RC22514 (W0207-042) July 11, 2002
Computer Science

IBM Research Report

Dynamic Suppression of Similarity in the Web: a Case for Deployable Detection Mechanisms

Frederick Douglis, Arun K. Iyengar
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Kiem-Phong Vo
AT&T Labs

Dynamic Suppression of Similarity in the Web: a Case for Deployable Detection Mechanisms

Fred Douglis, IBM Research
Arun Iyengar, IBM Research
Kiem-Phong Vo, AT&T Labs

Abstract

Compression and delta-encoding [18] of web resources is a well-understood, but infrequently used, technology. Two existing proposals [8, 22] to extend delta-encoding beyond successive versions of the same resource suffer from a requirement that the parties participating in the delta-encoding process coordinate knowledge of their caches. We make a case for more general mechanisms, and propose a distributed approach that does not require advance coordination.

1 Introduction

Over the past several years, researchers and ultimately product organizations have recognized the benefits of reducing network traffic through two related technologies, compression and delta-encoding. Compression involves reducing the volume of data by eliminating redundancy internal to the data; delta-encoding involves reducing redundancy relative to another piece of data already known to both the sender and receiver. These two technologies are very similar, and in fact the delta compression[1] technique in [14] combines both differencing and compression. An advantage of this approach is that the benefit of compression is still realized even when the two pieces of data are sufficiently different.

The benefit of encoding data to reduce network requirements depends on the relationship between the bandwidths of computation and networking, in two respects. First, end-to-end transmission time is a function of the performance of the sending and receiving machines, the algorithm used, the compression obtained, and the network bandwidth [9]. Second, economics may dictate a bias, as one pays for bandwidth and for computing resources, and either one must be maintained at some peak level even if underutilized much of the time.

Presently, there are two competing viewpoints with respect to the costs of bandwidth versus computation. The whole "Grid Computing" effort is based on the premise that networking is cheap relative to other aspects of computing, and that distributed access to resources is key to scaling systems to tackle large-scale problems [10]. But large web content providers pay high charges for the high-bandwidth network connections they use to send data throughout the Internet, and Content Distribution Networks (CDNs), which attempt to get web content to end users efficiently and reliably, currently charge content providers based on how much data the CDNs transmit on their behalf. Thus, content providers would presumably welcome methods to reduce network traffic, even at a cost of increased computation.

In this paper we assume that there is an economic incentive, and often a performance incentive, to trading computation for bandwidth through compression and delta-encoding. The basic argument and tradeoffs of a decade ago [9] still apply, but the application space has increased dramatically through new technologies for identifying where redundancy exists [16, 5]. Henceforth,

[1] Since "delta encoding" is the more commonly used term, we use this term throughout the remainder of the paper to mean delta compression as described by *vcdiff* [14].

we ignore simple compression, since it is a well-known and straightforward technology. But existing delta-encoding technology suffers from a lack of applicability when applied only to successive versions of a single resource [18]. Some other approaches [8, 22], which we discuss in the next section, address the lack of applicability but are too restrictive in the requirements they place on the machines performing the encoding. Here, we make a case for an expansion of the HTTP delta-encoding specification [17] to identify possible sources of redundancy dynamically, and generalize the past work to a wider range of environments.

The rest of this paper is organized as follows. The next section discusses past work on delta-encoding and other techniques for eliminating redundant traffic. Section 3 describes our approach, and Section 4 elaborates on some practical considerations. Sections 5 and 6 discuss related and future work, and Section 7 concludes.

2 Background

In 1997, Mogul, et al., analyzed the potential benefits of compression and delta-encoding in the context of HTTP [18]. They found that delta-encoding could dramatically reduce network traffic in cases where a client and server shared a past version of a resource (i.e., a web page), termed a "delta-eligible" response. When a delta was available, it reduced network bandwidth requirements by about an order of magnitude. However, in the traces evaluated in that study, responses were delta-eligible only a small fraction of the time (10% in one trace and 30% in the other, but the one with 30% excluded binary data such as images). On the other hand, most resources were compressible, and they estimated that compressing those resources dynamically would still offer significant savings in bandwidth and end-to-end transfer times—factors of 2-3 improvement in size were typical.

Later, Chan and Woo devised a method to increase the frequency of delta-eligible responses, by comparing resources to other cached resources with similar URLs [8]. Their assumption was that resources "near" each other on a server would have pieces in common, something they then validated experimentally. They also described an algorithm for comparing a file against several other files, rather than the one-on-one comparison typically performed in this context. However, they did not explain how a server would select the particular related resources in practice, assuming that it has no specific knowledge of a client's cache. (We believe there is an implicit assumption that this approach is in fact limited to "personal proxies" with exact knowledge of the client's cache [11, 2], in which case it has limited applicability. We return to this issue below.)

Spring and Weatherall [22] effectively generalized Chan and Woo's work by applying it to all data sent over a specific communication channel, and using work by Manber [16] and Broder [5] to efficiently detect similar, but not identical, documents in a collection of data. Instead of looking in a web cache, and identifying candidate pages by URL, they looked at all transmitted data, and identified candidate pages by similarity of content. However, like Chan and Woo's work, this system worked only with a close coupling between clients and servers, so both sides would know what redundant data existed in the client. In addition, the communication channel approach required a separate cache of packets exchanged in the past, which may compete with the browser cache and other applications for resources.

Bharat, et al. [3, 4] give methods for detecting mirrors (systematic replication of content). Mirrored URLs are likely to have similar content and are good candidates for delta-encoding [12]. The extent of duplication on the web is estimated at 30-40% [7, 21].

Here we discuss some possible new methods for improving the effectiveness of delta-encoding in HTTP, in a method that allows for widespread deployment without the tight cache coupling of earlier approaches. We consider multiple environments:

Client-server, where a server provides content to many clients, and cannot (or will not) maintain exact knowledge of the cache state of the clients. A server may also require that all deltas be relative to the content on the server, since it may not wish to generate deltas against content served by another host. On the other hand, a server might be willing to retrieve data from elsewhere to serve a client with particularly poor connectivity.

Client-proxy, where a proxy serves many clients, and content can come from many sites. There are two subcases:

- The proxy and client are not closely coupled, and cannot rely on shared state to identify similar resources.
- The proxy is a "personal" proxy, where the proxy and client (which might be another proxy on the same machine as the browser) can keep their caches tightly coupled. The Spring and Weatherall network-level system is an example of a pair of tightly coupled endpoints, though it does not integrate with the web caches.

Proxy-server, where a cache acts as a client to an origin server and as a shared proxy to downstream browsers or caches. Proxies may be independent from content providers or closely coupled with them, as is the case with "reverse proxies" and Content Distribution Networks (CDNs). Reverse proxies and CDNs will tend to have greater locality of reference than arbitrary servers or clients, may be more closely coupled with the origin servers to keep information about cache state, and may be more willing than forward proxies to use a special-purpose protocol to improve efficiency.

Figure 1 depicts these scenarios as a Venn diagram.

The difficulties in applying Chan and Woo's approach in cases where client cache state is unknown lead us to propose an alternative protocol for determining which cached versions should be used as a basis for delta-encoding, in cases where the caches are not coupled. Like Spring and Weatherall, we propose to use Broder's shingling approach for efficiently detecting similar documents in a collection [5], but we do so in a distributed fashion in the general case. The next section describes our approach in greater detail.

3 Distributed Redundancy Detection

While Chan and Woo demonstrated that their multi-file delta-encoding approach could reduce network traffic compared to the single-file approach [8], it is not clear how it fits into the proposed standardization of delta-encoding within HTTP [17]. RFC 3229 calls for a client to inform a server about which cached documents it has that the server might wish to compute a delta against. It could be trivially extended to use Chan and Woo's *npact*[2] algorithm, which encodes a document as differences from multiple other documents: the server would inform the client which files were used. However, simply basing the selection on name similarity in the client's cache seems insufficient:

- If there are many resources with similar distance, it may be fine to select any of them, or there may be particular resources that overlap most closely. Chan and Woo found some sites where comparing against other resources from the same site actually fared worse than simple compression using *gzip*.

[2] The *vcdiff* algorithm currently referred to by the delta-encoding RFC, and specified in its own Internet-draft [13], also can support multiple base files.

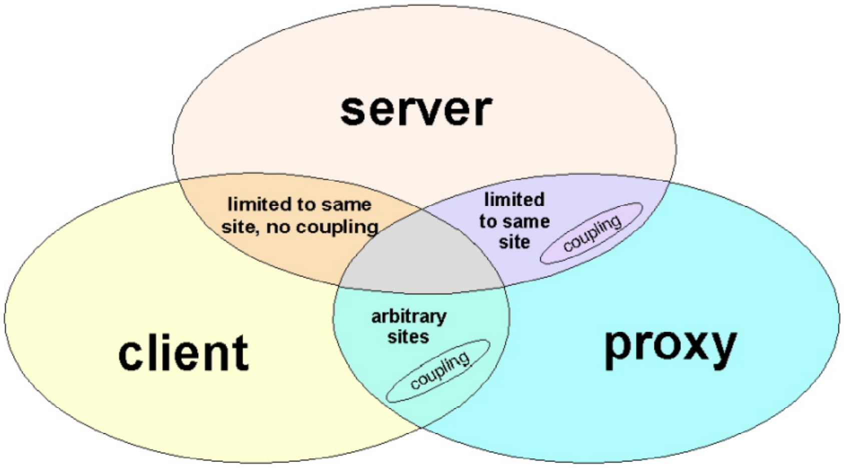

Figure 1: A Venn diagram showing the relationship between clients, servers, and proxies using multi-file delta-encoding.

- The prevalence of aliasing in the web, as found by Kelly and Mogul [12], and the finding by Spring and Weatherall that about 20% of redundancy came from different sites [22], suggest that limiting deltas to resources from the same site may be too restrictive. However, Chan and Woo found that comparing across sites, based simply on URLs, was ineffective. We postulate that comparing across sites by using *hints* about similar content will avoid that problem, though it is as yet unclear whether that ability will dramatically improve the effectiveness of delta-encoding.

- It has often been pointed out that encoding deltas against previous versions of otherwise uncachable resources can improve performance above and beyond traditional web caching [2, 18, 22]. In effect, a cache can retain an uncachable resource for the sole purpose of revalidating or updating it efficiently at a later time, via explicit communicaton with the origin server. However, resources that a server views as uncachable will not be usable by the server for later delta-encoding unless the server too views the resource as useful data to retain. (Note that the Spring and Weatherall approach makes this relationship explicit, by having both sides of the connection retain all data for a considerable time, but this does not scale well to large servers with many clients. Explicit hints between clients and servers about retention of resources may help improve effectiveness, as we describe below.

Given a collection of cached resources, and a server that can generate deltas, the server[3] needs to know which cached resources to encode a new response against. If the server caches all the responses it gives out—something possible for a limited time with a personal proxy, but generally not for a shared server—and it knows that the client retains all those responses as well, it can

[3] We use the term *server* generically here. In the case of a proxy cache, it acts as a server to the client.

arbitrarily decide which base page(s) to use. To do this, it would most likely use a combination of URL names (a new version of the same resource is probably similar to a previous version) and Broder's shingling system.

In the more general case, a server does not have exact knowledge of the client's cache. For large resources, where a delta would make a dramatic difference in network usage and end-to-end response time, sending a sketch of the resource in a way that allows the client to find *similar* resources, rather than only *identical* ones, will allow a client to tell the server how to delta-encode the resource. These sketches can be relatively small, for instance 800 bytes or so [5][4]. Of course, one could send the MD5 hash as well to shortcut the case where an identical resource is already known [12]. In either case, the original resource must be large enough to merit the overhead of sending the meta-information ahead of, or in addition to, the first packets containing the resource itself.

Here is an outline of a protocol for using multiple common source data:

1. The client initiates a request for a resource.
2. The server decides whether to send back the requested data, a sketch of the data, or both (in which case the sketch precedes the first packets, and the transfer can be aborted if a delta is to be sent instead).
3. In the case of a sketch being provided, the client may find a set of cached resources matching the sketch, possibly weighted by other information such as server retention information. It sends the server their URLs and MD5 hashes.
4. The server decides how to use that info to compute a delta-encoded dataset.

4 Practical Considerations

In addition to the issues raised above, some other questions arise. What if the client picks a resource that the server does not have cached? While this is possible, there are ways to reduce the likelihood considerably. One is to use a fixed set of base versions that are known to be cached and used for delta-encoding. The proposed standard for delta-encoding describes a mechanism for servers to give hints about base versions to retain [17], which would be even more applicable in this context. Finally, note that delta-encoding is always just an optimization; if a client asks for a delta against a version the server does not have, it will send the unencoded version.

As client and server cache capacities grow, it may be easier to ensure that both retain all relevant versions for a prolonged time. To the extent that cache replacement is an issue, the use of delta-encoding raises some interesting issues for cache replacement policies: one would factor in not only when a resource may be accessed again directly, and the cost of retrieving it from a server, but also the likelihood of using the resource as a base for other pages. This suggests another axis along the spectrum of nonuniform cache replacement policies [1].

5 Related Work

In addition to the general delta-encoding work described above, there are some other research efforts that bear on this proposal.

[4] If one is interested only in the case where the delta is small (i.e., where two documents are actually 80-90% similar), it is possible to reduce this sketch to only about 50 bytes [6]

Kelly and Mogul recently quantified the extent of data replication (identical copies, rather than similar documents) and proposed an HTTP extension to suppress duplicates [12]. They suggested that a web server could send a signature of a resource, such as an MD5 hash, ahead of the data. The client would compare the MD5 hash against all cached resources, and if found, it would refrain from retrieving the resource from the server. Our similarity detection uses a similar protocol, with the added step of sending MD5 hashes to the server to identify similar resources that are suitable candidates for deltas.

The *rproxy* system [19] is based on the *rsync* system [20] protocol for sending changes to documents without prenegotiating a specific older version. With rproxy, two cooperating proxies would identify a resource for which an older version is cached, and send signatures of data blocks from the client to the server. It is similar to the Spring and Weatherall system in terms of dynamically identifying redundant data, but is more like traditional delta-encoding in limiting itself to a single resource and without requiring that the server know what the client caches.

Wills et al. proposed a mechanism that bundles related resources together, and which can use delta-encoding to update those bundles efficiently [23]. Dynamically identifying past content, either in bundles or individual resources, for efficient encoding of those bundles is a natural extension.

6 Future Directions

Our work in this area has just begun. We realize there are many interesting issues to explore and experiments to perform:

Implementation We are beginning to prototype a system for passing sketches within HTTP, and letting clients specify candidate base versions in requests for pages.

Shingling Broder's shingling experiments looked for textual similarity, and stripped markup [5]. For delta-encoding, the greatest benefit may be from finding repeated markup, with changes to associated text. We will explore variations of the shingling algorithm to address the specifics of this application, to minimize storage and computation requirements on the clients and to best identify content that is suited to delta-encoding.

Parameters The performance of the system will depend on many factors, such as how large a resource should be before using the sketch approach, relative to network bandwidth.

Configurations As mentioned above, delta-encoding can be used in many environments, from personal proxies to shared servers to CDNs. The methods for using similar resources for delta-encoding will vary from configuration to configuration.

Invalidation When notifying a browser, proxy, or CDN that a resource has changed, one might include information about how it has changed. While a general server-initiated callback mechanism may not be able to identify appropriate base versions, client-driven approaches such as piggyback cache invalidation [15] may be easily integrated using the bundling approach plus multi-file delta-encoding.

Encoding protocols The current proposed standard for delta-encoding [17] does not support multiple sources of common data, and needs extension along the lines of the protocol sketched above.

7 Conclusions

Delta-encoding has been actively discussed for several years, but has never really become widespread. We believe that the reason for this is twofold: the relative lack of repeated accesses to identically named resources, compared to different resources with similar content, and the complexity and overheads of having devoted, closely coupled caches on both sides of a link.

Using approximate file matching to identify related content and permit otherwise unrelated clients and servers to negotiate cases where delta-encoding is applicable may open the door to more widespread use and significant reductions in network traffic.

Acknowledgments

Andrei Broder, Chris Howson, Srini Seshan, and John Tracey provided helpful comments on the ideas presented in this paper.

References

[1] Hyokyung Bahn, Sam H. Noh, Sang Lyul Min, and Kern Koh. Efficient replacement of nonuniform objects in web caches. *IEEE Computer*, 35(6):65–73, June 2002.

[2] Gaurav Banga, Fred Douglis, and Michael Rabinovich. Optimistic deltas for WWW latency reduction. In *Proceedings of 1997 USENIX Technical Conference*, pages 289–303, January 1997.

[3] K. Bharat and A. Broder. Mirror, mirror on the web: A study of host pairs with replicated content. In *Proceedings of the 8th International World Wide Web Conference*, pages 501–512, May 1999.

[4] Krishna Bharat, Andrei Z. Broder, Jeffrey Dean, and Monika Rauch Henzinger. A comparison of techniques to find mirrored hosts on the WWW. *Journal of the American Society of Information Science*, 51(12):1114–1122, 2000.

[5] Andrei Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences (SEQUENCES'97)*, 1997.

[6] Andrei Z. Broder. Identifying and filtering near-duplicate documents. In *Combinatorial Pattern Matching, 11th Annual Symposium*, pages 1–10, June 2000.

[7] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. In *Proceedings of the Sixth International World Wide Web Conference*, April 1997. Available as `http://www6.nttlabs.com/HyperNews/get/PAPER205.html`.

[8] Mun Choon Chan and Thomas Y. C. Woo. Cache-based compaction: A new technique for optimizing web transfer. In *Proceedings of Infocom'99*, pages 117–125, 1999.

[9] Fred Douglis. On the role of compression in distributed systems. In *Proceedings of the Fifth ACM SIGOPS European Workshop*, September 1992.

[10] Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, February 2002.

[11] Barron C. Housel and David B. Lindquist. WebExpress: A system for optimizing Web browsing in a wireless environment. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking*, pages 108–116. ACM, November 1996.

[12] Terence Kelly and Jeffrey Mogul. Aliasing on the World Wide Web: Prevalence and Performance Implications. In *Proceedings of the 11th International World Wide Web Conference*, May 2002.

[13] David G. Korn, Joshua P. MacDonald, Jeffrey C. Mogul, and Kiem-Phong Vo. *The VCDIFF Generic Differencing and Compression Data Format*, November 2001. Work in Progress, draft-korn-vcdiff-06.txt.

[14] David G. Korn and Kiem-Phong Vo. Engineering a differencing and compression data format. In *Proceedings of the 2002 Usenix Conference*. USENIX Association, June 2002.

[15] Balachander Krishnamurthy and Craig E. Wills. Study of piggyback cache validation for proxy caches in the World Wide Web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 1–12. USENIX, December 1997.

[16] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 17–21 1994.

[17] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. *Delta encoding in HTTP*, January 2002. RFC 3229.

[18] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 181–194, September 1997.

[19] rproxy. `http://rproxy.samba.org`, 2002.

[20] rsync. `http://rsync.samba.org`, 2002.

[21] N. Shivakumar and H. Garcia-Molina. Finding near-replicas of documents on the web. In *WEBDB: International Workshop on the World Wide Web and Databases, WebDB*. LNCS, 1999.

[22] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, August 2000.

[23] Craig E. Wills, Mikhail Mikhailov, and Hao Shang. N for the price of 1: bundling web objects for more efficient content delivery. In *Proceedings of the 10th International World Wide Web Conference*, 2000.