Report of option2

Dong Tang

dtan698

I implemented the server, cache, client (including all the functions in the requirements)

My train of thought is:

Convert image file to byte array: Reads an image file and converts its content to a byte array. The image file itself is binary data, so there is no need to convert it to a hexadecimal representation.

Split Byte Array: Splits a byte array into fixed-size or variable-size chunks. You can use the rolling hash strategy you mentioned earlier to achieve variable sized blocks.

Compute a hash value for each block: Compute a unique hash value for each data block using a cryptographic hash function such as SHA-256. Hash values are usually represented as hexadecimal strings.

Store blocks and hashes: On the server, store each block and its corresponding hash in the file system. To simplify implementation, you can store data blocks as separate binary files and store all hashes in a text file or database.

Caching a requested file: When caching a requested file, the server first sends metadata containing a list of file hashes. Caches can use this metadata to determine which chunks already exist and which need to be downloaded from the server. Then, the cache requests the missing blocks and stores those blocks locally.

File transfer from cache to client: The cache reassembles the file based on metadata, concatenates the data blocks into a byte array, and sends it to the client.

Server:

A "1" command is received, then the requested filename is read.

If the file is found, send the chunk number.

Traverse each chunk, if the chunk exists in the cache, send opcode "0" and hash value; otherwise, send opcode "1", chunk length, hash value, and finally chunk data.

Cache:

A "1" command is received, then the requested filename is read.

Connect to the server, forward the "1" command, and send the requested filename.

Read the server's response and continue to read the number of chunks.

For each chunk, read the opcode. If the opcode is "1", read the chunk length, hash value, and then read the chunk data and save it to the temporary folder and cache folder; if the opcode is "0", only read the hash value, and then read from Copy chunks from the cache folder to a temporary folder.

After processing all chunks, use the ReassembleFileFromChunks method to reassemble the file.

(e)

The main difference between the two options lies in the processing of files.

For option 1, the server does not need to perform any processing on the file. When it receives the request from the cache, it directly transfers the entire file. When the cache receives the request from the client, it only determines whether to request the server by judging the file name. This method is used in The secondary response when the processing content remains unchanged is very short, and for any file, as long as the name is different, the entire file must be transmitted and stored in the cache. Although this does not require much calculation, it consumes memory.

For option 2, the server cuts all files into variable-sized chunks during initialization, so there will be a certain delay at this stage, and then the server and cache process chunks instead of entire files throughout the process. In this way, the changes of the same file can be distinguished, and it also has a high reuse rate when processing different files with similar content, and the response is very fast, which saves bandwidth, but requires a certain amount of computing power. At the same time, the entire file is no longer stored in the cache, but file blocks, which can save memory.