Mandatory assignment 2 – infm 2202 – Fall 2015

Lars Ailo Bongo (larsab@cs.uit.no)
Ibrahim Umar (ibrahim.umar@uit.no)
Department of Computer Science, University of Tromsø.

Last updated: 17.09.2015.

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 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

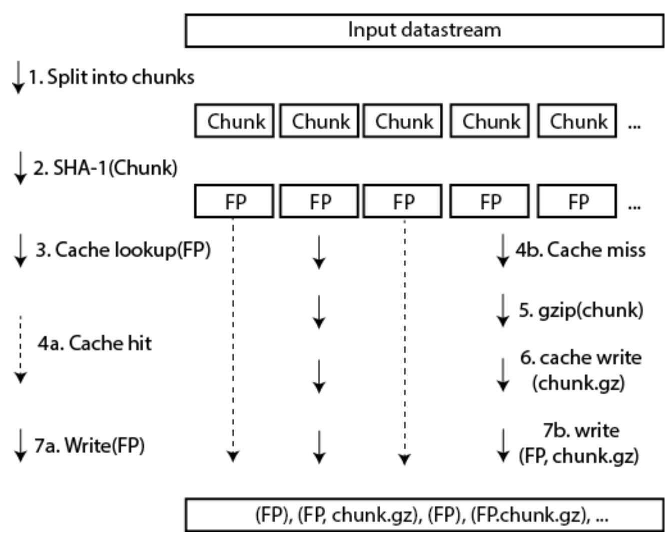

Figure 1: Sender side.

The sender should do the following to compress an input datastream (Figure 1):

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.

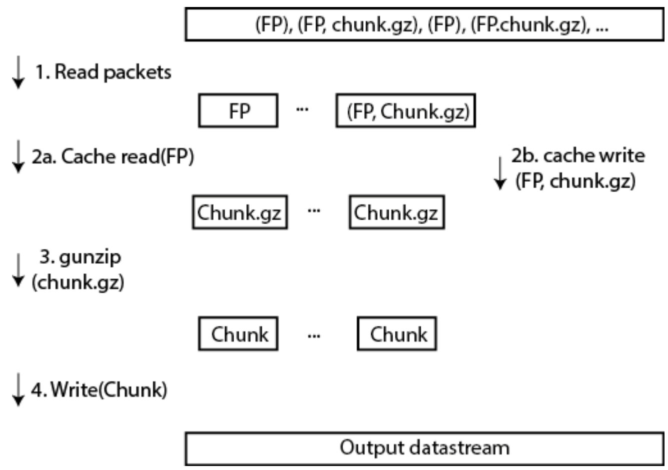

Figure 2: Receiver side.

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 get access to 14 versions of the UniProt database. These are available on:

ifilab102.stud.cs.uit.no:/data/inf2202
(note: use *sftp* or *scp* to access the files and **do not** run your code on ifilab102)

Note that *sprot* file is about 2.3GB, and that the *Trembl* files range in size from 30GB to 47GB. You therefore 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 may use an alternative dataset, or a synthetic dataset. If so, 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 (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:

- Read(fp): read the chunk with fingerprint FP
- Write(fp, chunk): create a cache entry with FP as index and chunk as data
- Lookup(fp): check if the cache contains an entry for 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 fingerpints and chunk data.
5. Implement a concurrent compression engine in Go.
6. Conduct a performance evaluation of your system.

7. Write a report that discuss your models, design, simulation (if any), implementation, experiment methodology, and experiment results.

Practicalities

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

Start date: **17.09.2015**

Due date: **12.10.2015**

Reports and code are handed in using GitHub.

Related work

- Neil T. Spring and David Wetherall. 2000. A protocol-independent technique for eliminating redundant network traffic. SIGCOMM Comput. Commun. Rev. 30, 4 (August 2000), 87-95. DOI=10.1145/347057.347408 http://doi.acm.org/10.1145/347057.347408
- Athicha Muthitacharoen, Benjie Chen, and David Mazières. 2001. A low-bandwidth network file system. In Proceedings of the eighteenth ACM symposium on Operating systems principles (SOSP '01). ACM, New York, NY, USA, 174-187. DOI=10.1145/502034.502052 http://doi.acm.org/10.1145/502034.502052
- Benjamin Zhu, Kai Li, and Hugo Patterson. 2008. Avoiding the disk bottleneck in the data domain deduplication file system. In Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08), Mary Baker and Erik Riedel (Eds.). USENIX Association, Berkeley, CA, USA, , Article 18 , 14 pages.