Flying Squid

Tufts University: Senior Design Project

Advisor: Prof. Fahad Dogar

Sunjay Bhatia, Victor Chao, Jim Mao, and Siddhartha Prasad

6th April 2016

Contents

1 Abstract

Flying Squid is a proxy that can provide a distributed, cloud based cache localized at the edge of a network. The system can eliminate unnecessary bandwidth use, and provide faster data delivery than content providers by themselves. Flying Squid aims to be a truly web-based personal proxy, with shared cloud based caches and value based partial caching. By improving content delivery speed and reducing client data usage, the proxy will be especially useful to mobile users and those with spotty network connectivity.

2 Background

Proxy servers help communication between two entities on a network. They often act as intermediaries between clients and content-delivering servers. They help optimize and add structure to networks and distributed systems. The latency incurred from downloading content directly from target servers is fast becoming a limiting factor on internet speeds. With sharp upward trends in the number of devices connected to the internet, this has forced cell-phone providers, ISPs and even large local area networks to use these servers to optimize both networks and content delivery. As a result, end users consciously and unconsciously end up using several proxies a day. While these proxies have sophisticated caching and compression mechanisms, they are stand-alone programs that do not do the most they can to share information about users and the data they are caching.

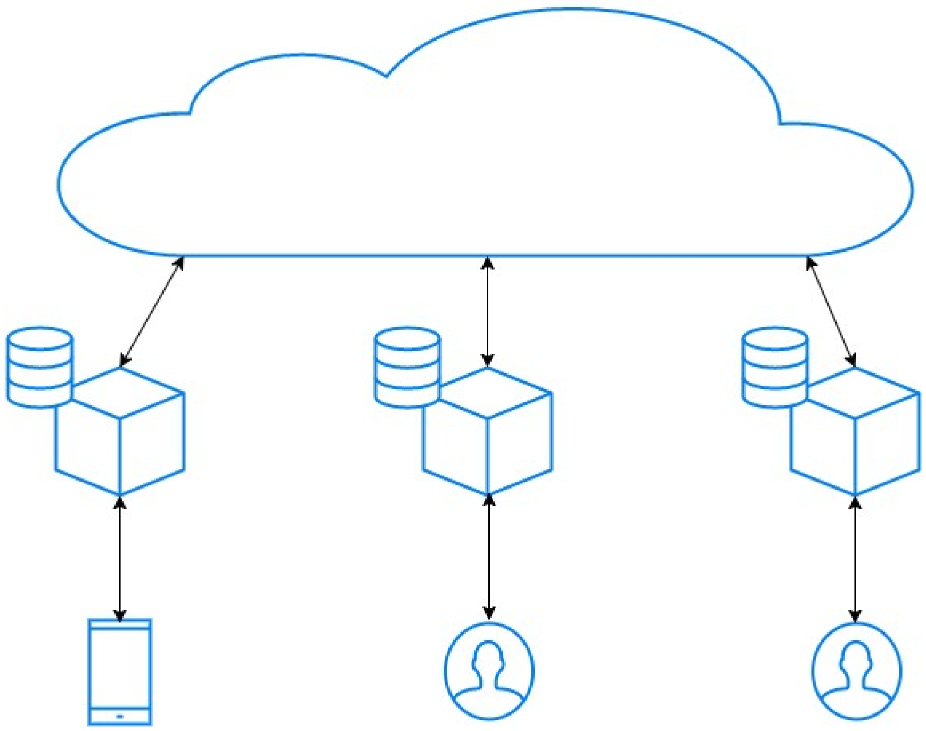


Figure 1: Traditional Client-Proxy Architecture

In Figure 1 we can see a simple representation of the traditional role proxies have. Users connect to proxy servers as intermediaries to their connection to a network, such as the internet. Each of these proxies traditionally has its own disk and RAM cache. It is also possible to utilize these proxies as a distributed network with a shared cache represented by ‘joining ’each of the disk caches into a unified cache. However, this cache still only incorporates the basic levels of disk and RAM storage and there is potentially significant overhead in communicating between proxies to locate content that may be on a different physical machine than the one handling the request and serving the content.

3 Flying Squid

4 Cloud Caching

5 Fingerprinting: Value Based Caching

Whether an application uses compression or intricate caching mechanisms, the easiest way to improve content delivery speed is to *send less over the network*. With this in mind, a lot of research has been done with respect to protocols focused on eliminating redundant data transfers over HTTP links. Much of this redundancy is caused by the first-class nature of the ‘file ’in traditional caching systems.

Inspired by research at Berkley on value based caching[1], Flying Squid’s Fingerprinting system caches with respect to data, not file name. As a result, only changed and new parts of content need ever retransmitted over a network. This is presented as a simple, C++ class based library.

5.1 The API

5.1.1 Client

```
class RabinClient
{
    public:
        /* Server name and port number*/
        RabinClient(char * hostname, int port_);

         ~RabinClient();

        /*
        * Receives a file from the server into the open, write enabled
            file pointer 'file'.
        * Returns the number of blocks added to the file.
         * This is a blocking call.
        */
        unsigned receive_file(FILE *file);

       /* Establishes a connection to the server */
        int connect_to_server();

         /* Disconnects from the server */
        int disconnect_from_server();

};
```

5.1.2 Server

```
class RabinServer
{

   public:
       /* Port at which to be open*/
       RabinServer(int port_);
       ~RabinServer();

       /* Sends a file of size s to the client.
             This is a blocking call.
       */
       int send_file(char *file, size_t s);


       /* Listens for the client and accepts a connection
       * Must be called before write_to_client is called */
       int connect_to_client();

        /* Connects to the client */
       int disconnect_from_client();

};
```

5.2 Technical Approach

Flying Squid's Rabin Fingerprinting library provides a clean file transfer abstraction to the user, hiding the intricacies of a partial file transfer abstraction.

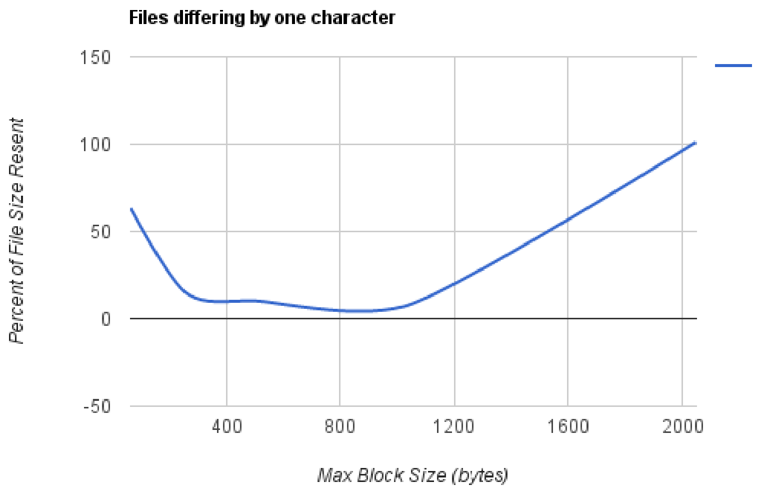
1. Server-side files are split into blocks of maximum size $1KB$ using a `djb-2`[2] based uniform random function of the form:
   $f : \{\text{byte} * \text{byte} * \text{byte}\} \rightarrow \{0, 1, ..., 1023\}$
2. Block boundaries are determined when this function returns 0. These blocks are treated as first-class objects and are hashed locally. A custom protocol can then be used to transfer the blocks representing a file over a TCP connection. Only the hash digests (identifiers) of previously transmitted blocks are transferred. Blocks of size 0 are used to denote EOF.
3. The client leverages the ordering properties of TCP to receive blocks in order. These are then locally stored, and identifiers are used to re-assemble them into traditional files.

5.3 Statistics

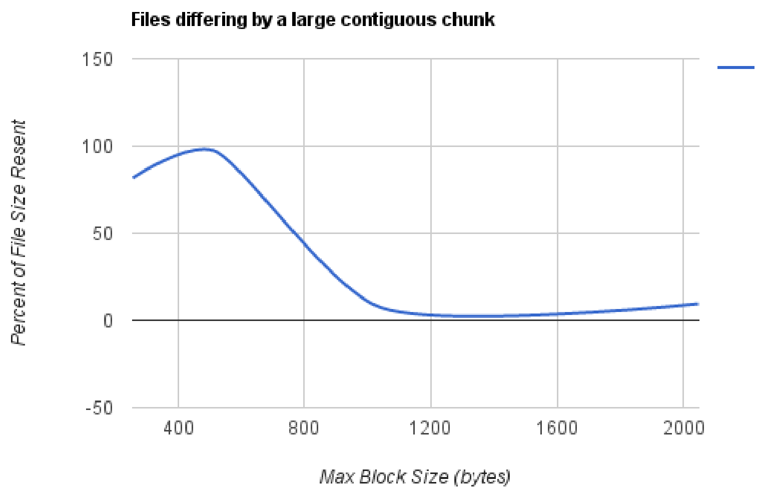
Pairs of files were generated that differ in the following ways:

1. Haskell: One file has an added byte.
2. HTML: Files differ by one large contiguous chunk.
3. C: Files differ by a large number of chunks distributed through the file

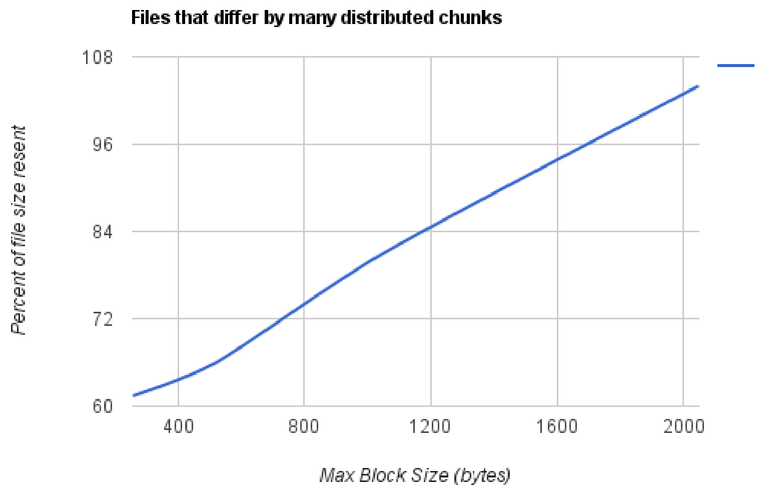
These files were tested over the Flying Squid fingerprinting system, with different maximum block sizes. Max block sizes smaller than 256 bytes were found to cause significantly more conflicts with the internal `djb2` hash.



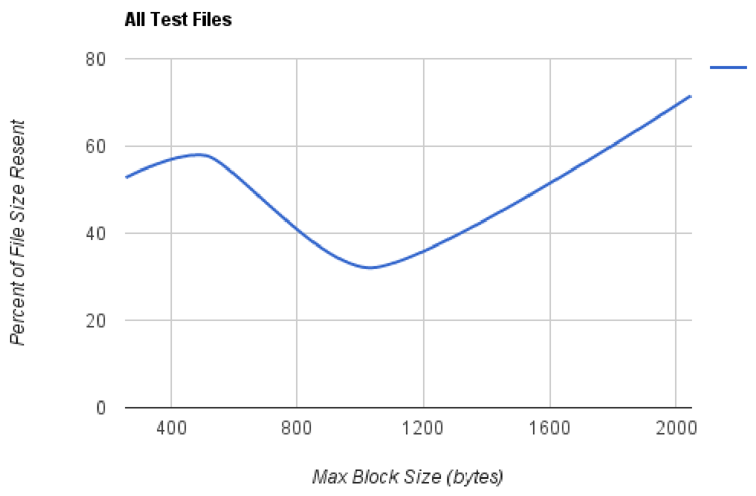
For files differing by one byte, a balance has to be struck between several small block headers being sent and the size of the retransmitted block(s). At most one block will be retransmitted due to the small difference between files. An optimal block size was found around 1000 bytes



For files differing by a large contiguous chunk a minimum was found around 1400 bytes.



For files differing by a large number of chunks distributed through the file, smaller max block sizes were generally found to be better.



Taking the average of all these files, it was found that a minimal amount was resent around a maximum block size of $1KB$. As a result, the maximum value chosen was 1024 bytes, with an expected max size of 512 bytes.

6 Conclusion

References

[1] Eric Brewer Sean C. Rhea Kevin Lian. "Value-Based Web Caching". In: *Proceedings of the 12th international conference on World Wide Web* (2003), pp. 619–628.

[2] Ozan Yigit. *Hash Functions*. URL: http://www.cse.yorku.ca/~oz/hash.html (visited on 04/06/2016).