CSCI4180 (Fall 2021)

Assignment 3: Deduplication

Due on December 16, 2021, 23:59:59

1 Introduction

This assignment is to implement a simple storage application using deduplication. The deduplication is based on Rabin fingerprinting.

2 System Overview

You're going to implement a single *Java* program called *MyDedup*. MyDedup supports the following operations: upload, download, and delete.

2.1 Upload

The upload operation includes the following functions:

- *Chunking*. It reads in the pathname of a file and divides the input file into chunks using Rabin fingerprinting.
- *Identifying unique chunks*. Only unique data chunks will be uploaded to the cloud. We use a fingerprint index to record and identify unique chunks.

2.1.1 Procedures

To upload a file, MyDedup first loads a metadata file named `mydedup.index`, which specifies the fingerprint index that keeps all fingerprints of currently stored files and identifies the unique chunks. It then uploads the unique chunks to the cloud. It also updates the fingerprint index with the fingerprints of the new unique chunks. Finally, the up-to-date fingerprint index will be stored in the index file again for the next run, and the program quits.

We make the following assumptions.

- The file may be a binary file (e.g., VM image) of size up to 100 MiB.
- The *mydedup.index* file and all file recipes are stored locally. If *mydedup.index* does not exist, MyDedup starts with an empty index structure. The formats of the index file and file recipes are up to your choice.
- After a file is uploaded, the file will be immutable (no modification), but it may be deleted.
- We assume that there is no crash of the client or the cloud. We also assume that only one client is using the cloud at any time (so there is no synchronization issue).

- We identify files using their upload pathnames. Different uploaded files must have different pathnames unless they are deleted.
- Files are always uploaded/downloaded in units of *containers*. All unique chunks to be uploaded are packed into a container of size up to 1 MiB. Specifically, the client maintains an in-memory buffer. It adds each new unique chunk to the buffer. If adding a new unique chunk causes the buffer to go beyond container size limit, the client flushes all chunks in the buffer as a new container and uploads the container to the cloud. Note that after the client reaches the end of a file, it should always upload all chunks in the buffer as a new container to the cloud.

After each file upload, you should report the statistics. Note that the statistics are cumulative (i.e., including all files that have been stored). Specifically, we report the following details (excluding the metadata statistics):

- Total number of files that have been stored
- Total number of pre-deduplicated chunks in storage
- Total number of unique chunks in storage
- Total number of bytes of pre-deduplicated chunks in storage (denoted by s_1)
- Total number of bytes of unique chunks in storage (denoted by s_2)
- Total number of containers in storage
- Deduplication ratio: s_1/s_2 (rounded to two decimal places).

2.1.2 Chunking

We use Rabin fingerprinting for variable-size chunking; see lecture notes for details. In particular, we divide chunks by checking if an offset matches an *anchor-point criterion*. An anchor point is determined by the *anchor mask* with multiple 1-bits. If the bitwise AND operation of the Rabin fingerprint and the anchor mask is equal to zero, we have an anchor point.

A data chunk is defined by the byte range starting from the first byte right after the previous anchor point (or the beginning of the file) to the end of the current anchor point. While we reach the end of the file, we simply produce a chunk between the last anchor point and the end of the file, even though the length of the new chunk can be very small.

MyDedup takes the following parameters from the command line as input parameters: (i) `min_chunk`, the minimum chunk size (in bytes), (ii) `avg_chunk`, the average chunk size (in bytes), (iii) `max_chunk`, the maximum chunk size (in bytes), (iv) d, the multiplier parameter. Each chunk size parameter is required to be a power of 2; please return an error otherwise.

Chunks are identified based on SHA-256.

2.2 Download

Given the pathname, MyDedup retrieves chunks (in containers) and reconstructs the original file.

2.3 Delete

Given the pathname, MyDedup deletes the file from the cloud. If a deleted chunk is no longer shared by any other file, its entry in the index structure should be removed. However, its physical copy may remain in the original container. If all chunks in a container are deleted and not shared by any other file, the container should also be physically removed from the storage backend.

2.4 Cloud Storage backends

Your program must support two types of cloud storage backends: local and Windows Azure. In each operation, you specify an option (`local` or `azure`) in the command-line argument to specify which type of storage backend is used. For local storage, all data chunks will be stored under the directory `data/`; for Azure storage, the data chunks will be remotely stored. You will use Azure APIs to access the remote storage. The TAs will give you details on how to use the APIs.

Note that we only apply deduplication independently to each type of storage backends. There is no deduplication across different storage backends.

2.5 Sample Input/Output Format

Upload:

```
java MyDedup upload <min_chunk> <avg_chunk> <max_chunk> <d> \
    <file_to_upload> <local|azure>

Report Output:
Total number of files that have been stored:  10
Total number of pre-deduplicated chunks in storage:  10000
Total number of unique chunks in storage:  2300
Total number of bytes of pre-deduplicated chunks in storage: 40960000
Total number of bytes of unique chunks in storage:  9000000
Total number of containers in storage:  13
Deduplication ratio: 4.55
```

Download:

```
java MyDedup download <file_to_download> <local_file_name> \
     <local|azure>
```

Delete:

```
java MyDedup delete <file_to_delete> <local|azure>
```

After executing each command, the program will quit. Please make sure to properly store the latest metadata in `mydedup.index` for the next run.

3 Miscellaneous

Note that the program should be completed within a reasonable time for a given dataset. The time limit is set by the TAs. Marks will be deducted if the program runs too long.

The tutorial notes may provide additional requirements. Please read them carefully. However, the specification may not make specific requirements for some implementation details. You can make assumptions and give justifications if necessary.

Bonus (5%) The top 3 groups who have the shortest total upload and download time for their programs will receive the bonus marks. We will test the local backend only, so as to eliminate the network transmission overhead. You may use different techniques (e.g., elegant index structures, parallelizing fingerprint computations) to speed up your program. To get the bonus marks, you must first correctly implement all the upload/download/delete operations.

3.1 Submission Guidelines

You must at least submit the following files, though you may submit additional files that are needed:

- Makefile
- MyDedup.java
- Declaration form for group projects:
  `http://www.cuhk.edu.hk/policy/academichonesty/Eng_htm_files_(2013-14)/p10.htm`

Demo will be arranged on the next day. Have fun! :)