data access performance can be very poor since hash pointers should be translated to disk addresses and this usually incurs time-consuming disk index lookups.

3.1. The Structure of BP-DAGs

In order to combine the advantages of address-based and hash-based indexes, BP-DAGs have been used to establish a number of relationships between files and chunks. A BP-DAG is a special kind of DAG (Directed Acyclic Graph) whose nodes refer to other nodes by their hash and their address on disk. Specifically, a BP-DAG pointer is a <*type, fingerprint, address, size* > group with a total of 32 bytes in size, which records the chunk type (1 byte), fingerprint (20 bytes), storage address (9 bytes with 5 bytes for container ID and another 4 bytes for storage offset within the container) and chunk size (2 bytes, we impose a limitation of maximum chunk size of 64KB) of the corresponding child, respectively. The fingerprint is a hash pointer and the < *address, size* > is an address pointer, so a BP-DAG pointer is actually a hash plus address pointer pair which has been pictorially denoted as a double arrow. Fig. (**3**) depicts a file represented as a BP-DAG. There are three different types of BP-DAG nodes that are all stored on disk as chunks, namely the root chunk, index chunk and data chunk. A BP-DAG has a unique root chunk that contains two fields: the metadata field that stores the file metadata, and the pointer field that stores BP-DAG pointers to file data. Data chunks contain the contents of the file and locate the leaf of the BP-DAG. For a small file, its data chunks can be directly indexed by the root chunk pointer field. For a large file, more than one index chunk, which is an array of BP-DAG pointers, may be required to constitute a hierarchical index structure to access its data. It should be noted that although a BP-DAG is a hierarchical structure, it is in general not a tree, but rather a DAG since one child can have multiple parents.

BP-DAGs use variable-sized index chunks with a maximum size of 32KB. An index chunk can contain a maximum of 1024 BP-DAG pointers (32KB/ 32bytes =1024), so an index layer with a depth of 2 can index a maximum of 1024*1024 data chunks. An expected chunk size of 8KB means 1024*1024*8KB=8GB, which is exceeding the size of most files, so, most file's index layers are no more than 2 in depth. In practice, for most small files, there are no index chunks in their BP-DAGs.

Hash pointers also give BP-DAGs a number of useful properties. First, all the BP-DAGs chunks are uniquely stored in the system, no duplicate chunks exist, this property satisfies the 'unique' chunk storing policy of de-duplication system. Second, a BP-DAG is automatically acyclic since creating a cycle in the parent-child relationship between two BP-DAG nodes is cryptographically hard [21]. Third, more importantly, BP-DAGs are intrinsically structure sharable. Except root chunks, all the BP-DAGs chunks can be shared. For example, two files with the same data contents but different metadata (e.g., different names or even the same name but in different jobs) will have different root chunks but share the same index chunks and data chunks. Fig. (**4**) illustrates the structure sharing between two BP-DAGs.

3.2. BP-DAGs Building

When a file is de-duplicated, the *BP-DAGs builder* module sequentially reads its fingerprints and new chunks from the *fingerprint filter* to build BP-DAGs. It maintains an in-memory container to which BP-DAGs chunks are written. When the in-memory container is full, it is flushed to the *container store* and then another empty in-memory container is created and its container ID is requested. To make a fingerprint *h* read from the *fingerprint filter*, the BP-DAGs building algorithm does the following:

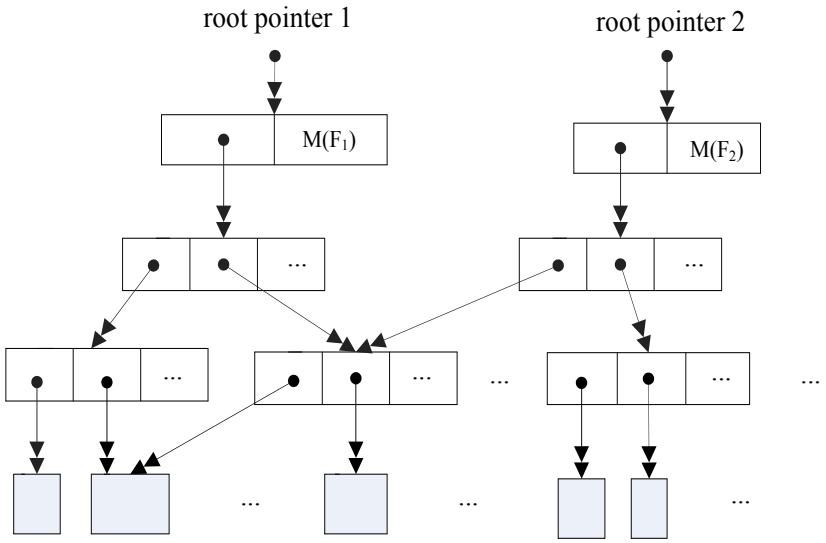

Fig. (4). Structure sharing between BP-DAGs.

- If it is a duplicate, then its < *address, size* > from the fingerprint cache (see Section 2.2) is read and stored < *dc, h, address, size* > to the BP-DAG. Here, '*dc*' represents the chunk type '*data chunk*'.
- If it is new, it stores the new data chunk $D(h)$ to the in-memory container, and its <*dc, h, address, size* > is stored to the BP-DAG. The address pointer < *address, size* > can be obtained from the in-memory container ID and the chunk's storage offset within the container.

For a small file, BP-DAG pointers are directly stored in its root chunk. For a large file, index chunks are recursively created in the process until in the end a unique root chunk is created and stored for the file. When an index chunk is generated, its fingerprint is sent to the *fingerprint filter* to check if it is new. If it is new, the index chunk is stored to the container and its new address is built to the BP-DAG, otherwise, it is discarded and its address pointer is copied from the fingerprint cache to the BP-DAG. Although storing an index chunk incurs de-duplication overhead, its impact on backup performance is negligible, since the amount of index chunks is far smaller than that of the data chunks.

3.3. Restoring Files from BP-DAGs

BP-DAGs are effective structures for fast access to file data. BP-DAGs root chunks are much like UNIX file system inodes and can be used to improve file read performance. During backup, the root chunks are also sent to a catalog database. When doing a restoring job, they are directly read to an in-memory read cache for quick file restoration. The desired file chunk (data chunk or index chunk) is directly read from the cache if found there. Otherwise, according to its BP-DAG pointer, the container that stores the requested chunk is read to the cache. The BP-DAG pointers avoid time-consuming disk index lookups to search chunks. Moreover, the locality-preserved container guarantees that a BP-DAG's chunks are stored on the disk of a continuous region if they are not shared by previous files. This continuous data layout improves cache hit rate that in turn reduces disk I/Os to read chunks. So the performance of file restore can be effectively improved.

4. EXPERIMENTAL EVALUATION

Here, a prototype of the system in Linux called Dedup-BP (De-duplication backup system with BP-DAGs), has been implemented and by modifying Dedup-BP, another system called Dedup has been developed, which removes BP-DAGs and builds file index just as a flat sequence of fingerprints that map to the file chunks. In this section, Dedup-BP has been evaluated in terms of de-duplication ratio, compared with Dedup and Bacula in terms of backup/restore throughputs under representative workloads The comparison of Dedup-BP and Dedup on metadata storage overhead has also been given. Bacula is non de-duplication free network backup software available under the GNU Version 2 software license. There are many versions of Bacula available but bacula-2.0.2 was selected for comparison.

4.1. System Setup

Here, system setup for the evaluation was based on three computers PC1-PC3. The servers (the backup servers, storage servers and catalog databases of Dedup-BP, Dedup and Bacula) run on PC1 and the clients (the backup agents of Dedup-BP, Dedup and Bacula) on PC2. PC1 and PC2 were equipped with Inter 604-pin EM64T (NoconaTM) Xeon 3.0 GHz processor and 4GB DDR SDRAM memory. A Highpoint Rocket 2240 Raid controller attached to 5 SATA disks was connected to PC1 served as container store and disk index for de-duplication backup, the SQLite version 2.8.4 was also installed on PC1 served as catalog database. These two machines were connected over a 1000 Mbit/s Ethernet through a windows 2003 server enterprise internal router run on PC3, which can be configured to measure traffic and impose bandwidth limitations.

In order to examine the benefit of BP-DAGs on improving file restore performance, two representative workloads were used. Workload-1 is based on backup versions in their chronological order from a massive storage system [22], where significant amounts of data remained unchanged between adjacent versions; it represented backup stream workloads with well duplicate locality. There were a total of

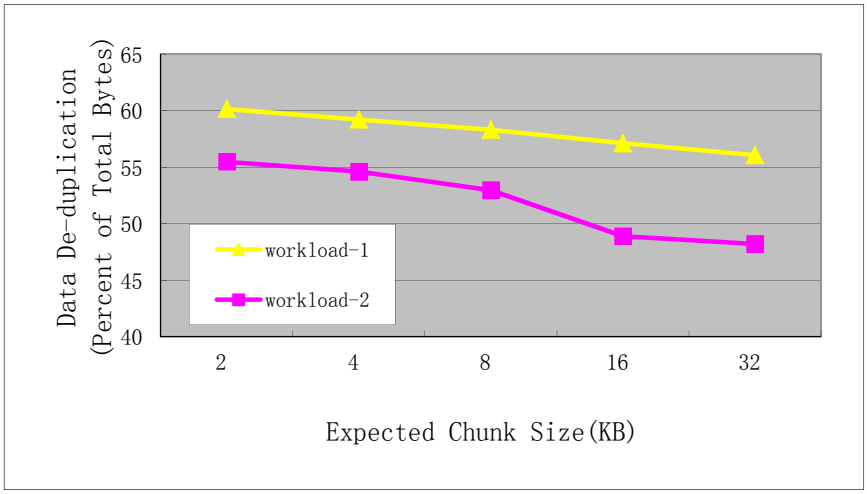

Fig. (5). Data de-duplication using various expected chunk size.

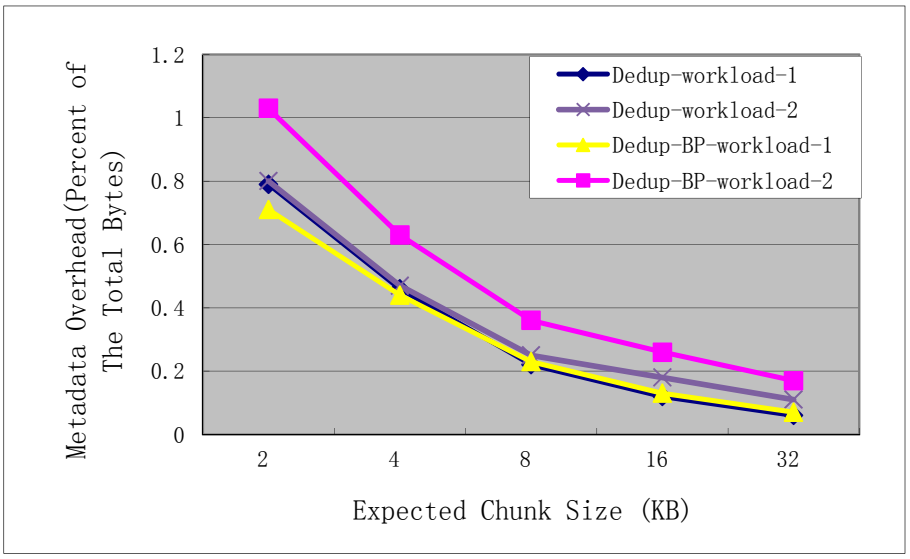

Fig. (6). Metadata storage overheads using various expected chunk size.

572GBs of data, 3177 files in workload-1. Workload-2 was collected from different personal workstations; it contained development sources and documentations, object files generated from code compiling, meeting records and technology discussions, and research papers and reports in various formats and languages. Although there were large percentages of duplicate files in workload-2, majority of them were random small files (usually several ten of kilobytes in size) scattered in various parts of the workload, giving rise to very poor duplicate locality. Workload-2 contained 241GBs of data, 947258 files.

4.2. Results and Discussions

The expected chunk size is an important parameter for de-duplication backup. It is a trade-off between the degree of data de-duplication and the storage overhead of the metadata. Larger chunks not only reduce the storage overhead of the metadata but also reduce the number of duplicate data eliminated.

Fig. (**5**) shows the amount of duplicate bytes eliminated by Dedup-BP on the two workloads as a percentage of the total number of bytes in the workloads. As expected, the number of eliminated duplicates decreases slightly as the chunk size increases because of the coarser granularity. For workload-2, the data de-duplication degree drops faster as the expected chunk size varies from 8KB to 16KB. This may be because many files in this workload are small files around 10KB, expected chunk size larger than 8KB will make most of these files be divided into no more than one chunk. In this case, Dedup-BP will be degraded to ordinary file-based backup and will lose many duplicates. Overall, by performing de-duplication, the storage requirement for backup can be remarkably reduced, as shown in Fig. (**5**), with expected chunk size of 8KB, over 58% and 52% storage spaces were saved when backing up workload-1 and workload-2, respectively. It should be noted that the percentage of storage saved will grow higher over time with the system holding more backup data.

Fig. (**6**) shows the amount of extra storage needed as a percentage of the total number of bytes in the workload for storing the file index. The results do not take into account the file metadata (i.e., file name, ctime, mtime, etc.) overhead

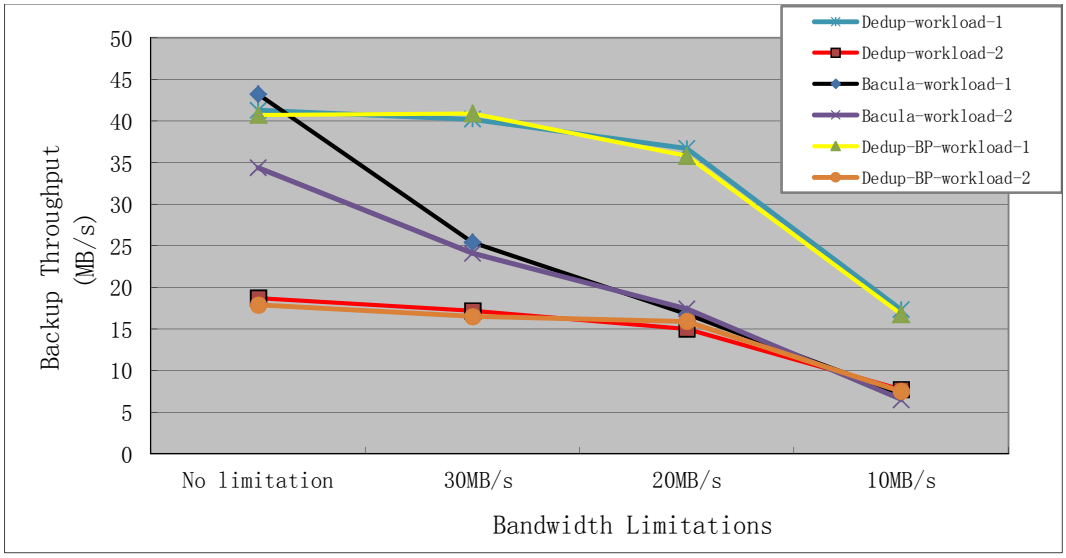

Fig. (7). Backup throughputs on 1000 Mbit/sec LAN with different bandwidth limitations.

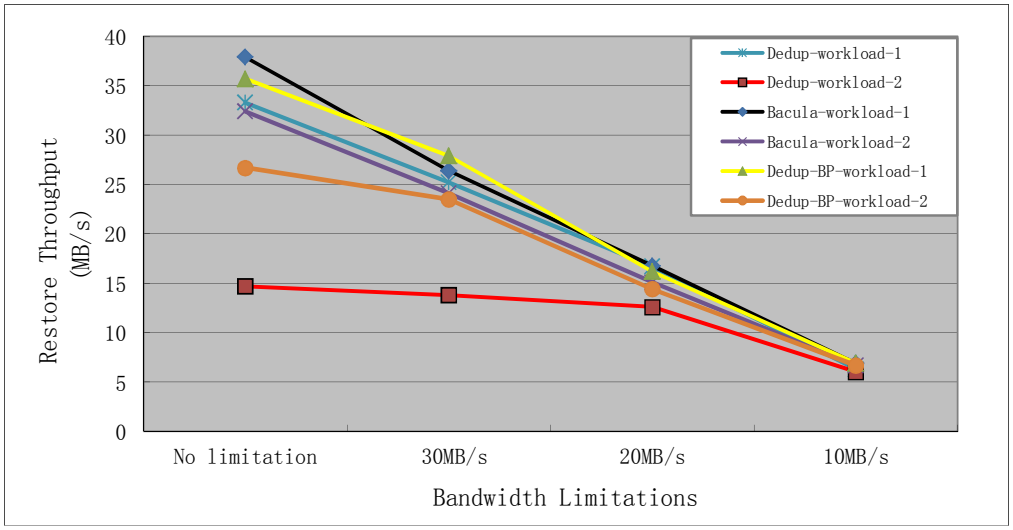

Fig. (8). Restore throughputs on 1000 Mbit/sec LAN with different bandwidth limitations.

for convenience of comparison. The metadata storage overhead is negatively affected by the expected chunk size; smaller expected chunk size suffers from relatively high metadata storage overhead. Moreover, Dedup-BP's metadata storage overhead is also affected by the de-duplication rate and duplicate locality while Dedup does not (Dedup's metadata overhead is only related to the average chunk size). With higher de-duplication rate and good duplicate locality, more index chunks will be shared between BP-DAGs, resulting in lower metadata overhead. This can be seen from Fig. (**6**) where Dedup-BP suffers from relatively low metadata overhead on workload-1 than on wokload-2. Overall, the metadata overhead for BP-DAGs is reasonably low, as can be seen in Fig. (**6**), using an expected chunk size of 8 KB. The amount of metadata storage overhead is very small, less than 0.4%.

Figs. (**7** and **8**) show the backup and restore throughputs of Dedup, Bacula and Dedup-BP on 1000Mbit/sec LAN with different bandwidth limitations. Fig. (**7**) shows that both Dedup and Dedup-BP outperform Bacula in terms of backup throughput on workload-1 in low bandwidth environments. This improvement mainly comes from the de-duplication of data transmitted over the network in backup session. However, on workload-2, both Dedup and Dedup-BP yielded relatively low backup throughput than Bacula, this is because the poor duplicate locality within workload-2 reduced the cache hit rate, making the disk index lookup become a performance bottleneck.

Fig. (**8**) shows that both Dedup and Dedup-BP achieve comparable read throughput than Bacula under workload-1 because of the well duplicate locality contained in this workload. But under workload-2, Dedup experienced a sharp decline in read performance compared with Bacula, this is because workload-2 presents poor duplicate locality, which results in many random small disk I/Os for file read. In contrast to Dedup, Dedup-BP effectively slowed the decline in read performance under workload-2. As expected, Dedup-BP outperforms Dedup in read performance under both workloads, achieving improvements over Dedup by a factor of 1.07 and 1.81 under workload-1 and workload-2 respectively. This is because Dedup-BP indexes files using BP-DAGs that completely eliminate time-consuming disk index

lookups in the read process, while Dedup suffers from a great number of random on-disk fingerprint lookups especially in the case of poor duplicate locality.

5. CONCLUSIONS

Data de-duplication, an emerging backup technique, is sparking a revolution in the area of data protection as it can dramatically reduce the backup storage requirement. Despite its rapid advance witnessed in recent years this technique is still facing many challenges on how to meet the performance requirements in high-continuity application environments. In this paper, the design of a de-duplication backup system that encodes files as BP-DAGs was presented. The system using representative workloads was also evaluated. Experimental results showed that the proposed BP-DAGs can effectively improve file read throughput under poor duplicate locality workloads in which the cases of current mainstream solutions would experience a very sharp performance decline. How to effectively improve the backup throughput under poor duplicate workloads still remains a problem and will be the research direction of our future work.

CONFLICTS OF INTEREST

The authors confirm that this article content has no conflicts of interest.

ACKNOWLEDGEMENTS

This paper was supported by the Key Science-Technology Project of Henan of China under Grant No. 112102210446.

REFERENCES

[1] Gens, F., "IDC Predictions 2012: Competing for 2020", IDC TOP10 Predictions 2012, pp.1-26, [Online] Available: http://cdn.idc.com/research/Predictions12/Main/downloads/IDCTOP10Predictions2012. pdf . [Accessed 21th, March 2013].

[2] L.Y. Lawrence, "*Efficient Archival Data Storage*", PhD thesis, University of California, Santa Cruz, CA, USA, 2006.

[3] C. A. Thekkath, T. Mann, and E. K. Lee, "Frangipani: A Scalable Distributed File System", In: *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)*, 1997, pp. 224-237.

[4] A. C. Veitch, E. Riedel, S. J. Towers, and J. Wilkes, "Towards Global Storage Management and Data Placement", In: *Eighth IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, 2001, pp. 184-184.

[5] J. Wilkes, "Traveling to Rome: Qos specifications for automated storage system management", In: *Proceedings of the Int. Workshop on QoS (IWQoS'2001)*, 2001, pp.75-91.

[6] D. Patterson, G. Gibson, and R. Katz, "A case for redundant arrays of inexpensive disks (RAID)", In: *Proceedings of ACM SIGMOD*, 1988, pp.109-116.

[7] ASARO, T., and BIGGAR, H. *Data De-duplication and Disk-to-Disk Backup Systems: Technical and Business Considerations*. White Paper, the Enterprise Strategy Group, 2007.

[8] BIGGAR, H. *Experiencing Data De-Duplication: Improving Efficiency and Reducing Capacity Requirements*. White Paper, the Enterprise Strategy Group, 2007.

[9] B. Zhu, H. 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*, 2008, pp. 269-282.

[10] M. Lillibridge, K. Eshghi, D. Bhagwat, V. Deolalikar, G. Trezise, and P. Camble, "Sparse indexing: Large scale, inline deduplication using sampling and locality", In: *Proceedings of the 7th USENIX Conference on File And Storage Technologies*, 2009, pp. 111-123.

[11] T. Yang, H. Jiang, D. Feng, Z. Niu, K. Zhou, and Y. Wan, "DE-BAR: a scalable high performance deduplication storage system for backup and archiving", In: *IEEE International Symposium on Parallel and Distributed Processing*, 2010, pp. 1-12.

[12] P. Shilane, M. Huang, G. Wallace, and W. Hsu, "Wan optimized replication of backup datasets using stream-informed delta compression", In: *Proceedings of the 10th USENIX Conference on File And Storage Technologies*, 2012, pp. 57-71.

[13] K. Srinivasan, T. Bisson, G. Goodson, and K. Voruganti, "idedup: Latency-aware, inline data deduplication for primary storage", In: *Proceedings of the 10th USENIX Conference on File And Storage Technologies*, 2012, pp. 307-320.

[14] S. Quinlan and S. Dorward, "Venti: a new approach to archival storage", In: *Proceedings of the USENIX Conference on File And Storage Technologies*, 2002, pp. 89-101.

[15] PUB F, *Secure Hash Standard*. Public Law, 1995.

[16] D. Bhagwat, K. Eshghi, D. D. Long, and M. Lillibridge, "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*, 2009, pp.1-9.

[17] A.Muthitacharoen, B.Chen, and D.Mazieres, "A low-bandwidth network file system", *ACM SIGOPS Operating Systems Review*, vol. 35, pp. 174-187, October 2001.

[18] S. Rhea, R. Cox, and A. Pesterev, "Fast, inexpensive content-addressed storage in foundation", In: *Proceedings of the 2008 USENIX Annual Technical Conference*, 2008, pp. 143-156.

[19] B. Bloom, "Space/time trade-offs in hash coding with allowable errors", Comm. ACM, vol. 13, pp. 422-426, 1970.

[20] Reiner, D., Press, G., Lenaghan, M., Barta, D., and Urmston, R., "Information life-cycle management: the EMC perspective", In: *Proceedings of 20th International Conference on Data Engineering*, 2004, pp. 804-807.

[21] Kave Eshghi, Mark Lillibridge, Lawrence Wilcock, Guillaume Belrose, and Rycharde Hawkes, "Jumbo store: Providing efficient in cremental upload and versioning for a utility rendering service", In: *Proceedings of the 5th USENIX Conference on File And Storage Technologies*, 2007, pp.123-138.

[22] Zeng, L.F., Zhou, K., Shi, Z., Feng, D., Wang, F., Xie, C.S., Li, Z.T., Yu, Z.W., Gong, J.Y., Cao, Q., Niu, Z.Y., Qin, L.J.,Liu, Q., Li, Y., and Jiang, H., "HUSt: a Heterogeneous Unified Storage System for GIS Grid", In: *Proceedings of ACM/IEEE Conference on Supercomputing*, 2006, pp.325-338.

Received: July 04, 2013 Revised: July 07, 2013 Accepted: July 14, 2013