CSCI4180 (Fall 2015)

Assignment 3: Deduplication

Due on December 17, 2015, 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 an index structure to record and identify unique chunks.

2.1.1 Procedures

To upload a file, MyDedup first loads a metadata file named `mydedup.meta`, which specifies the index structure 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 index structure with the fingerprints of the new unique chunks. Finally, the up-to-date index structure will be stored in the metadata 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).
- The metadata file is stored in the same working directory as MyDedup; if no metadata file is available, then MyDedup starts with an empty index structure. The designs of the metadata file format and the index structure are up to your choice.
- After a file is uploaded, the file will not be updated.
- We identify files using their upload pathnames. Different uploaded files must have different pathnames unless they are deleted.

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:

- Total number of chunks in storage (including all files that have been stored)
- Number of unique chunks in storage (including all files that have been stored)
- Number of bytes in storage with deduplication (only data chunks are counted) (denoted by s_1)
- Number of bytes in storage without deduplication (only data chunks are counted) (denoted by s_2)
- Space saving: $1 - s_1/s_2$.

2.1.2 Chunking

We use Rabin fingerprinting for variable-size chunking. Please refer to 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, then 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 base parameter. Each chunk size parameter is assumed to be a power of 2.

Chunks are identified based on SHA-1.

2.2 Download

Given the pathname, MyDedup retrieves chunks from the cloud and reconstruct the original file.

2.3 Delete

Given the pathname, MyDedup deletes the file from the cloud. Also, if a chunk is no longer shared by any other file, it should be physically removed from the cloud and its entry should also be removed from the index structure.

2.4 Storage backends

Your program must support two storage backends: local and remote. In each upload, you specify an option (either `local` or `remote`) 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 remote storage, the data chunks will be stored on Azure. You will use Azure APIs to access the Azure object storage. The TAs will give you details on how to use the APIs.

2.5 Sample Input/Output Format

Upload:

```
java Mydedup upload <min_chunk> <avg_chunk> <max_chunk> <d> \
    <file_to_upload> <local|remote>

Report Output:
Total number of chunks in storage:
Number of unique chunks in storage:
Number of bytes in storage with deduplication:
Number of bytes in storage without deduplication:
Space saving:
```

Download:

```
java Mydedup download <file_to_download>
```

Delete:

```
java Mydedup delete <file_to_delete>
```

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

3 Miscellaneous

Note that the program should be completed within a reasonable timeframe 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

The assignment is due on December 17, 2015, 23:59:59. Demo will be arranged on the next day. Have fun! :)