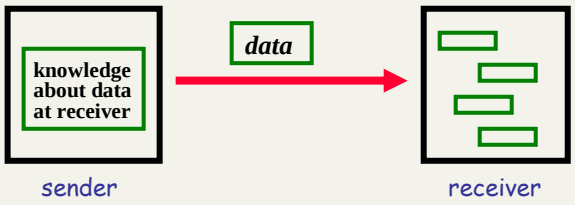
Algoritmi per IR

Compression of file collections

Background



- network links are getting faster and faster but
  - many clients still connected by fairly slow links (mobile?)
  - people wish to send more and more data

how can we make this transparent to the user?

Two standard techniques

- caching: *"avoid sending the same object again"*
  - done on the basis of objects
  - only works if objects completely unchanged
  - How about objects that are slightly changed?
- compression: *"remove redundancy in transmitted data"*
  - avoid repeated substrings in data
  - can be extended to history of past transmissions (overhead)
  - How if the sender has never seen data at receiver ?

Types of Techniques

- Common knowledge between sender & receiver
  - Unstructured file: delta compression
- "partial" knowledge
  - Unstructured files: file synchronization
  - Record-based data: set reconciliation

Formalization

- **Delta compression** [*diff*, *zdelta*, *REBL*,…]
  - Compress file *f* deploying file *f'*
  - Compress a group of files
  - Speed-up web access by sending differences between the requested page and the ones available in cache
- **File synchronization** [*rsynch*, *zsync*]
  - Client updates old file f_{old} with f_{new} available on a server
  - Mirroring, Shared Crawling, Content Distr. Net
- **Set reconciliation**
  - Client updates structured old file f_{old} with f_{new} available on a server
  - Update of contacts or appointments, intersect IL in P2P search engine

Z-delta compression (one-to-one)

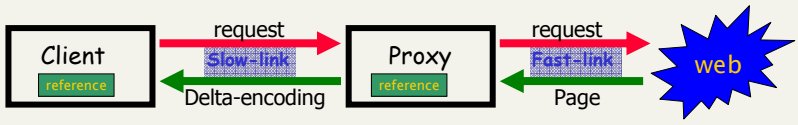
Problem: We have two files f_{known} and f_{new} and the goal is to compute a file f_d of minimum size such that f_{new} can be derived from f_{known} and f_d

- Assume that block **moves** and **copies** are allowed
- Find an optimal covering set of f_{new} based on f_{known}
- **LZ77-scheme** provides and efficient, optimal solution
  - f_{known} is "previously encoded text", compress $f_{known}f_{new}$ starting from f_{new}
- **zdelta** is one of the best implementations

		Emacs size	Emacs time
uncompr		27Mb	---
gzip		8Mb	35 secs
zdelta		1.5Mb	42 secs

Efficient Web Access

Dual proxy architecture: pair of proxies located on each side of the slow link use a proprietary protocol to increase performance over this link



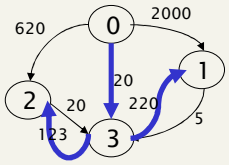
- Use *zdelta* to reduce traffic:
  - Old version available at both proxies
  - Restricted to pages already visited (30% hits), URL-prefix match

Small cache

Cluster-based delta compression

Problem: We wish to compress a group of files F

- Useful on a dynamic collection of web pages, back-ups, …
- Apply *pairwise* **zdelta**: find for each f ∈ F a *good reference*
- Reduction to the Min Branching problem on DAGs
  - Build a weighted graph G_F, nodes=files, weights= **zdelta**-size
  - Insert a dummy node connected to all, and weights are **gzip**-coding
  - Compute the min branching = directed spanning tree of min tot cost, covering G's nodes.



		space	time
uncompr		30Mb	---
tgz		20%	linear
THIS		8%	quadratic

Improvement (group of files)

What about many-to-one compression?

Problem: Constructing G is very costly, n^2 edge calculations (*zdelta exec*)

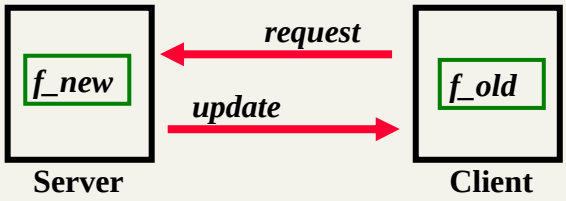
- We wish to exploit some pruning approach
  - **Collection analysis**: Cluster the files that appear similar and thus good candidates for zdelta-compression. Build a sparse weighted graph G'_F containing only edges between those pairs of files
  - **Assign weights**: Estimate appropriate edge weights for G'_F thus saving zdelta execution. Nonetheless, strict n^2 time

		space	time
uncompr		260Mb	---
tgz		12%	2 mins
THIS		8%	16 mins

Algoritmi per IR

File Synchronization

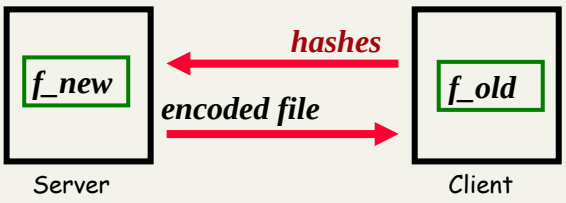
File synch: The problem

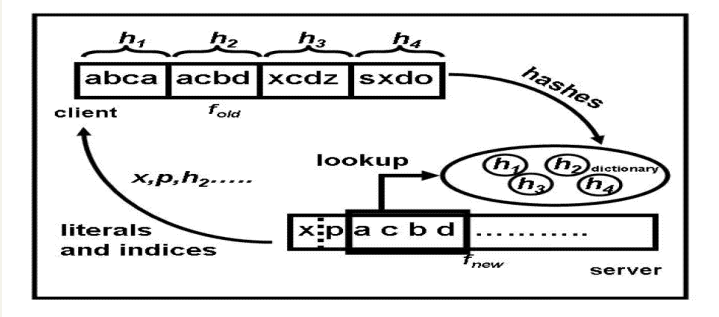


- **client** wants to update an out-dated file
- server has new file but does not know the old file
- update without sending entire *f_new* *(using similarity)*
- *rsync*: file synch tool, distributed with Linux

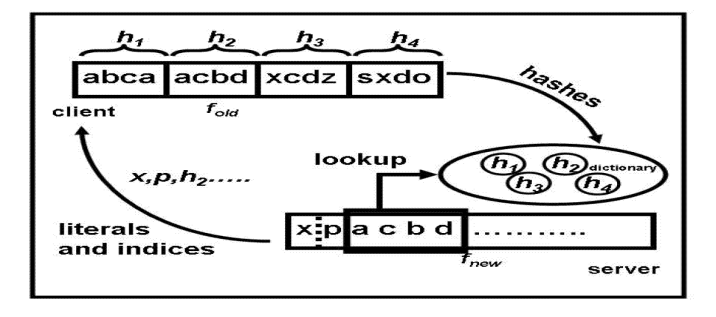
Delta compression is a sort of local synch
Since the server has both copies of the files

The rsync algorithm





The rsync algorithm (contd)



- simple, widely used, **single** roundtrip
- optimizations: 4-byte rolling hash + 2-byte MD5, *gzip* for literals
- choice of block size problematic (*default*: max{700, √n} bytes)
- not good in theory: granularity of changes may disrupt use of blocks

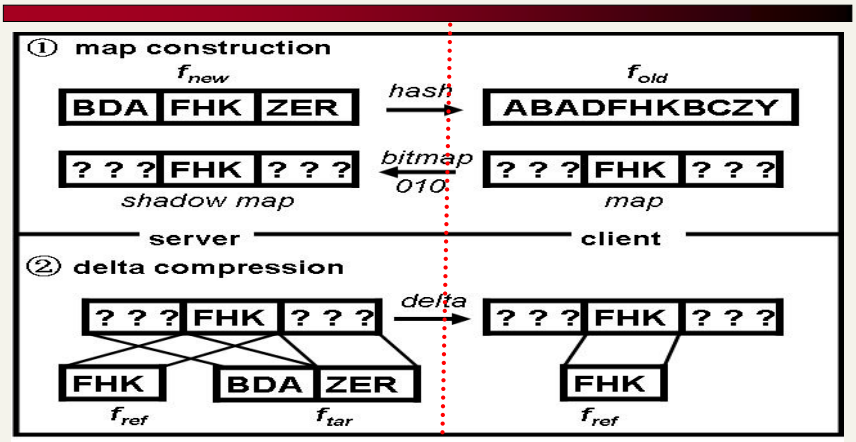
Rsync: some experiments

	gcc size	emacs size
total	27288	27326
gzip	7563	8577
zdelta	227	1431
rsync	964	4452

Compressed size in KB (slightly outdated numbers)

Factor 3-5 gap between rsync and zdelta !!

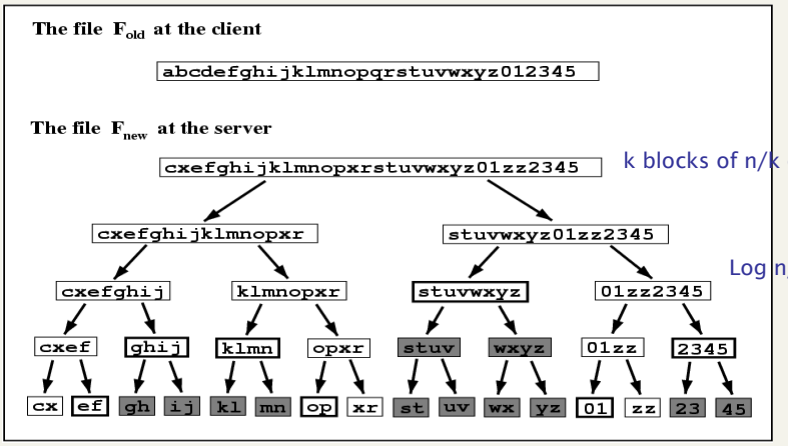
A new framework: zsync



Server sends hashes (unlike the client in **rsync**), clients checks them

Server deploys the common f_{ref} to compress the new f_{tar} (**rsync** compress just it).

A multi-round protocol



If distance k, then on each level ≤ k hashes not find a match in the other file.

The communication complexity is O(k lg n lg(n/k)) bits

Next lecture

Set reconciliation

Problem: Given two sets S_A and S_B of integer values located on two machines *A* and B, determine the difference between the two sets at one or both of the machines.

Requirements: The cost should be proportional to the size k of the difference, where k may or may not be known in advance to the parties.

Note:

- ✓ file synchronization: O(k (lg n) (lg n/k))
- ✓ set reconciliation: O(k lg n)
  - ✓ set reconciliation is "easier" [it is *record-based*]