Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DELTA REFERENCE DATA DEDUPLICATION IN LOW UPLOAD BANDWIDTH NETWORK

[1,2]XIANZHUO LIU, [1,2]JINLIN WANG, [1]MING ZHU

[1]Laboratory of Network Communication System and Control, Department of Automation, University of Science and Technology of China, Hefei 230027, China

[2]National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China

E-mail: [1]xianzhuo@mail.ustc.edu.cn , [2]wangjl@dsp.ac.cn, [1]mzhu@ustc.edu.cn

ABSTRACT

When users put their files on remote storage, it is important to update local changes in short time. Due to the redundancy between successive versions of files, compare-by-hash and delta compression have been studied to reduce total volume locally transmit. Better performance are desirable since plain compare-by-hash does not fully exploit redundancy and delta compression lay additional space demands and application oriented limits on local storage system. In this paper, a delta reference approach is proposed to further reduce the volume sent to remote server by dynamically searching for similar file and reference chunks for the chunk sent from the local host. Experiments on practical datasets reveal that the proposed approach can reduce the volume sent to remote server up to 28.3% thereby decreasing transfer time as much as 26.2% in typical low upload bandwidth networks.

Keywords: *Delta Compression, Data Deduplication, Content Defined Chunking, Bandwidth Asymmetric Network, Data Synchronization*

1. INTRODUCTION

With the fast development of information technology, there have been a large amount of new smart terminals in use, such as smart phones, tablet PCs, etc. People can view their documents and handle relative works on their mobile smart terminals anytime, anywhere. With the increasing broadband speed and successful wireless network implementation in large-scale applications in recent years, remote data storage and backup–since its convenience, availability and affordable price–becomes a competitive choice for users to synchronize data among separate devices and to share data with friends.

Despite of the rising speed of broadband, both wired and mobile, upload bandwidth remains far less than download bandwidth. To fit the behaviour of average broadband consumer, the download bandwidth often accounts for a large proportion of the total spectrum [1], whereas upload only a small fraction, from one fourth to less than one percent [2]. When data are transferred from the user to storage server via such a broadband, the low upload bandwidth turns into a bottleneck which significantly increases the total transfer time and results in poor user experience, especially in mobile accesses.

Fortunately, it has been shown that data exchanged between hosts often exhibits high level of redundancy [3, 4]. Approaches, such as chunking and compare-by-hash (CBH) [4, 5], are proposed to eliminate transfer of redundant part of file such that the total transfer time through the low upload bandwidth network is reduced.

The comparison in 2009 [6] implies that delta encoding can achieve higher compression ratio than hash-based deduplication in backup scenarios. While delta compression is widely adopted in version control systems [7, 8], backup servers [9-11] and even file systems [12], transfer deduplication is something different. Terminal users usually do not

Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

hold multi-version of a single file, moreover, it is not economical for mobile terminals to build a delta specific storage system. Modified version of files which users copied from anywhere else would also be a fully new file to the local delta compression system.

Several recent studies had been carried out to combine hash-based deduplication and delta-based deduplication in backup storage systems. Shilane et al. proposed a stream-informed delta compression method to avoid replication of duplicate part when backuping datasets to repository over wide area network [13]. While we will discuss in detail in section 2, this approach requires both source and destination hosts to preserve common chunks with the same base fingerprints. Paper [14] applied similar approach to improve compression ratio in backup system by delta compressing similar chunks.

This paper is primarily motivated by the observation that two consecutive versions of a file often differ slightly from each other. Users edit their documents, and the synchronizer sends them to remote server every time interval. While chunking and CBH are applied to explore most of the redundancy which in the form of chunks share exactly the same hash, there is still innegligible duplicate proportion in the left chunks, especially those adjacent to duplicate ones with logical offset in file. If the receiver could find similar chunks for non-duplicate one and then send them back to the sender as delta reference, the data volume the sender transmitted and the total transfer time will be decreased, because the bandwidth from the receiver to the sender is larger than the reverse direction.

The rest of the paper is organized as follows. Firstly, the related works about data deduplication and improvements in distributed scenarios are introduced in section 2. A new delta reference algorithm based on CBH and similar file matching is proposed in section 3. After that, experiments and performance evaluation are presented in section 4. Section 5 discusses relative issues in data synchronization. The paper is concluded in section 6.

2. RELATED WORKS

Tridgell proposed a rsync algorithm in 1996 to avoid exchanging redundant part of files between source and destination hosts [5]. Rsync splits file on the recipient into a series of non-overlapping, contiguous, fixed-sized chunks, then computes a weak checksum and a strong checksum for each chunk and send them to the sender. The sender searches all chunks with the same size of F' to find that have the same weak and strong checksums. Though weak checksums of chunks at any offset can be easily got in a "rolling" manner, fixed-size chunking is very vulnerable to insert/delete modification because of the shifting offset problem. The sender must maintain checksums of chunks at all offsets. It also takes more time to search though file F if checksum of a chunk in F' is absent in it. Despite the extra overhead put on sender, the strategy to find possible similar file simply depends upon file name matching.

Deduplication strategy proposed in low-bandwidth network file system (LBFS) [4] employed Rabin fingerprint to set chunk boundaries based on file contents, which is called content defined chunking (CDC) and it is insensible to modification pattern. Only a cryptographic hash function, SHA-1 for example, is used to calculate a near-unique identity for each chunk with negligible collision probability. Though all RPC traffic is compressed using conventional gzip compression, resemblance between files are not totally reflected by those duplicate chunks.

These studies applied compare-by-hash method to avoid transmitting duplicate chunks with appropriate metadata overhead (a small amount of checksums and chunk offset description fields). The generalized protocol of such a compare-by-hash distributed deduplication scheme can be interpreted as Figure 1.

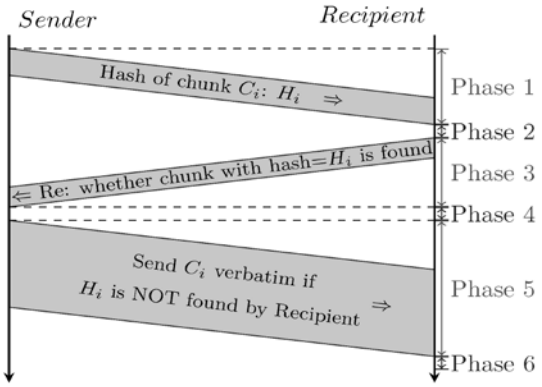

Figure 1: Protocol Of Compare-By-Hash Distributed Deduplication

- Phase 1, the sender divides the file to be send F into a sequence of non-overlapping, consecutive chunks, $C_1, C_2, \cdots, C_n$, and

Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

calculates a corresponding hash H_i for each chunk, then sends these hashes to the recipient.

- Phase 2, once the recipient received H_i, finds it in the chunk hash database.
- Phase 3, the recipient replies the sender with the searching result of H_i : HASH_FOUND if found, HASH_NOT_FOUND otherwise.

Phase 4, prepare the literal data of chunk C_i if the reply for H_i is HASH_NOT_FOUND.

Phase 5, depending on reply of the recipient, the sender sends an index instruction for chunk C_i in file F if HASH_FOUND is replied, otherwise, sends chunk C_i verbatim.

Phase 6, the recipient follows instruction to construct file F in place.

Barreto et al. proposed a technique called hash challenges (HC) to reduce metadata overhead when sending data across network using compare-by-hash protocol [15]. This technique leverages the fact that the number of chunks on the recipient is relatively smaller than the range the full hash expand, thus a hash prefix with less bits are sent to the recipient and the recipient would still efficiently inform the sender whether the corresponding chunk should be upload verbatim. Though benefit is got from the shifting of a small part of meta-data to the direction with higher bandwidth, the larger part of non-duplicate chunks follows the regular routine yet.

In 2012, Shilane et al. presented a combined approach to decrease replication of similar chunks when hash cannot found in remote repository [13]. For those non-duplicate chunks, the backup server (sender as refer previously) sends sketches to remote repository. The sketches of chunks act as resemblance hashes which have the property that similar chunks will have identical sketches. If sketch matches a stream-informed cache, the repository responds with the fingerprint corresponding to the similar chunk, called base fingerprint. The backup server will delta compress the non-duplicate chunk relative to the base chunk before transmitting if the base fingerprint found in local database. While it is ordinary for the backup server to preserve a large collection of old versions of files, this scheme will impose additional storage requirements on smart terminals.

3. THE PROPOSED SCHEME

We proposed a scheme, called delta reference, to reduce the data transferred by the uploading stream with low bandwidth. As depicted in Figure 2, protocol of delta reference differs from the compare-by-hash distributed deduplication in the follow phases.

- Phase 1, instead of single Hi, the sender sends pair $< H_i, L_i >$ to the recipient, in which L_i stands for the length of chunk C_i.
- Phase 2, the recipient searches H_i in its hash database. If HASH_NOT_FOUND, try its best to find reference chunk(s).
- Phase 3, the recipient replies with HASH_FOUND if H_i found in database, FOUND_REF plus reference data if several reference chunks found, or HASH_NOT_FOUND otherwise.
- Phase 4, if HASH_NOT_FOUND replied from the recipient, the sender prepares the literal data of chunk C_i. Otherwise, in case of FOUND_REF, it delta encodes C_i with the reference data, the output called D_i.
- Phase 5, the sender transmits either C_i or D_i literally to the recipient with construct instruction.
- Phase 6, once instruction received, the recipient follows it to construct part of file F locally.

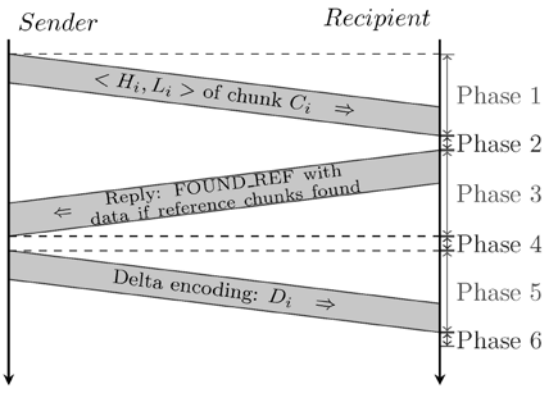

Figure 2: Protocol Of Delta Reference Deduplication

Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.1 Analysis

The new delta reference protocol acts like the generic compare-by-hash distributed deduplication, except for several alternations. We mark the time needed by each phase in Figure 1 as $t_1, t_2, \cdots, t_6$, times in Figure 2 as $t'_1, t'_2, \cdots, t'_6$, respectively. Total time to "transfer" a chunk follows: $T = t_1 + t_2 + t_3 + t_4 + t_5 + t_6$ or $T' = t'_1 + t'_2 + t'_3 + t'_4 + t'_5 + t'_6$.

In the new delta reference protocol, time required by separate phase changes as follows. Phase 1 performs exactly the same action as plain compare-by-hash, thus $t'_1 - t_1 = 0$.

Phase 2 in delta protocol performs an additional search for reference chunks. Though it may depend on information of successive chunks to make a better decision, we just rely on what the application can get from the operating system API immediately. That is, the search algorithm never blocks itself to try to make a better decision. So additional time needed is at which the search procedure executes, we mark it as $t_s = t'_2 - t_2$.

In Phase 3, the recipient needs more time to transmit reference chunks, since we never reply the sender with data more than the size of chunk (marked as cs) the sender sent, thus $t'_3 - t_3 = cs / B_d$, given the download bandwidth as B_d.

In Phase 4, the sender requires more time to carry out delta encoding, so $t'_4 - t_4 = t_{enc}$.

Given the upload bandwidth as B_u and size of delta output in Phase 4 as ds, $t_5 = cs / B_u$, $t'_5 = ds / B_u$.

The recipient requires more time to decode the delta encoding in Phase 6, so $t'_6 - t_6 = t_{dec}$.

Assuming the search procedure finds reference chunks in probability P_s for those chunks absent in database, the difference of time consumed can be got by:

$$T' - T = t_s + P_s\left(\frac{cs}{B_d} + t_{enc} + \frac{ds}{B_u} - \frac{cs}{B_u} + t_{dec}\right)$$

It takes less than 1ms to finish a round of reference chunks search procedure on CPU whose frequency no less than 2GHz. Depending on the similarity between the two input data of delta encoding, the output varies from 0% to about 65% [16]. As tested on a 2GHz CPU machine, delta encoding of two 4KB chunks would be finished in 2ms and decoding finished in 100 μs on average. Figure 3 gives the time retrenchment credits to the new protocol when cs =4KB, t_s =1ms, P_s =0.5, t_{enc} =2ms, t_{dec} =0.1ms, ds =30%cs, the upload bandwidth B_u =1Mbps. Time saved in each round reaches from 5ms to 9ms.

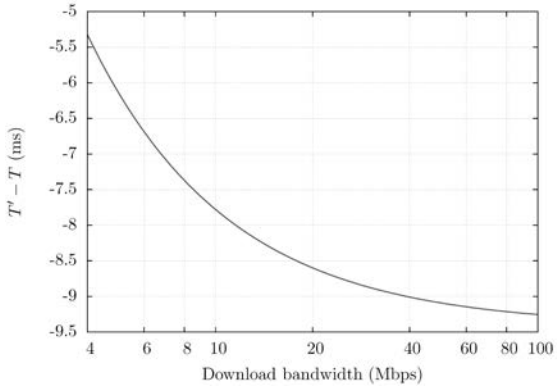

Figure 3: Time Retrenchment Of The New Protocol

3.2 Finding Similar Chunks

The primary conception of the proposed scheme to reduce the total transfer time lies in the decreased uploading volume in a low upload bandwidth. It is important for the recipient to find really similar chunks and reply to the sender. The output of delta encoding should not be larger than the chunk to be send, or all we done will be arduous but fruitless.

As demonstrated by Douglis' study [16], the more the similarity, the less the delta output. However, when file is split into chunks, a large proportion of similarity is reflected by matching chunks. Since duplicate chunks appear in roughly the same stream-ordered patterns [17] and chunks in neighbourhood exhibit similarity [13], this characteristic can be leveraged for dynamic similar chunk finding. Unlike stream-informed sketches, however, similar chunks finding strategy here just searches around matching chunks for low time complexity. The strategy is described in Algorithm 1 and depicted in Figure 4. The sender transmits information of chunks in the order which they are arranged in the logic space of file. The recipient

Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

does not get a global landscape until the last chunk reached, it makes a decision based on matching chunks and offset of the received chunk refer to them. λ controls the ratio of the size of received chunk to that of those reference chunks, 0.7~0.9 will be appropriate to prevent the size of reference chunks from growing too large.

Algorithm 1 Reference chunks searching

Require: H_i – hash of chunk C_i in file F, H_i cannot be found in hash database

Ensure: Reference chunks of C_i : $R(C_i)$

1. $R(C_i)$=Ø
2. **if** reference file of F found, marked as F' **then**
3. $j \leftarrow$current reference index of F'
4. $od \leftarrow$offset difference since the last matching chunk
5. **for** each $k >= j$ **do**
6. **if** $\left|\left[0, len\left(C(F')[k]\right)\right] \cap \left[od, od + len\left(C_i\right)\right]\right| > \lambda * len\left(C(F')[k]\right)$ **then**
7. $R(C_i) = R(C_i) \cup \{C(F')[k]\}$
8. $od = od + len\left(C(F')[k]\right)$
9. current reference index of $F' \leftarrow k$
10. $k = k + 1$
11. **else**
12. **break**
13. **end if**
14. **end for**
15. $od = od - len\left(C_i\right)$
16. **end if**
17. **return** $R(C_i)$

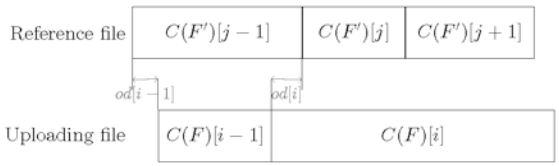

Figure 4: Schema of Algorithm 1

There have been many studies on similar file finding, ether computed hashes of overlapping sequences of bytes [18], or extracted super-fingerprint from calculated fingerprints [19]. Since the recipient would get hashes of chunks in stream order, the hashes of duplicate chunks at hand are regarded as "shingles" to find similar files share several common chunks regardless their arrangement. **Algorithm 2** illustrates the details.

Algorithm 2 Reference file searching

Require: H_i – hash of chunk C_i in file F, stream $S(F)$ containing successive hashes of chunks in F.

Ensure: A file F' on recipient which most similar to F

1. **if** NULL != F' **then**
2. **return** F'
3. **end if**
4. **if** H_i found in database **then**
5. Search forward in $S(F)$ to find more hash that can be found in database: $H_j, H_k, \cdots$
6. Get the file set FS, the files in it referenced by chunks in F, whose hash can be found in database, at least once
7. $F' \leftarrow$ element in FS with the largest reference number
8. $od = 0$
9. current reference index of $F' \leftarrow$ chunk index in F' with hash H_i
10. **else**
11. **return** NULL
12. **end if**

3.3 Delta Encoding

Studied by [16], the output size seems not sensitive to the delta algorithm. Therefore an open source implementation of the widely used vcdiff algorithm–xdelta is chosen for its portability. On the other hand, we found that it is not worth to reach higher compression ratio at the cost of much more encoding time, the lowest 0 level is selected.

4. PERFORMANCE EVALUATION

To evaluate the performance of the delta reference algorithm, we implemented a prototype of file synchronizer using the proposed scheme. Experiments are designed to explore: (1) *how much upload volume can delta reference save?* (2) *how much speed can delta reference improve in a really network?*.

First, the recipient parses an old version of file set, divides each file into chunks based on content defined boundaries. Rabin fingerprint [20] with window size 48 is used, the same as LBFS. Expected chunk size (ecs) varies from 256B to

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

32KB geometrically. The recipient maintains a chunk list for each file in the set, and maintains a reference file list for each chunk. Files and chunks are both indexed by their MD5 signatures.

Second, once connected to the recipient, the sender divides files in the new dataset it hold into chunks based on the same parameters got from the recipient. There are three threads execute concurrently at the sender side: one is in charge of chunking, another is responsible for sending hash signatures (or 3 bytes hash prefix in hash challenges) of chunks, and the last replies the recipient with construct instructions and literal chunks (or delta output). At the recipient side, two threads work together: one receives hashes, looks up in the hash database, then replies to the sender, the other collects instructions and data from the sender, and construct files in place.

The recipient runs in a machine with 2 Intel® Xeon® processor E5504 (2.00Ghz, 4MB L3 Cache) and 4GB RAM. The sender runs in a machine with Intel® Pentium® dual-core processor E5300 (2.60GHz, 2MB L3 Cache) and 2GB RAM. These two Linux hosts are connected by a 100Mbps Ethernet network, with RTT=0.277ms. When necessary, traffic control tool *tc* on Linux are used to limit the transfer speed.

As for datasets, we test two real datasets: gcc-src and linux-kernel-src. These datasets have been used by other state-of-the-art deduplication protocols [21, 22]. In gcc-src, the snapshots correspond to the compile's source code trees of versions of 4.6.0 (old) and the latest 4.7.2 (new). The two versions of linux-kernel-src corresponding to 3.5 (old) and the latest 3.7 (new) Linux kernel source code trees. The recipient holds old version already, and the sender sends the new version. Table 1 summarizes characteristics of these datasets.

It is worth noting that, vcdiff does not only compress one object relative to another, it also compresses a single object based on the information extracted internal. So a control group is examined additional to inspect how delta reference performs precluding the internal compression. The control test, we called Delta Zero, follows exactly the same process to find similar file, reference chunks, except that the recipient call memset() to zero all the data just before replying to the sender. The proposed algorithm in this paper is represented by Delta Ref.

Table 1: Characteristics Of Test Datasets

	old version			new version			duplication ratio			
	total	average	median	total	average	median	256	1K	4K	16K
linux-kernel-src	427MB	11465B	4276B	444MB	11394B	4180B	86%	78%	66%	53%
gcc-src	409MB	6035B	1101B	466MB	6120B	1057B	77%	71%	63%	56%

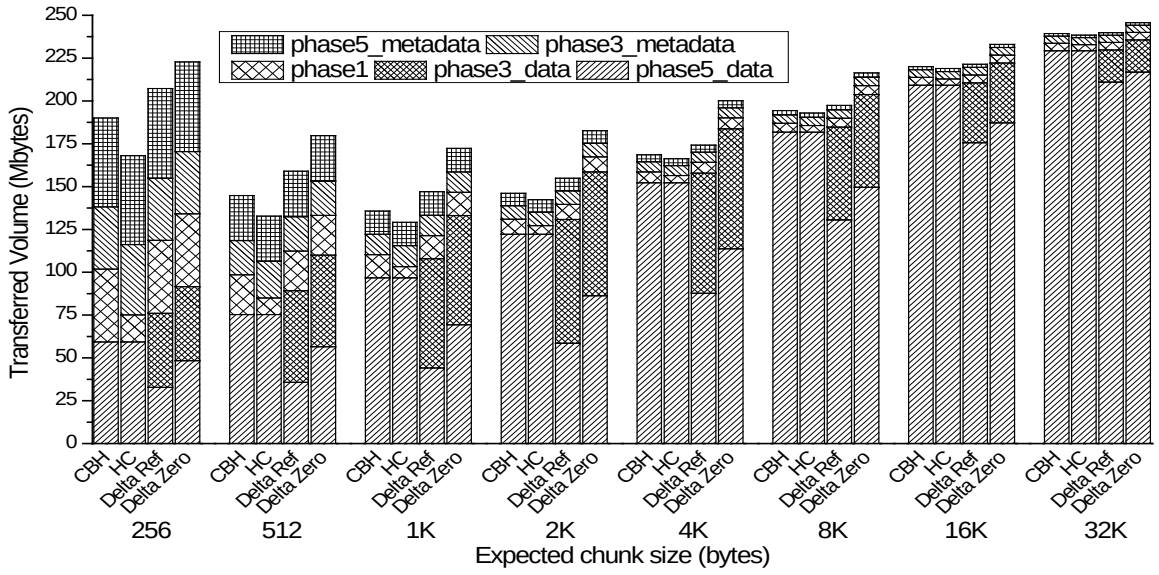

Figure 5: Volume Transferred At Each Phase For Different Expected Chunk Size On The Linux-Kernel-Src Dataset

Journal of Theoretical and Applied Information Technology
10th July 2013. Vol. 53 No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 5 illustrates transferred volume at each phase for different expected chunk size on the linux-kernel-src dataset: phase1 denotes hash signatures (or hash prefix in hash challenges), chunk length and file names transmitted by the sender in Phase 1, phase3_metadata marks the search result of hashes sent by the recipient in Phase 3, phase3_data represents volume of all reference chunks, phase5_data shows chunk data volume sent by the sender in Phase 5, and at last, phase5_metadata summates the meta-data of construct instructions. Totally up to 3 408 800 bytes of file names are included in phase3_metadata and phase5_metadata. Moreover, phase3_data, phase3_data and phase5_metadata may vary among each run depending upon how many bytes the recipient can get from the receive buffer immediately, they are average numbers of five runs. However, the variation does not exceed 0.3%.

As illustrated earlier in Figure 5, hash challenges does reduce volume of phase1 dramatically, from 17.6% (ecs=32K) to 62.8% (ecs=256), however, this retrenchment accounts for only 0.3% to 17.4% of those volume the sender totally transmitted to the recipient. The small the duplication ratio, the small the retrenchment percent. Comparing to CBH and hash challenges, delta reference introduces more transfer volume both the peers transmitted, for ecs=1KB (the choice of ecs that minimizes the volume transfers), 8.3% more than CBH and 13% more than hash challenges. Because of delta compression based on internal information, Delta Zero reduces volume of phase5_data by 28.3% compared to CBH and hash challenges. Furthermore, delta reference compresses relatively to reference chunks provided by the recipient, reducing volume of phase5_data up to 36.4% compared to Delta Zero. This retrenchment accounts for 28.3% of the total volume the sender transfers.

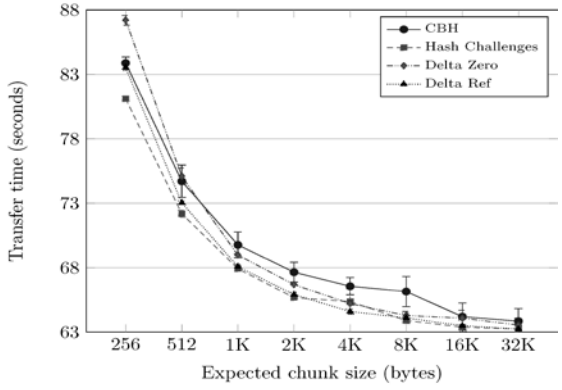

Figure 6: Synchronize Time Of Linux-Kernel-Src In Symmetric100mbps Network

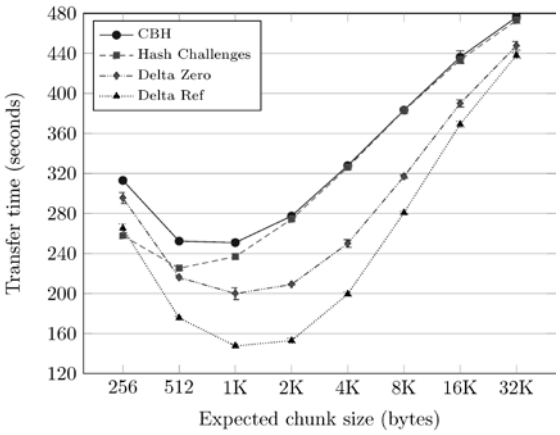

Figure 7: Synchronize Time Of Linux-Kernel-Src In 4Mbps Upload, 100Mbps Download Network

We now pay attention to the performance of each algorithm in real workloads. Figure 6 presents the average speed of five runs each in symmetric 100Mbps network. Transfer time of Delta Ref and Delta Zero do not increase significantly under 0.90 confidence level in spite of slightly more total volume. Figure 7 demonstrates the substantial performance gains of Delta Ref in scenario where the uplink as limited bandwidth. We applied a 4Mbps filter to the sender-to-recipient link to simulate a common home broadband internet connection. The bandwidth of recipient-to-sender remains 100Mbps. In order to complete the synchronization of linux-kernel-src, Delta Ref and Delta Zero take significantly less time, despite the raised total transfer volume. Considering the best ecs choice 1KB that minimizes the total transfer volume, hash challenges is 5.6% faster than CBH, whereas Delta Ref is 26.2% faster than Delta Zero precluding those 20.3% gained from internal compression.

Figure 8 shows improvement pattern of gcc-src similar to Figure 7. Hash challenges improves by 3% than CBH, Delta Ref speeds up by 16.2% than Delta Zero precluding those 20.3% contribute to internal compression.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

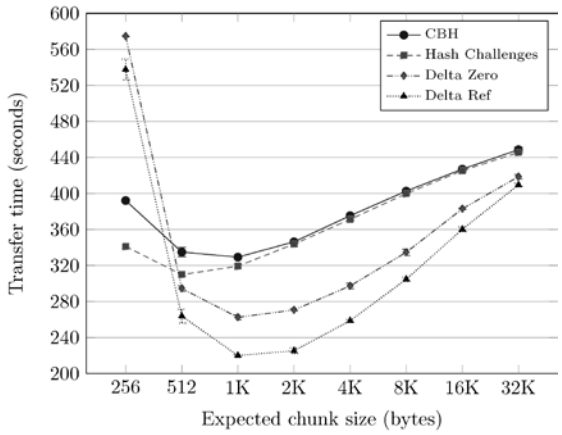

Figure 8: Synchronize Time Of Gcc-Src In 4Mbps Upload, 100Mbps Download Network

Figure 5~Figure 8 all together demonstrate how the expected chunk size (ecs) affects the behaviours of these distributed deduplication solutions. Smaller ecs leads to larger number of chunks, more meta-data and fewer literal chunks to transfer, whereas may produce larger total volume (eg. ecs=256) thereby long transfer time. Larger ecs leads to larger number of chunks, less meta-data and more literal chunks to transfer. With ecs=1KB, total volume reaches its trough and results in least time (except hash challenges with ecs=512).

As for improvement of Delta Ref, smaller ecs explores almost all duplication in the form of matching chunks, leaving less room to further improve. On the other hand, larger ecs puts off the first appearance of matching chunk, leaving those chunks in front of it have no choice but transmitting verbatim. Furthermore, fewer duplicate chunks may increase the likelihood of false candidates in similar file finding procedure.

5. DISCUSSION

Section 4 demonstrates the substantial improvements introduced by the proposed delta reference scheme. However, it seems superfluous given that tar balls of linux-kernel-src and gcc-src are only about 80MB, it takes less than 10 seconds to send such a tar ball over a 100Mbps connection. Whereas, more than 60 seconds are taken to synchronize such a decompressed source tree in the same network as Figure 6 presented. Actually, it takes about 46 seconds to fingerprint and chunk all files in linux-kernel-source. In this scenario, CPU resources but not connection speed becomes the bottleneck. Besides, three phases over network, instead of single phase when sending directly, need more time to accomplish the mission.

As many studies about deduplication protocol did [21, 22], different versions of decompressed linux kernel source code and gcc source code are chosen to inspect performance of deduplication protocol when applied to similar workloads, in which files differ slightly between successive versions. As refer to compress/decompress methods which are widely used to reduce transfer time over low bandwidth network, it takes substantial time to compress and decompress files. In fact, compression and decompression of gcc-4.7.2 source code tree using the fastest zip/unzip take 93 seconds and 8.4 seconds, respectively. On the other hand, not all file types can get such a low compression ratio as source codes [23]. When uplink bandwidth was limited to 4Mbps, delta reference takes much less time to transfer gcc-src than compress/decompress routine (220 seconds at ecs=1KB versus 93+8.4+247 seconds).

6. CONCLUSIONS

The compare-by-hash (CBH) scheme for distributed data deduplication transmits literal data for those chunks whose hash can not be found on remote host. While traditional delta deduplication approaches concentrate on redundancy eliminating locally, they introduce more space demand and application specific limits on storage system. We propose delta reference, a novel algorithm that leverages plain compare-by-hash solutions to find reference chunks to the sender thus substantially reduce the total volume it transmits. Formula analysis and experiments on real datasets demonstrate that delta reference can significantly decrease total transfer time: as much as 26.2% in a typical asymmetric broadband connection, despite considering the internal delta compression.

We would like to study in greater depth about how delta reference performs comparing against related deduplication protocols, if all traffic is compressed using conventional lossless data compression algorithms, which is an obvious approach in deployed systems.

7. ACKNOWLEDGEMENT

This work was supported in part by grants from the Key Program of the Chinese Academy of Sciences (No. KGZD-EW-103-1 and No. KGZD-EW-103-2), the National High Technology Research and Development Program of China (863 Program) (No. 2011AA01A102), and the National Key Technology Research and Development Program of China (No. 2011BAH16B03).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES:

[1] I. T. Union, "G.992.1: Asymmetric digital subscriber line (ADSL) transceivers", *Transmission Systems and Media, Digital Systems and Networks*, G.992.1(06/99) 1, 1999, pp. 1-256.

[2] H. Hussain, D. Kehl, B. Lennett *et al.*, "The Cost of Connectivity", New America Foundation's Open Technology Institute, 2012.

[3] N. Mandagere, P. Zhou, M. A. Smith *et al.*, "Demystifying data deduplication", *Proceedings of the ACM/IFIP/USENIX Middleware'08 Conference Companion*, 2008. pp. 12-17.

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

[5] A. Tridgell, and P. Mackerras, "The rsync algorithm", 1996.

[6] D. Meister, and A. Brinkmann, "Multi-level comparison of data deduplication in a backup scenario", *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, 2009. pp. 1-12.

[7] P. Mukherjee, "A fully Decentralized, Peer-to-Peer Based Version Control System", Technischen Universität Darmstadt, 2011.

[8] A. Seering, P. Cudre-Mauroux, S. Madden *et al.*, "Efficient versioning for scientific array databases", *Proceedings of 2012 IEEE 28th International Conference on Data Engineering (ICDE)*, 2012. pp. 1013-1024.

[9] R. C. Burns, and D. D. Long, "Efficient distributed backup with delta compression", *Proceedings of the fifth workshop on I/O in parallel and distributed systems*. pp. 27-36.

[10] Y. Chen, Z. Qu, Z. Zhang *et al.*, "Data redundancy and compression methods for a disk-based network backup system", in International Conference on Information Technology: Coding and Computing, 2004, 2004, pp. 778-785.

[11] L. L. You, K. T. Pollack, D. D. Long *et al.*, "PRESIDIO: a framework for efficient archival data storage", *ACM Transactions on Storage (TOS)*, vol. 7, no. 2, 2011, pp. 1-60.

[12] J. MacDonald, "File system support for delta compression", Department of Electrical Engineering and Computer Science, University of California at Berkeley, 2000.

[13] P. Shilane, M. Huang, G. Wallace *et al.*, "WAN-optimized replication of backup datasets using stream-informed delta compression", *ACM Transactions on Storage (TOS)*, vol. 8, no. 4, 2012, pp. 13.

[14] P. Shilane, G. Wallace, M. Huang *et al.*, "Delta compressed and deduplicated storage using stream-informed locality", *Proceedings of 4th USENIX conference on Hot Topics in Storage and File Systems*, 2012. pp. 1-5.

[15] J. Barreto, L. Veiga, and P. Ferreira, "Hash challenges: Stretching the limits of compare-by-hash in distributed data deduplication", *Information Processing Letters*, vol. 112, no. 10, 2012, pp. 380-385.

[16] F. Douglis, and A. Iyengar, "Application-specific delta-encoding via resemblance detection", *Proceedings of 2003 USENIX Annual Technical Conference*. pp. 113-126.

[17] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system", *Proceedings of FAST '08: 6th USENIX Conference on File and Storage Technologies*. pp. 269-282.

[18] U. Manber, "Finding similar files in a large file system", *Proceedings of the USENIX winter 1994 technical conference*, 1994. pp. 1-10.

[19] A. Z. Broder, "On the resemblance and containment of documents", *Proceedings of Compression and Complexity of Sequences*, 1997. pp. 21-29.

[20] M. O. Rabin, *Fingerprinting by random polynomials*: Center for Research in Computing Technology, Aiken Computation Laboratory, The Hebrew University of Jerusalem, 1981.

[21] N. Jain, M. Dahlin, and R. Tewari, "Taper: Tiered approach for eliminating redundancy in replica synchronization", *Proceedings of FAST '05: 4th conference on USENIX Conference on File and Storage Technologies*, 2005. pp. 281-294.

[22] D. R. Bobbarjung, S. Jagannathan, and C. Dubnicki, "Improving duplicate elimination in storage systems", *ACM Transactions on Storage (TOS)*, vol. 2, no. 4, 2006, pp. 424-448.

[23] W. Bergmans. "Summary of all single file-type lossless data compression tests", 10-08, 2012; http://www.maximumcompression.com/data/summary_sf.php.