

Cluster-Based Delta Compression of a Collection of Files

Zan Ouyang Nasir Memon
Torsten Suel Dimitre Trendafilov

CIS Department
Polytechnic University
Brooklyn, NY 11201

Problem:

“how to achieve better compression of large collections of files by exploiting similarity among the files”

Overview:

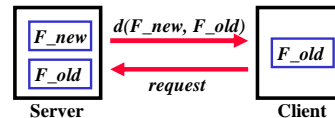
1. Introduction and Motivation
2. Some known Approaches and Results
3. A Cluster-Based Approach
4. Experimental Results
5. Conclusions

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
➔ 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)

Problem in this Paper:

- we are given a set of N files
- some of the files share some similarities, but we do not know which are most similar
- what is the best way to efficiently encode the collection using delta compression?
- limitation: we allow only a single reference file for each target file
- need to find a good sequence of delta compression operations between pairs of files

Delta Compression Tools and Approaches

- compute an edit sequence between the files
 - used by *diff* and *bdiff* utilities
 - what is a good distance measure? (block move, copy)
- practical approach based on Lempel/Ziv:
 - encode repeated substrings by pointing to previous occurrences and occurrences in the reference file
- available tools
 - *vdelta/vcdiff* K.-P. Vo (AT&T)
 - *xdelta* J. MacDonald (UCB)
 - *zdelta* Trendafilov, Memon, Suel (Poly)

Some Technical Details:

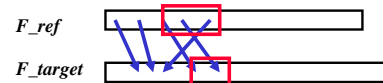
- in reality, some more difficulties
- how to express distances in reference file



- sources of copies in *F_ref* aligned - better coding

- how to handle long files

- window shifting in both files



Performance of delta compressors:

- *gzip*, *vcdiff*, *zdelta*, *zdelta*
- files: - *gnu/emacs* (Hunt/Vo/Tichy)
- caveat for running times: I/O issues
gzip & *zdelta*: first number direct access, second standard I/O
vcdiff numbers: standard I/O

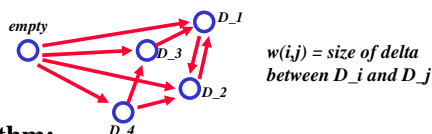
gcc	size (KB)	time (s)	emacs	size (KB)	time (s)
orig.	27288		orig.	27326	
<i>gzip</i>	7479	24/30	<i>gzip</i>	8191	26/35
<i>xdelta</i>	461	20	<i>xdelta</i>	2131	29
<i>vcdiff</i>	289	33	<i>vcdiff</i>	1821	36
<i>zdelta</i>	250	26/32	<i>zdelta</i>	1465	35/42

2. Some Known Approaches and Results

- reduction to the *Maximum Branching Problem* in a weighted directed graph
- a *branching* is an acyclic subset of edges with at most one incoming edge for each node
- a *maximum branching* is a branching of max. weight (*sum of the weights of the edges*)
- can be solved in near-linear time

Reduction to Optimum Branching:

- one node for each document
- directed edges between every pair of nodes
- weight equal to benefit due to delta compression



Algorithm:

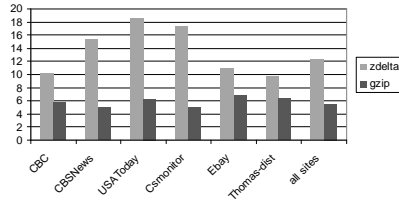
- generate graph
- compute optimum branching
- delta compression according to branching

Summary of known results:

- optimal delta in polynomial time if each file uses only one reference file (*optimal branching*)
- Problem with the optimal branching approach:
 - N^2 edges between N documents
 - solution might have long chains
- NP-complete if we limit length of chains [Tate 91]
- NP-complete if we use more than one reference per file (*hypergraph branching*, Adler/Mitzenmacher 2001)

Experimental Results for Optimum Branching:

- fast implementation based on *Boost* library
- tested on collections of pages from web sites



- inefficient for more than a few hundred files
- most time spent on computing the weights
- branching much faster (but also N^2)

3. A Cluster-Based Approach

Challenge: to compute a good branching without computing all N^2 edges precisely

General Framework:

(1) Collection Analysis

Perform a clustering computation that identifies pairs of files that are good candidates for delta compression. Build a sparse subgraph G' with edges between pairs.

(2) Weight Assignment

Compute or approximate edge weights for G' .

(3) Maximum Branching

Perform a maximum branching computation on G'

Techniques for efficient clustering

- Recent clustering techniques based on sampling by Broder and by Indyk
 - first generate all q -grams (substrings of length q)
 - files are similar if they have many q -grams in common
 - use minwise independent hashing to select a small number w of representative q -grams for each file
- Broder: $O(w * N^2)$ to estimate distances between all N^2 pairs of files (also by Manber/Wu)
- Indyk: $O(w * N * \log(N))$ to identify pairs of files with distance below a chosen threshold

Techniques for efficient clustering (cont.)

- Broder: minwise hashing (MH)
 - use random hash function to map q -grams to integers
 - retain the smallest integer obtained as sample
 - repeat w times to get w samples (can be optimized)
 - use samples to estimate distance between each pair
- Indyk: Locally sensitive hashing (LSH)
 - sampling same as before
 - instead of computing distance between each pair of samples, construct signatures from sample
 - signature = bitwise concatenation of several samples
 - samples have large intersection if identical signature
 - repeat a few times

Discussion of Distance Measure

- Two possible distance measures on q -grams:
 - Let $S(F)$ be the set of q -grams of file F
 - Intersection:
$$d(F, F') = (S(F) \text{ intersect } S(F')) / (S(F) \text{ union } S(F'))$$
 - Containment:
$$d(F, F') = (S(F) \text{ intersect } S(F')) / S(F)$$
- Intersection symmetric, Containment asymmetric
- There are formal bounds relating both measures to edit distance with and without block moves
- Inaccuracy due to distance measure and sampling
- In practice OK (as we will see)

Design decisions:

- Intersection or inclusion measure?
- Which edges to keep?
 - all edges above a certain similarity threshold
 - the k best incoming edges for each file (NN)
- Approximate or exact weights for G' ?
- Different sampling approaches
- Sampling ratios
- Value of q for q -grams (we used $q=4$)
- Indyk approach supports only intersection and similarity threshold (extension difficult)
- Indyk does not provide weight estimate

4. Experimental Results

- Implementation in C++ using *zdelta* compressor
- Experiments on P4 and UltraSparc architectures
- Data sets:
 - union of 6 commercial web sites used before (49 MB)
 - 21000 pages from poly.edu domain (257 MB)
 - gcc and emacs distributions (>1000 files, 25 MB)
 - changed pages from web sites
- Comparison with *gzip*, *tar+gzip*, and *cat+gzip*
- See paper and report for details

Threshold-Based Approach:

- Keep edges with distance below threshold
- Based on Broder (MH)
- Fixed sample size and fixed sampling ratio

algorithm	smp size	threshold	remaining edges	br. Size	benefit over zlib
optimal			2782224	1667	6980935
MH	100	20%	357961	1616	6953569
intersect		40%	154533	1434	6601218
		60%	43289	988	5326760
		80%	2629	265	1372123
MH	1/128	20%	391682	1641	6961645
intersect		40%	165563	1481	6665907
		60%	42477	1060	5450312
		80%	4022	368	1621910
MH	1/128	20%	1258272	1658	6977748
contain		40%	463213	1638	6943999
		60%	225675	1550	6724167
		80%	79404	1074	5016699

Discussion of threshold-based approach:

- Sampling method not critical
- Containment slightly better than Intersection (but can result in undesirable pairings)
- There is no really good threshold
 - low threshold keeps too many edges (inefficient)
 - high threshold cuts too many edges (bad compression)
- Possible solutions
 - use k nearest neighbors
 - use estimates instead of precise edge weights
 - use other heuristics to prune edges from dense areas

Nearest Neighbor-Based Approach:

- Keep edges from k nearest neighbors ($k = 1, 2, 4, 8$)
- Based on Broder (MH) with fixed sampling ratio

sample size	k	cluster tim	weighting	br. time	benefit over zlib
1/2	1	1198.25	51.44	0.02	6137816
	2	1201.27	84.17	0.02	6693921
	4	1198	149.99	0.04	6879510
	8	1198.91	287.31	0.09	6937119
1/128	1	40.52	47.77	0.02	6124913
	2	40.65	82.88	0.03	6604095
	4	40.57	149.06	0.03	6774854
	8	40.82	283.57	0.09	6883487

- Good compression even for small k
- Time for computing weights increases linearly with k
- Clustering time linear in ratio and quadratic in # files
- Open theoretical question on performance of k -NN

Nearest Neighbor with Estimated Weights:

- Keep edges from k nearest neighbors ($k = 1, 2, 3, 4$)
- Use intersection estimate as weight of edge

k	cluster tim	branching	zdelta time	benefit over zlib (MB)
1	39.26	0.02	45.63	5.832565308
2	39.35	0.02	48.49	6.111814499
3	39.35	0.02	48.14	6.164761543
4	39.4	0.06	49.63	6.20189476

- Compression only slightly worse than for exact weights
- Time for computing weights reduced to zero
- Clustering time still quadratic in # files but OK for up to thousands of files
- *zdelta* time affected by disk (compared to *tar+gzip*)

Indyk (LSH) with Pruning Heuristic:

- Threshold and fixed sampling ratio
- Heuristic: keep only sparse subgraph "everywhere"

threshold	edges	branching	benefit over zlib
20%	28,866	1640	6689872
40%	8,421	1612	6242688
60%	6,316	1538	5426000
80%	2,527	1483	4945364

- Compression degrades somewhat
- Could try to use more sound approach (biased sampling)
- Clustering time better than with Broder for a few thousand files and more
- Could do Indyk (LSH) followed by Broder (MH)

Experiments on Large Data Set:

- 21000 pages, 257 MB
- Results still very unoptimized in terms of speed

algorithm	running time	size (MB)
zlib	74	42.3
cat+zlib	80	30.5
best MH	996	23.7
best LSH	800	21.7

- However: cat+bzip2 does even better in some cases!
 - depends on page ordering and file sizes
 - our techniques are (now) faster

5. Conclusions

- We studied compression of collections of files
- Framework for cluster-based delta compression
- Implemented and optimized several approaches
- Rich set of possible problems and techniques
- Results still preliminary
- Part of ongoing research effort on delta compression and file synchronization

Current and future work

- Extending *zdelta* to several base files
- Compressing a collection using several base files
 - optimal solution NP-Complete
 - approximation results for optimum benefit
- Approach based on *Hamiltonian Path*
 - greedy heuristic for *Hamiltonian Path*
 - almost as good as *Maximum Branching*
 - useful for case of several base files
 - useful for tar+gzip and tar+bzip2
- Goal: towards a better generic utility (than tar+gzip)
- Study based on data sets from ftp sites and web

Application: Large Web Archival System

- Consider a massive collection of web pages
- *Internet Archive*: 10 billion pages, 100TB
- Many versions of each page:
Wayback Machine (at www.archive.org)
- Archive does not currently employ delta compression [Kahle 2002]
- How to build a TB storage system that
 - employs delta compression
 - has good insertion and random read performance
 - has good streaming performance.
- Limited to very simple heuristics