

Server-Friendly Delta Compression for Efficient Web Access

Anubhav Savant Torsten Suel

CIS Department
Polytechnic University
Brooklyn, NY 11201

The Problem:

“how to use delta compression at a proxy or origin server in a way that minimizes bandwidth consumption while not putting too much strain on the proxy or origin server”

Talk Outline:

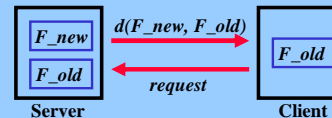
- **intro:** delta compression techniques and their use in HTTP
- **related work**
- **delta compression schemes and evaluation**
- **file synchronization schemes and evaluation**
- **concluding remarks**

1. Introduction

Delta Compression: (differential compression)

- **Goal:** to improve communication and storage efficiency by exploiting file similarity
- **Example: Distribution of software updates**
 - new and old version are similar
 - only need to send a patch to update
 - delta compression: how to compute concise patch
- a delta compressor encodes a target file w.r.t. a reference file by computing a highly compressed representation of their differences

Application Scenario:



- server has copy of both files
➔ local problem at server

Remote File Synchronization: (e.g., rsync)

- server does not know F_old
➔ need protocol between two parties

Applications

- patches for software upgrades
- revision control systems (RCS, CVS)
- versioning file systems
- improving HTTP performance
 - optimistic deltas for latency (AT&T)
 - deltas for wireless links
- storing document collections (web sites, AvantGo)
- incremental backup
- compression of file collection (improving tar+gzip)

LZ-based algorithms for delta compression

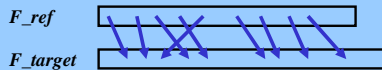
- good and fast (like gzip)
- used in available tools:
 - vdelta/vcdiff K.-P. Vo (AT&T)
 - xdelta J. MacDonald (UCB)
 - zdelta Trendafilov, Memon, Suel (Poly)
- in a nutshell (assume small files)



- put reference file F_old into hash table (dictionary)
- compress F_new - look for best match in both files
- almost like $gzip(cat(F_old, F_new)) - gzip(F_old)$

Some Technical Details

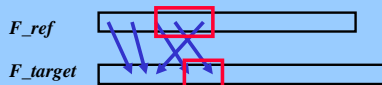
- in reality, some more difficulties
- how to express distances in reference file



- sources of copies in F_ref aligned - better coding

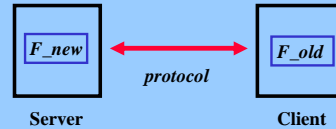
- how to handle long files

- window shifting in both files



Remote File Synchronization

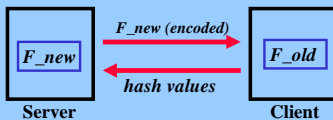
- server does not know F_old



- consider: file comparison problem
- “compare, divide & conquer”: alignment issue
- two main approaches:
 - practice: `rsync` (no good provable bounds)
 - theory: Orlitsky and others (often unimplementable)

The `rsync` algorithms (Tridgell/MacKerras 1996)

- one round of communication:
 - client sends hash values of blocks of F_old to server
 - server uses hash values to compress F_new



- Problem: alignment (suppose insertion at start of file)
- Solution:
 - “consider many different alignments at server”

The `rsync` algorithms (ctd.)

- Client:
 - partition F_old into blocks B_i of some size k
 - compute hash for each block and sends to server



- Server:
 - inserts each received hash ($i, h(B_i)$) into dictionary (hash as key)
 - iterates over F_new :
 - if next window of size k has hash that matches with a hash $h(B_i)$ in dictionary, encode block by i
 - else emit char, shift window by one, and try again
- In practice: two hash functions, plus `gzip`

Some Numbers

- `gzip`, `vcdiff`, `xdelta`, `zdelta`, `rsync`
- files: - `gnu` and `emacs` versions (Hunt/Vo/Tichy)

	gcc size	emacs size
total	27288	27326
gzip	7479	8191
xdelta	461	2131
vcdiff	289	1821
zdelta	250	1465
rsync	876	4428

Compressed size in KB (slightly outdated numbers)

Using Deltas in HTTP

- proxy-based vs. end-to-end
 - include in HTTP?
 - proxies at AOL, NetZero, Sprint PCS
 - class-based encoding by server
- standard vs. optimistic deltas
 - overlap with check for update *Banga/Douglas/Rabinovich 97*
- same URL vs. different URL
 - mostly same URL only
 - use related pages on same site *Chan/Woo 99*
- delta compression vs. file synchronization
 - *rproxy*: proxy based on `rsync` *Tridgell et al 99*

Issues in Current Approaches

- same-URL approaches limited in applicability
- requires proxy or servers to store multiple old versions
- need to fetch old version from disk
- related-page approaches need to find good references
- who chooses reference file?
- also need to store and fetch old pages
- synchronization gives more limited compression
- promising: value-based caching *Brewer WWW2003*
... but compression could be similar to synchronization

2. Delta Compression Results

Setup

- look at site visits from NLANR proxy traces
- choose reference files only among very recently accessed pages
- these pages can be held in memory

Strategies for selecting reference files

- *last-k*: select the last k files
- *longest match-k*: select the pages with longest URL prefix match
- *best-k*: greedily select k best pages
- *best set-of-k*: select set of k best pages
- *last-k+* and *longest match-k+*:
avoid dups among reference files, use previous access if exists

Experimental Setup

- downloaded proxy traces from NLANR
- hashes of “client” IP addresses
- CGI parameters etc. removed
- “clients” can be downstream proxies
- we want to distill client site visits from this
- timeout 5-10 minutes between accesses
- time will hopefully separate different end users
- about 75000 pages
- downloaded next day and stored
- caveats: dynamic pages, duplicates, “odd” sessions

last-k and *longest match-k*

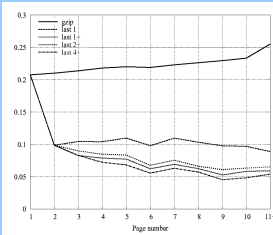


Figure 1. Average compressed size for *last-k* policies.

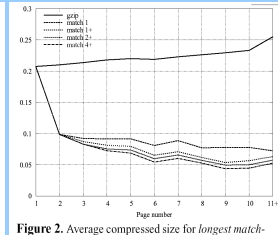


Figure 2. Average compressed size for *longest match-k* policies.

- after one access, compression of 10 instead of 4 for *gzip*
- modified (*last-k+*, *longest match-k+*) strategies much better

best-k and all strategies

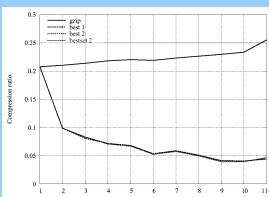


Figure 3. Average compressed size for *best-1*, *best-2*, and *best set-of-2* policies. The three policies result in almost identical graphs.

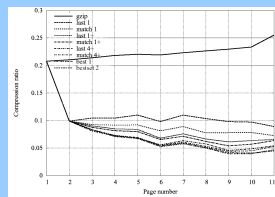


Figure 4. Average compressed size for various policies. The order of the graphs is (from the top) *gzip*, *last-1*, *longest match-1*, *last-1+*, *longest match-1+*, *last-4+* and *longest match-4+*, and *best-1* and *best set-of-2*.

- *best-1* as good as *best set-of-2* !!!
- more reference files do not really help ...
- unless you are not sure which one is best
- also, *last-k+* almost as good as *longest match-k+*

Issue of Page Size Distribution

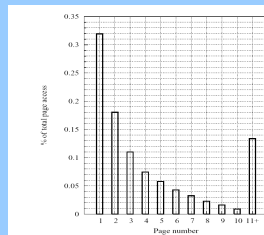


Figure 5. Distributions of page accesses in our traces.

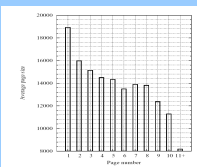


Figure 6. Distributions of page sizes in our traces.

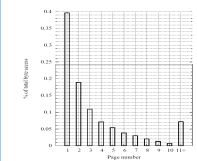


Figure 7. Distributions of total bytes in our traces.

- most sessions are short
- thus most pages close to beginning
- byte savings more moderate
- issue with page sizes for long sessions

Byte Savings

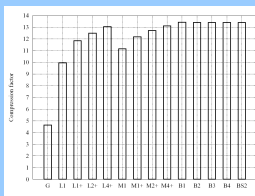


Figure 8. Average compression for policies on eligible pages. The policies are gzip, last-1, last-k+, longest match-1, longest match-k+, best-k, and best set-of-2.

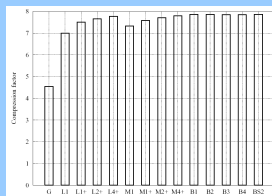


Figure 9. Average compression for policies on all pages.

- factor 12 for best-1+ and longest match-1+ for eligible pages
- best schemes get a bit more than 13
- drop to 7.5 to 8 for all pages (including first page)

Finding the Best Match by Sampling

- if the one best file gives all the benefit, how do we find it?
- sampling-based similarity estimation (Broder, Manber)
- server keeps samples of old files, compares to new

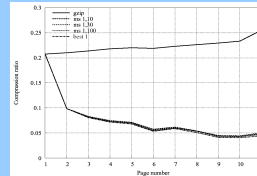


Figure 10. Performance of a sampling-based approach for determining the most similar file, for sample sizes 10, 30, and 100.

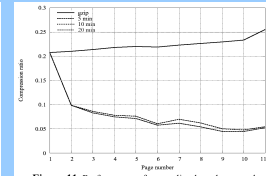


Figure 11. Performance of a sampling-based approach for determining the most similar file, for histories of length 5, 10, and 20 minutes. The latter two cases are almost identical.

- fairly small sample sizes and short history suffice
- overheads in drawing samples, and waiting for new file

Impact of Duplicates

Policy	With duplicates		Duplicates removed	
	Eligible	All pages	Eligible	All pages
GZIP	4.54	4.63	4.64	4.72
L1	9.95	7.00	7.91	6.31
L1+	11.85	7.51	7.91	6.31
L4+	13.03	7.77	8.78	6.62
M1	11.16	7.33	8.12	6.39
M1+	12.17	7.58	8.12	6.39
M4+	13.10	7.79	8.84	6.64
MS 1,10	13.02	7.82	8.73	6.61
B1	13.43	7.86	9.02	6.71

Table 1. A comparison of compression factors achieved with and without duplicates, for policies last-1, last-k+, longest match-1, longest match-k+, most similar-1 with sample size 10, and best-1.

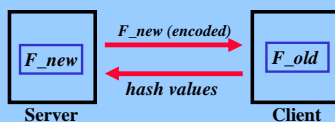
- benefits more moderate if we reduce duplicates

Summary for Delta Compression

- simple strategies for reference file selection work
- most benefits can be obtained with single reference file
- insights on URL matching strategy of Chan/Woo
- best reference file can be obtained with sampling
- very short history (little cache space) suffices

3. Results for File Synchronization

- one round of communication based on rsync
 - client sends hash values of blocks of F_{old} to server
 - server uses hash values to compress F_{new}



- rsync uses default 700 bytes per block size
- 6 bytes per block for hash

Results for Related Pages

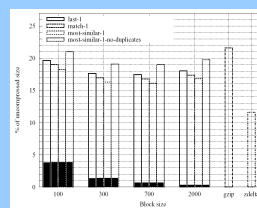


Figure 12. Performance of rsync under the last-1, longest match-1, and most similar-1 policies for different block sizes.

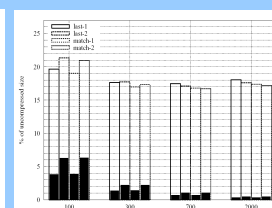


Figure 13. Performance of rsync for one and two reference files with different block sizes.

- only slightly better than gzip on related files in file visit
- more than one reference file does not help
- delta vs. synchronization performance ratio

Results for Same Pages

- 10000 “random” pages recrawled after 2, 20, 72 days

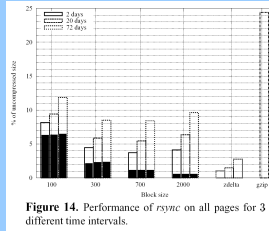


Figure 14. Performance of *rsync* on all pages for 3 different time intervals.

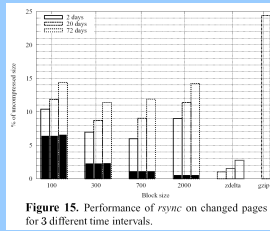


Figure 15. Performance of *rsync* on changed pages for 3 different time intervals.

- much better than gzip even after 72 days
- delta is again factor 3-4 better but ...

Some Observations

- deltas good for site visits, even with very short history
- rsync pretty good for pages revisited after long time
- but why not cache hashes at proxy?
- ... becomes similar to value-based caching (*Brewer 2003*)

Recommendations

- three-level approach
- cache pages at proxy for a few minutes for delta
- cache hashes of pages for longer time for value-based caching
- use rsync for fairly old pages (supply some hashes)