# rproxy: a practical HTTP delta compression system

Andrew Tridgell
tridge@samba.org

# The rsync algorithm

- The rsync algorithm is a remote differencing and update algorithm. It allows you to efficiently update a file on one machine with the contents of a file on another machine while taking advantage of the common content between the old and new file.
  - **signature generation** A signature block is generated for the old file
  - **signature search** The differences between the old and new data are computed using a checksum search
  - **reconstruction** The new file is reconstructed

# rsync properties

- Low latency due to single pass update
- Handles arbitrary byte-level insertions, deletions and movements
- Pipelined for single pass operation on multiple files
- Two signatures per block
  - a rolling hash for very efficient match generation
  - a strong hash for verification

# rdiff example

- Create a signature of the old file
  - `rdiff signature OLDFILE > sig.dat`
- Create a delta using the signature and new file
  - `rdiff delta sig.dat NEWFILE > delta.dat`
- Apply the delta to the old file
  - `rdiff patch delta.dat OLDFILE > new.dat`
- Check the result
  - `cmp NEWFILE new.dat`

# rsync in HTTP

- Large parts of the web are moving to dynamic content. Traditional web caches can't cache dynamic content. Integrating rsync in http solves this problem.
  - builds on existing web infrastructure
  - all content is cacheable
  - no extra round trips

# Integrating rsync in HTTP

- A choice of either server generated or client generated signatures.
- Client signatures use one extra HTTP header and a new HTTP Content-Encoding type.
  - The client generates a signature from the cached file and adds it to the request as a Rsync-Signature header. It is base64 encoded.
  - The server generates the page as usual then performs a checksum search to generate the differences.
  - The client receives a "Content-Encoded: rsync" reply and decodes it to give the new page.

# Server generated signatures

- The client generates a null signature block and adds it to the request as a Rsync-Signature header.
- The server generates the page then performs a rsync differencing run to generate a rsync-encoded page. This leads to a set of deflate compressed literal data in a reply marked as "Content-Encoding: rsync"
- The server also sends a rsync signature for the new page and appends this to the rsync-encoded reply.
- The client receives the reply and decodes it to give the new page.
- The next time the client requests that URL it provides the signature back to the server.

# Which is better?

- The client doesn't need to know the signature format. This allows the server a lot of flexibility. It also allows the server to use a signature-token instead of a full signature if it wants to.
- The signature only passes over the wire between clients and servers that both know about rsync.
- There are possible patent problems with the client-generated signatures. There are no patent problems with server generated signatures.

# Disadvantages

- The main disadvantage is the higher computational load on the server.
  - Without the deflate compression the rsync algorithm can run at 15-25 Mbyte/sec on a cheap PC. Very few web servers are on links that fast.
  - With deflate compression this reduces to about 6 MB/sec on my laptop, which is still quite acceptable for most servers.
  - In either case, it is often much easier to add more CPU power than it is to add network bandwidth.

# Proxy servers

- There can be a lot of benefit to implementing rsync in web proxy servers. The proxy has 4 scenarios to deal with:
  - The downstream client supplied a rsync header and we got a rsync reply. We send on the reply as is.
  - The client didn't supply a rsync header and we didn't get a rsync reply. We send on the reply as is.
  - The client didn't supply a rsync header but we got an rsync reply. We decode it before sending it on.
  - The client gave a rsync header and we got a non-encoded reply. We need to rsync encode the reply and send it on to the client.

# Failure probability

- The rsync algorithm is probabilistic, as all algorithms of this kind must be. That means there is a non-zero chance of failure. The probability of failure can be reduced to arbitrarily small levels by the choice of appropriate signature lengths.
- With the signature algorithm and sizes currently in use in rsync and a rate of one million transfers per second we should see a failure in about $10^{11}$ years. The universe is thought to be about $10^{10}$ years old.

# Cache file selection

- An interesting property of a rsync based web cache is that you can choose a cached page that doesn't exactly match a URL.
- The most obvious thing to do is to truncate the URL at the first '?', removing CGI parameters. An alternative be to try an exact URL then progressively trim until a match is found.
- The result is that you get the speedup if there is any common content between the page you want and a previous page from the same site.

# A working prototype

- rproxy is a simple fork-per-connection web proxy written in C. It implements both the client and server side of rsync in HTTP.
  - can be chained with other proxy servers.
  - uses a maximum signature size of 512 bytes.
  - cache files URLs are truncated at the first '?'.
  - all requests are pipelined for minimum latency
  - implemented on top of librsync, a simple rsync library implementation.
  - zlib is used for deflate compression of the encoded data.

# Initial results

- Overall, rproxy reduced the repeat URL traffic on my link by 76%

| Site | Saving % |
|---|---|
| linuxtoday | 81 |
| slashdot | 82 |
| linux.com | 93 |
| excite.com | 93 |