ACM SRC Entry: Himabindu Pucha (hpucha@purdue.edu, hpucha@cs.cmu.edu)
Research Advisers: David G. Andersen, Michael Kaminsky and Y. Charlie Hu
Category: Graduate, First Place in Grace Hopper Conference 2007

Exploiting Similarity for Multi-Source Downloads Using File Handprints

1. Summary

Many contemporary approaches for speeding up large file transfers (e.g., BitTorrent) download chunks of a data object from multiple sources with an exact copy of the desired object. However, both our study and previous work demonstrate that such multi-source transfers are slow. Our work proposes that sources that serve similar but non-identical objects can be leveraged to improve the performance of these systems. Further, we present handprinting, a technique that locates these identical and similar sources for data objects efficiently using a constant number of lookups and inserting a constant number of mappings per object into a global lookup table.

2. Problem and Motivation

Bulk data transfers represent more than 70% [2] of Internet traffic. Given their importance, extensive research over the past several decades has explored many techniques to improve data transfer speed and efficiency. Despite this effort, bulk data transfers often remain slow because: (1) receivers may be bandwidth-limited; (2) the source or sources may be unable to saturate the receiver's bandwidth; (3) congestion or failures in the ``middle'' of the network may slow the transfer.

Downloading data from multiple sources simultaneously is a popular technique to speed transfers when the receiver is not the bottleneck. Many peer-to-peer content distribution systems use this idea (e.g., BitTorrent, Gnutella): Suppose that a receiver wants to download file A. Typically, in these systems, file A is broken into several pieces/chunks (e.g., 16KB each). The receiver now obtains different sets of chunks from different identical sources (sources of file A), in parallel, thereby increasing its throughput. Unfortunately, both our measurement study and previous studies [5] show that the performance of transfers in spite of using these multi-source systems is often unacceptably slow, with users requiring hours or even days to download content. For example, Figure 1 shows the CDF of throughput achieved while downloading 6208 large files from popular multi-source file-sharing networks. The median transfer achieved under 10 Kbit/s of average throughput, and the 90th percentile only managed 50 Kbit/s, despite running these experiments from an extremely well-connected academic network. Thus, available identical sources cannot saturate receivers in current multi-source systems.

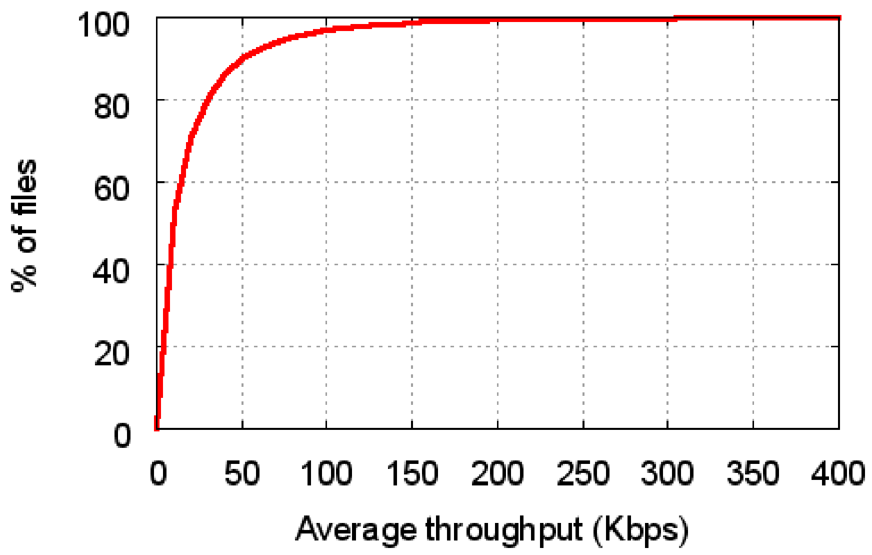

Figure 1: Throughput observed in 6208 file downloads from file-sharing networks.

Our work proposes Similarity-Enhanced Transfer (SET), a system that speeds up bulk data transfers by discovering **additional similar sources** for a file (sources that provide only some chunks of file A to the receiver). Moreover, SET needs to discover these sources with low overhead.

3. Related work

Systems such as CFS [4] and Shark [1] exploit similarity at the chunk level, similar to SET's goal. In such per-chunk systems, receivers locate sources for individual chunks of the desired file. The cost incurred here is to create a mapping between every unique chunk in the identical and similar files and their corresponding sources, i.e., O(N) mappings per object (where the object has N chunks) in some directory service. The receiver then performs O(N) lookups in such a directory service, one for each of the N chunks in the file being downloaded. SET significantly reduces the overhead of this approach by using only a small, constant number of lookups and constant number of mappings.

4. Approach and Uniqueness

Our approach in designing SET to exploit both identical and similar sources can be broken down into the following steps:

First, to determine whether similarity exists among files popular in today's file sharing networks (such as ed2k, edonkey), we conduct a detailed similarity analysis of **1.7TB** of data fetched from these networks. These data represent a variety of file types, with an emphasis on multimedia files. This is a unique contribution of our work as multimedia files were often ignored by previous similarity studies. Apart from measuring chunk-level similarity in these files, we also propose a metric, parallelism, to capture the improvement in download time by exploiting this similarity.

We then propose and analyze a novel technique to efficiently exploit this similarity, called **handprinting**. Handprinting first divides each file into a series of chunks, C1 ... CN, using Rabin fingerprinting (see [6]). Next, it computes the hash (fingerprint) of each chunk, and takes a deterministic sample of these fingerprints. To sample, handprinting sorts the hashes in lexicographic order, selects the first k hashes (the handprint), and inserts them into a global lookup table. To find similar files, a receiver obtains the chunk hashes for its desired file and searches for entries for any of the first k hashes. The algorithm detects similar files if they collide on any of the k chunks. This process is illustrated in Figure 2.

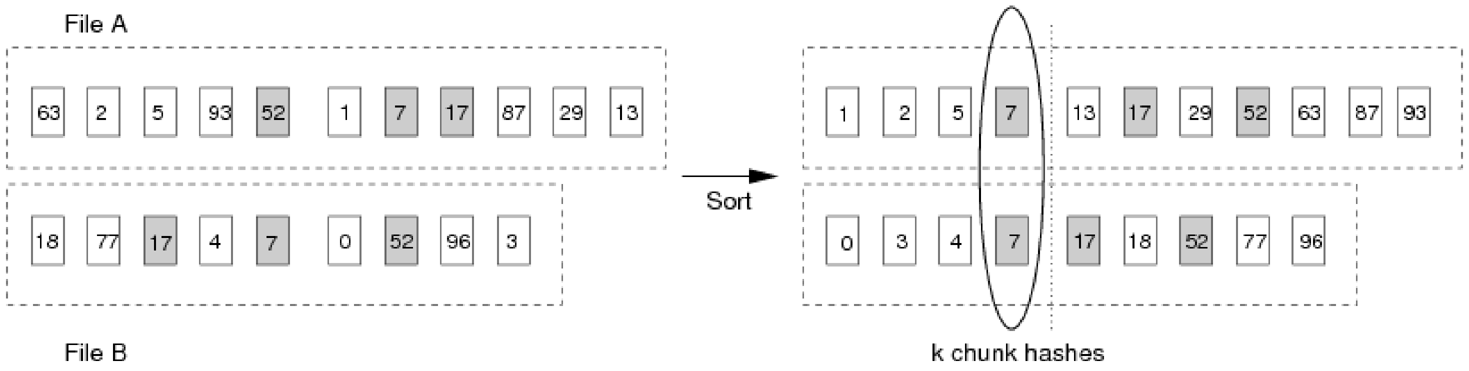

Figure 2: Handprinting. By sorting the chunk hashes before insertion, handprinting imposes a consistent order on hashes so that if a set of one or more shared (shaded) hashes are chosen from each file, the sets will intersect.

Handprinting substantially improves the probability of finding similar files versus a naive random sampling of chunks. In the limit, consider a scheme that selects one chunk at random from each file. The probability that this scheme will work is the probability of selecting a shared chunk from the first file (m/N) and also selecting the same chunk from the second file ($1/N$), or m/N^2. If the files share 10 out of 100 chunks, the probability of detecting them using the random scheme is 0.001; using handprints, the probability is 0.01, an order of magnitude better.

We also analyzed the probability of similarity detection p_d using handprinting to determine a suitable value of k (the number of fingerprints in a handprint) [Details in [8]]. Intuitively, p_d is a function of the extent of similarity between files (s) and k. Our analysis indicates that 28 fingerprints are needed to detect files with at least 10% similarity with a probability of at least 90%. Further, a threshold of 10% is satisfactory as observed from our data analysis. Thus, handprinting enables SET to discover **identical and similar** sources using constant (**k**) mappings per object and constant (**k**) lookups. Handprinting is inspired by earlier work on similarity detection, shingling [3]. However, it differs in goal and analysis more than in mechanism: it is used as the basis for a distributed lookup mechanism to detect exploitable similarity, not document resemblance.

We then design the SET system that allows receivers to download objects from multiple sources, which contain either exact or similar objects, using a constant number of lookups. Every object in SET is identified by its Object ID (OID), a collision-resistant hash of the object. To download a file, the receiver must locate the sources for the actual file it wishes to download (*exact sources*) and for the other files that share chunks with the target file (*similar sources*). We assume the receiver already has a complete list of chunk hashes for the desired object. (For example, such hashes are included as part of the BitTorrent “torrent” file.) The receiver uses these hashes to fetch the individual chunks from the sources it discovers. For every object that a source makes available, it inserts a set of mappings into a global lookup table. This lookup table could be a Distributed Hash Table (DHT), or a service run by a single organization such as Napster. The lookup table contains two types of mappings: (1) chunk hash → OID, and (2) OID → source.

Mapping 1 associates the hash of a particular chunk in the object with the object identifier for that object. There are exactly k mappings that point to each OID. These mappings allow receivers to locate objects that are similar to the one they want to download. Mapping 2 associates the OID of an object with the sources for that object. The lookup table contains one such mapping for each source that serves the corresponding object. If three sources serve OID x, then three OID-to-source mappings will exist in the lookup table (but still only k chunk to OID mappings, since the selected chunks will always be the same for a given OID). The mappings are shown in Figure 3-A. Given the list of chunk hashes, the receiver computes the object's handprint (the k chunk hashes), and looks each hash up in the global lookup table. The result of this query is a list of OIDs that are exploitably similar to the desired file (they share one or more large chunks of data). For each OID, the receiver queries the lookup table to determine the list of potential sources. Finally, the receiver must query one source for each OID to obtain the list of hashes for that OID. At this point, the receiver knows a set of sources that have or are in the process of downloading each of the chunks in the object the receiver wishes to download. This process is illustrated in Figure 3-B.

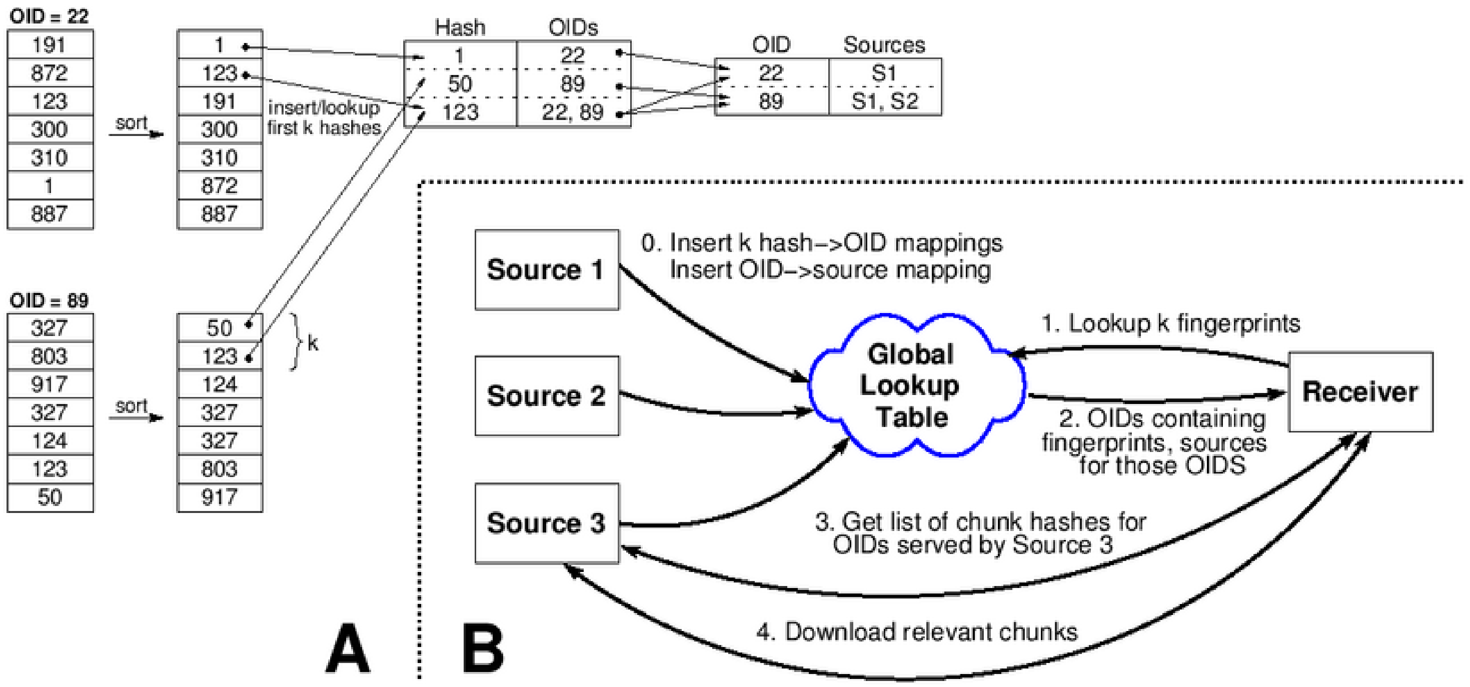

Figure 3: A—Mapping in SET. Mapping 1 links chunk hashes to OIDs; mapping 2 links OIDs to sources of those OIDs. B—SET design overview.

This process requires a constant number of lookups regardless of the file size: *k* queries to the distributed lookup table to determine the similar files, plus 2 × the number of similar OIDs (first to find potential sources for each OID and second to determine the list of hashes for each similar OID). SET limits the number of similar OIDs to a constant (in our implementation, 30), preferring OIDs that matched more chunk hashes if it has an excess number of choices. Given the list of potential sources for each chunk in the desired file, the receiver can begin downloading the chunks in parallel from the sources.

We implemented SET in the DOT [9] framework. SET works by having the sources insert a chunk hash --> object and an object --> source mapping in a global lookup table. The receiver looks up the handprint of the file being downloaded to first learn about similar files and then about the identical and similar sources for those files. At this point, the receiver knows a set of sources for the chunks it wishes to download and completes the download by contacting these sources. We evaluated SET in both an emulated environment (Emulab [10]) and a real world testbed (PlanetLab[7]).

5. Results and Contributions

Significant contributions of our work are:

- Our collected data shows that significant cross-file similarity exists even in highly compressed multimedia files. When we considered files with any non-zero similarity, we found that mp3 files have a median similarity of 99%. One reason for this similarity is two files that differ only in metadata (ID3 tags). Avi files show a median similarity of 20%. This can be for example due to the same movie in different languages. Mpeg files also show a median similarity of 99%. Our data analysis shows that by exploiting this similarity, receivers can locate several times the number of potential sources.

- The second contribution of our work is the handprinting technique to locate identical and similar sources to the file being downloaded using only O(1) lookups and O(1) mappings.

- Finally, our results from emulation and deployment show that:

- SET successfully exploits similarity: For example, in a scenario with a single receiver and sender at 768Kbps(up) and 3Mbps(down) bandwidth, adding three 90% similar sources halves the download time.
- SET scales well with increasing similar sources by incurring negligible overhead (roughly 0.5% per similar file).
- SET performs well for **real** workloads from our measurement study: SET reduces download time by 30% for a trailer download and by 70% for a music download.
- Comparisons with BitTorrent show that without using similar sources SET meets or exceeds BitTorrent's performance.

From our evaluation, we know that SET works well when there are adequate amounts of exploitable similarity and the original senders cannot saturate the receivers' available bandwidth. Based upon prior work and our own analysis [8], we conclude that such similarity exists in a number of important applications: large software updates (e.g., synchronizing a large collection of machines that may start with different versions of the software), transmitting large email attachments, and downloading multimedia content, among others. This makes SET a useful integral part of future data transfer systems.

The complete version of this work appears in USENIX NSDI 2007 [8]. I have also designed and implemented more complex data transfer systems that build upon SET functionality [11]. **This work has been featured in an** article on slashdot **and on** BBC**. Other press coverage includes** MIT Technology Review, ars technica and a Wiki entry.

6. References

[1] S. Annapureddy, M. J. Freedman, and D. Mazieres. Shark: Scaling file servers via cooperative caching. In Proc. 2nd USENIX NSDI, Boston, MA, May 2005.
[2] N. B. Azzouna and F. Guillemin. Analysis of ADSL traffic on an ip backbone link. In Proc. IEEE Conference on Global Communications (GlobeCom), San Francisco, CA, Dec. 2003.
[3] A. Broder, S. Glassman, M. Manasse, and G. Zweig. Syntactic clustering of the web. In Proceedings of the 6th International WWW Conference, pages 1157-1166, Santa Clara, CA, Apr. 1997.
[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, Canada, Oct. 2001.
[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In Proc. 19th ACM Symposium on Operating Systems Principles (SOSP), Lake George, NY, Oct. 2003.
[6] A. Muthitacharoen, B. Chen, and D. Mazieres. A low-bandwidth network file system. In Proc. 18th ACM Symposium on Operating Systems Principles (SOSP), Banff, Canada, Oct. 2001.
[7] L. Peterson, T. Anderson, D. Culler, and T. Roscoe. A blueprint for introducing disruptive technology into the Internet. In Proc. 1st ACM Workshop on Hot Topics in Networks (Hotnets-I), Princeton, NJ, Oct. 2002.
[8] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In Proc. 4th USENIX NSDI, Cambridge, MA, Apr. 2007.
[9] N. Tolia, M. Kaminsky, D. G. Andersen, and S. Patil. An architecture for Internet data transfer. In Proc. 3rd Symposium on Networked System Design and Implementation (NSDI), San Jose, CA, May 2006.
[10] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In Proc. 5th USENIX OSDI, pages 255270, Boston, MA, Dec. 2002.
[11] H. Pucha, M. Kaminsky, D. G. Andersen, and M. Kozuch. Adaptive File Transfers for Diverse Environments. In Proc. USENIX, Boston, MA, June 2008.