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.

The basic algorithm consists of the following steps:

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 in http

The web is 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 into HTTP

Integrating rsync and HTTP can be done with eaither server generated or client generated signatures.

Using client generated signatures involves the use of one extra HTTP header and a new HTTP Content-Encoding type. If we assume that the client has a cached file to work with then this is what happens:

- 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 with the signature to generate the differences.
- The client receives a "Content-Encoded: rsync" reply and decodes it to give the new page.

Server generated signatures

An alternative is to use server generated signatures. I'll assume the client starts out with no suitable cache file.

- The client initially generates a null signature block and adds it to the request as a Rsync-Signature header.
- The server generates the page as usual then performs a rsync differencing run with the null signature to generate the 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 generates a full rsync signature for the new page and appends this to the rsync-encoded reply.
- The client receives a "Content-Encoded: rsync" reply and decodes it to give the new page. The client saves the server

generated signature alongside the page in its cache.

- The next time the client requests that URL (or a related URL) it provides the signature back to the server.

Which is better?

We think that server generated signatures are a better choice.

- 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.

Note that in both cases all operations can be streamed, thus avoiding a latency increase.

Disadvantages

The main downside of rsync in HTTP is that it puts a higher computational burden on the server. The main thing influencing the computational cost is the choice of whether to deflate compress the data as well as rsync encode it. Without the deflate compression the rsync algorithm can easily run at 6-10MB/sec on a cheap PC. Very few web servers are on links that fat.

With deflate compression this reduces to about 1.5 MB/sec on my PII/350, 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

Rsync doesn't need to be implemented in the end-points of a HTTP connection. There can be a lot of benefits to implementing it in web proxy servers. In that case the proxy supplies the signature header if the downstream client doesn't. The proxy then has 4 scenarious to deal with:

- The downstream client supplied a Rsync-Signature header and we got a Rsync-Encoded reply from the upstream server. We send on the reply to the client as is.
- The downstream client didn't supply a Rsync-Signature header and we didn't get a Rsync-Encoded reply. We send the reply on to the client as is.
- The downstream client didn't supply a Rsync-Signature header

but we got an Rsync-Encoded reply. We need to decode it before sending it on to the client.

- The downstream client did supply us with a Rsync-Signature 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 probabuility 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 interesing property of a rsync based web cache is that it doesn't need to be accurate. This allows the cache to 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 '?',thus removing CGI parameters. Even better would be to first try an exact URL then progressively trim the URL until a match is found. If none is found then a null signature is used.

The result is that you get the speedup of rsync encoding if there is any common content between the page you want and a previous page from the same site. With the widespread use of large boilerplate HTML templates, this is a big win.

A working prototype - rproxy

rproxy is a simple fork-per-connection web proxy written in C. It implements both the client and server side of rsync in HTTP.

- it can be chained with other proxy servers.
- It uses a maximum signature size of 512 bytes.
- Files smaller than 2k are not cached.
- Cache files are based on a hash of the URL truncated at the first '?'.
- All requests are pipelined for minimum latency impact.
- The proxy is implemented on top of librsync, a simple rsync library implementation.
- zlib is used for deflate compression of the rsync encoded data.

Initial results

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

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