Mandatory Assignment 2 - INF-2202

Lars Ailo Bongo (larsab@cs.uit.no)
Tim Teige (tim.teige91@gmail.com)
Department of Computer Science, University of Tromsø

20.09.2016

Introduction

In this mandatory assignment you will implement a deduplication sender and receiver using Go. Deduplication is a global compression technique that is often used by backup systems. It achieves a very high compression ratio by identifying redundancy over the entire dataset instead of just a local window. Both sides maintain a big cache of previously sent data, and for redundant data a short fingerprint is sent instead of the data content. Deduplication systems need to support high throughput.

Deduplication

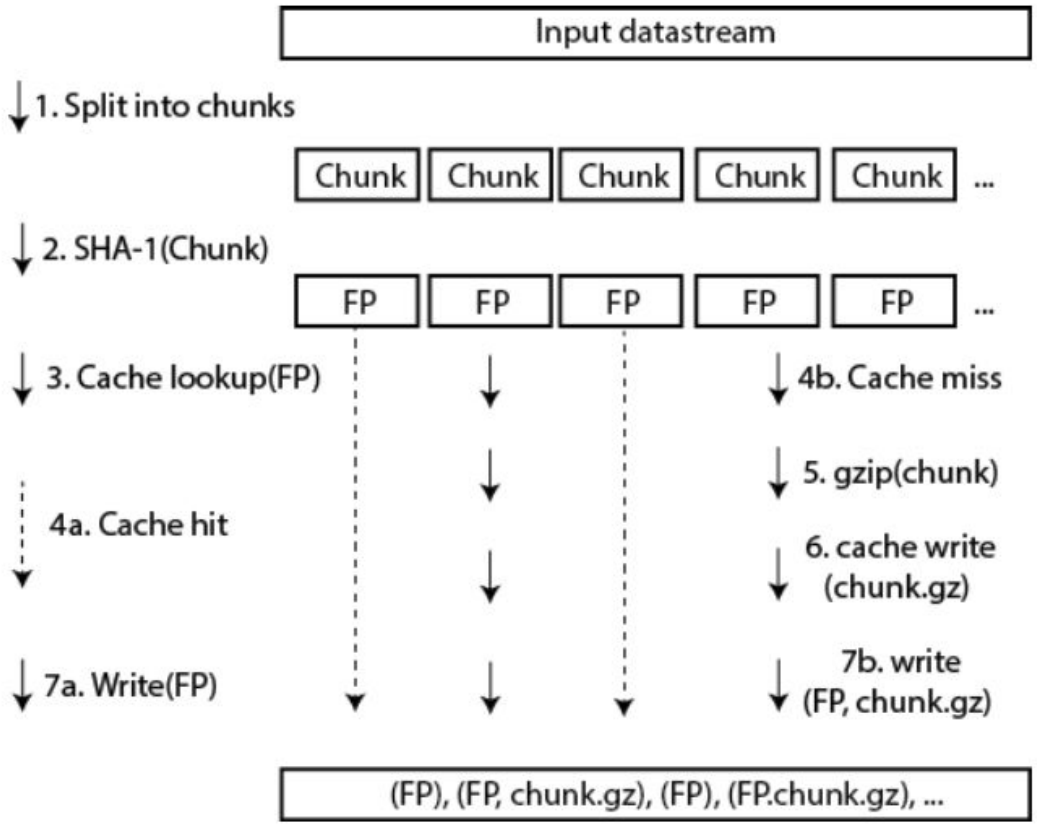

Figure 1: Sender side.

The sender should do the following to compress an input datastream

1. Split the input data into chunks.
2. Calculate a SHA-1 fingerprint for each chunk.
3. Check if the cache contains a chunk with the calculated fingerprint.
4. If an entry was found, the chunk has been sent earlier and hence the chunk is also cached at the receiver side. Only the fingerprint is therefore sent.
5. If an entry was not found, the chunk has not been sent earlier. It is therefore compressed using for example gzip, the compressed data is written to the cache, and the fingerprint and compressed data are sent to the receiver.

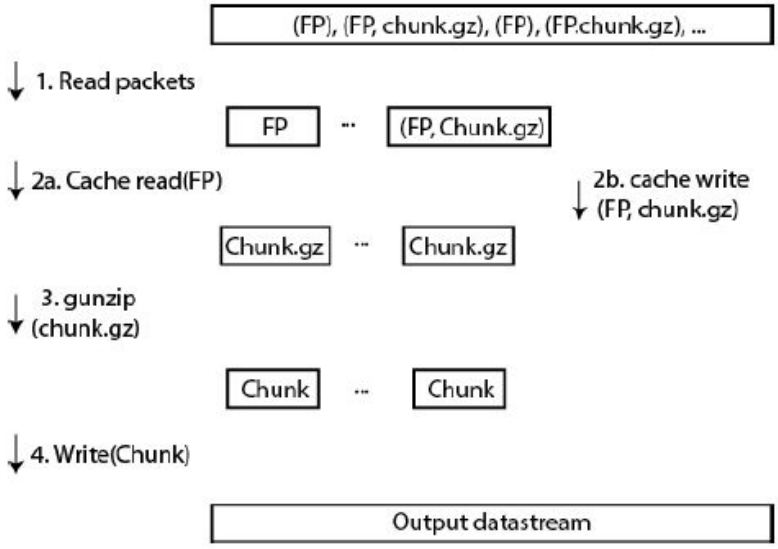

Figure 2: Receiver side

If the receiver received **fingerprint** from the sender it does the following:

1. Read the compressed chunk data from the cache, using the fingerprint as index.
2. Decompress the chunk data.
3. Write the chunk data to the output datastream

If the receiver received a **fingerprint** and **compressed** chunk data it does the following:

1. Write the compressed data to the cache using the fingerprint as index.
2. Decompress the chunk data.
3. Write the chunk data to the output datastream

Input data

You will use the UniProt database, which is available here:

http://ftp.ebi.ac.uk/pub/databases/swissprot/release/uniprot_sprot.dat.gz
http://ftp.ebi.ac.uk/pub/databases/swissprot/release/uniprot_trembl.dat.gz

Note that the sprot file is much smaller than the trembl dataset. Also note that these are compressed. We are currently working on storing these on a local server which you will have access to.

You need to make a model for how you will access the data and how much time this will take. This model should take into account the dataset size, network bandwidth, and other students.

You can use an alternate dataset, or a synthetic dataset. If you do, your report must discuss about the workload selection and workload properties.

Chunking

The UniProt data is structured into records. Code that you can use to split the file into these records is provided in parser.go.

Fingerprints

The fingerprints should be 160-bit SHA-1 hashes. Using such large hash values ensures that there will be no collisions.

Cache

We assume that the cache can hold all non-redundant chunks and that it fits in DRAM. However, the actual size of the cache may be too large for the computers you have available. If that is the case you must simulate a cache.

The cache should be indexed using the SHA-1 fingerprint, and contain chunks. To reduce the memory requirements for the cache, we recommend compressing the chunks with for example gzip. However, you may store uncompressed data, use a faster compression algorithm, or an algorithm with better compression ratio.

The cache should support the following operations:

1. Read(fp): read the chunk with fingerprint FP.
2. Write(fp, chunk): create a cache entry with fp as index and chunk as data.
3. Lookup(fp): check if the cache contains an entry for the fp.

The cache should support concurrent reads and writes.

Local compression

You should compress the data before sending over the network using a local compression algorithm (see http://golang.org/pkg/compress/).

Protocol

You need to design a protocol for sender-receiver communication. The protocol may send chunks out of order, but it is expected that the input datastream and output datastream are identical.

Compression engine

You should implement a concurrent compression engine using Go. Please use available libraries.

Evaluation

You should do a performance evaluation of your system. To do this you must set goals, select metrics, instrument the code, design the experiments, and report the results.

Summary

The following should be done, as discussed above:

1. Create a model for accessing the dataset.
2. Model, design and either implement or simulate the chunk cache.
3. Implement deduplication using SHA-1 fingerprints and local compression
4. Design a protocol for sending fingerprints and chunk data.
5. Implement a concurrent compression engine in Go.
6. Conduct a performance evaluation of your system.
7. Write a report that discusses your models, design, simulation (if any), implementation, experiment methodology and experiment results.

Practicalities

The assignment will be done in groups of two or alone. If done in group, one student will build and evaluate the sender, and one student the receiver. All students must submit an individual report.

Start date: 20.09.2016

Due date 11.10.2016

All code and report will be handed in using Github at your private repositories.

Related work

Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. https://djw.cs.washington.edu/papers/spring-sigcomm00.pdf

Athicha Muthitacharoen, Benjie Chen, and David Mazières. A Low-bandwidth Network File System. 2001. https://www.cis.upenn.edu/~bcpierce/courses/dd/papers/lbfs.pdf

Benjamin Zhu, Kai Li, Hugo Patterson. Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. 2008.
https://www.usenix.org/legacy/event/fast08/tech/full_papers/zhu/zhu.pdf