EEL 5737 Principles of Computer System Design

Homework #3 - Solutions

PART 2

Table of Contents

1. Design for Hierarchical file system using MongoDB

1.1. Introduction

This design describes how the persistent file system is implemented in `NoSQL_FS.py` using MongoDB, a NoSQL database.

Connection to MongoDB is made by using the URL given as an input by the user and the database is created at that location.

1.2. Concept

In this implementation, the File system is stored in MongoDB database as a collection of dictionaries. Each dictionary is specific to a directory or a file. The structure of this file system is as shown below.

```
Database:
            Collection:
                    {                                 #FILE NODE 1
                    "path1" :       "key"
                    "meta"  :       { ….
                                        …}
                    "data"  :       {….
                                      …..}
                    "list_nodes" :  {….
                                      ….}
                                            }         #END OF FILE NODE 1

                    {                                 #FILE NODE 2
                    "path2" :       "key"
                    "meta"  :       { ….
                                        …}
                    "data"  :       {….
                                      …..}
                    "list_nodes" :  {….
                                      ….}
                                            }         #END OF FILE NODE 2 and so on
```

Interface 'UPDATE' is used to insert a new item or update an existing item in the Database and interface 'FIND_ONE' is used to retrieve an item from Database. If the requested item is not present in the database, 'None' is returned to the requesting procedure.

1.3. Retention of Database

Once a database is created and File Nodes added to its collection, they are retained for later. For this purpose, File system is not initialized on every mount i.e., root node (or any other directory/file node) is created and initialized only for the first time when it is created. On subsequent mounts, a check is done to see whether the requested path has a node already existing in the database. If yes, then file node is not initialized again. In this way, we can retain the file system even after unmount.

2. Design for Cache Based File system

2.1. Introduction

To implement a Cache based File system in Cached_FS.py, where the File system actually exits in MongoDB database and a set of most recently used files/directories are also located in Cache for faster access.
MemCache is used for this implementation which provides the interfaces for storing data in cache and for retrieving from Cache.

2.2. Concept

Memcache provides ‘set’, ‘replace’, ‘get’, and ‘delete’ interfaces for adding new entry into cache, Updating an existing entry, retrieving an entry and removing an entry in cache respectively. These interfaces are used in current implementation.

The cache size is configurable by modifying ‘Max_CacheSize’ variable which has a value of 10 by default. This implies that Cache can hold contents of 10 files/directories.

A list (with name Cache_Files) is used which holds the paths of directories/files whose contents are available in Cache. This is necessary for implementing LRU policy. The contents are placed in List in such a way that most recently used item is placed in the front and least recently used item at the end. This makes it easier to remove Least recently used item from the list.

When ‘put’ interface is invoked, the path is searched in Cache_Files list for its presence in Cache.
- If present, the contents in cache are modified with incoming information and Cache_Files is updated to have this item at the beginning of the list.
- If not present, a new item is added in Cache and least recently used item is removed both from Cache_Files and Cache.

Then, the contents are also written in database for synchronization

When ‘get’ interface is invoked, the path is searched in Cache_Files.
- If present, the contents are returned to the caller and Cache_Files is modified to have the item at the beginning of the list.
- If not present, the contents are fetched from database, added to cache as most recently used item and then returned to caller.

2.3. Algorithm/ Pseudocode

```
Put (key, value):
    Cache_put(key, value):
        If Filenode available in Cache:
            Modify File node with new value for Key
            Move this file node to the beginning of Cache_List (MRU)
        Else:
            If Cache is Full:
                Remove the Least Recently used filenode from Cache
            Add this file node to the beginning of Cache (MRU)

    Db_put(key, value):
        Add/Update the filenode in database.

Get(key):
    Cache_get(key):
        If Filenode available in Cache:
            Move the Filenode to the beginning of Cache_List
            Return the value of key from file node
        Else:
            Db_get(key):
                Retrieve the filenode from database
            Add the filenode to the beginning of Cache_List
            Return the value of key from file node
```

3. Test Specification and Report

For testing Cached file system, Cached_FS.py is mounted onto fusemount on linux machine.

```
   jai@ubuntu:~/fusepy/$ python Cached_FS.py mongodb://localhost:27017/ 7.0.0.1:11211
```

A shell script is written with a series of commands to perform all the file /directory operations.

The improvement in time to access is noted for file when available in Cache vs when it is to be retrieved from database.

The improvement observed with cache was about 5-10 times of that without cache.

Apart from that basic functionalities are tested as below

```
jai@ubuntu:~/fusepy/fusemount$ ls
jai@ubuntu:~/fusepy/fusemount$ mkdir dir1
jai@ubuntu:~/fusepy/fusemount$ ls
dir1
```

```
jai@ubuntu:~/fusepy/fusemount$ mkdir dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount$ ls
dir1
jai@ubuntu:~/fusepy/fusemount$ mkdir dir2 dir3 dir4 dir5 dir6 dir7 dir8 dir9 dir10
jai@ubuntu:~/fusepy/fusemount$ ls
dir1  dir2  dir3  dir4  dir5  dir6  dir7  dir8  dir9  dir10

jai@ubuntu:~/fusepy/fusemount$ cd dir1
jai@ubuntu:~/fusepy/fusemount/dir1$ mkdir dir1.2
jai@ubuntu:~/fusepy/fusemount/dir1$ mkdir dir1.3
jai@ubuntu:~/fusepy/fusemount/dir1$ ls
dir1.1  dir1.2  dir1.3

jai@ubuntu:~/fusepy/fusemount/dir1$ cd ..
jai@ubuntu:~/fusepy/fusemount$ echo "hello">hello.txt
jai@ubuntu:~/fusepy/fusemount$ ls
dir1  dir2  dir3  dir4  dir5  dir6  dir7  dir8  dir9  dir10  hello.txt

jai@ubuntu:~/fusepy/fusemount$ cat hello.txt
hello

jai@ubuntu:~/fusepy/fusemount$ echo "examples of memory">dir1/dir1.1/examples.txt
jai@ubuntu:~/fusepy/fusemount$ cd dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
examples.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ mkdir dir1.1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  examples.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cat examples.txt
examples of memory

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ mv examples.txt memorytypes.txt
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  memorytypes.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cat memorytypes.txt
examples of memory

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1  memorytypes.txt

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ rm memorytypes.txt
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
dir1.1.1

jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ cd ..
jai@ubuntu:~/fusepy/fusemount/dir1$ cd ..
jai@ubuntu:~/fusepy/fusemount$ rmdir dir1/dir1.1/dir1.1.1
jai@ubuntu:~/fusepy/fusemount$ cd dir1/dir1.1
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$ ls
jai@ubuntu:~/fusepy/fusemount/dir1/dir1.1$
```