Metadata Management in File System

COEN283 – Term Project

Kaushik Krishnakumar(kkrishnakumar@scu.edu), Srinivas Maram Reddy(smaram@scu.edu), Nayan Deep Vemula (nvemula@scu.edu)

ABSTRACT

Implement a modular metadata system on existing file system stack for quick search, storage and retrieval of data present in different types of files. Current Operating Systems do not support such functionality. Only file specific applications like Picasa, iTunes, and iPhoto support searching specific file types. From a user perspective, it becomes very important as the number of photos, songs and documents we store on our disks increase. A disk search also is very slow. The current proposal will allow large number of data to be searched and retrieved based on metadata in a very short time. The current operating systems do support the current feature (like OS X) with the support of tags, but the functionality is not available at a file system level. We will add programmatic API support to add metadata information on the file system such that extensions can be written around these APIs to extend the addition/deletion of attributes to a file system entity. The access times will be compared with existing filesystem's metadata access time.

PREFACE

Acknowledgements

Table of Contents

1. Introduction

1.1. Objective

To introduce metadata management functionality into local filesystem such that it eliminates domain specific metadata management applications.

1.2. What is the problem

Current file systems are designed for large file transfers and not for metadata querying. Nowadays, metadata access is much more frequent than file transfers. There is a need for quick search, storage and retrieval of data present in different types of files. Some user applications like iTunes, Picasa offer this functionality on some OSes. There are also domain specific applications. But they are not at file system level.

1.3. Why this is a project related to this class

File systems are one of the most important Operating System Concepts.

1.4. Why other approach is no good

Domain specific applications currently manage metadata. These applications build a user-level metadata management system that crawls the file system periodically to extract metadata. The data is situated outside the mainline metadata modification path, not real time and which can result in stale query results due to inconsistent metadata

1.5. Area or scope of investigation

a) Perform Background research on existing solutions for metadata in file systems
b) Understand the performance of different file systems for querying Metadata
c) Compare and analyze about implementing the metadata file system in Kernel Space and User Space.

2. Theoretical bases and literature review

2.1. Related research to solve the problem

[1] Deals with implementing the metadata management as part of a new file system stack.
[2] and [3] discuss the implementation of metadata using a NoSQL based database

2.2. Your solution to solve this problem

In our approach, we will be integrating the metadata directly with the file system instead of a domain specific application. Our approach will be similar to a file system API extension.

2.3. Where your solution different from others

[1] advocates building an entirely new stack for filesystem and metadata management. We feel that it is impractical to migrate to an entirely new filesystem stack.
Domain specific applications do not provide realtime data and performance comparable to file systems. The metadata data store is also maintained outside the filesystem.
In our approach, we will be using a NoSQL datastore that is embedded into the existing filesystem.

2.4. Why your solution is better

Metadata management will be provided either through wrapper CLI or enhancement to existing Filesystem APIs. This ensures backward compatibility.

3. Hypothesis

3.1. Positive/negative hypothesis

Our Goal is to implement a metadata management system that is closely integrated to existing file systems thereby ensuring high speed, scalability and backward compatibility with existing data.

4. Methodology

4.1. How to generate/collect input data

We will be writing a script to generate random file types and metadata for the files.

4.2. How to solve the problem

The metadata management will co-exist with any existing file system. To achieve this, we will adopt a layered approach for the metadata operations.

The functionalities are divided as follows

a) Command Abstraction Layer

The Command Abstraction Layer takes the input as the CLI input and convert the command according to the underlying filesystem.

b) Metadata Engine

The metadata engine contains plugin modules respective to each filesystem that is supported for metadata management. Each Plugin module contains 2 parts.

i) Splitter – Used to split the data into Filesystem's native metadata management eg: INODE

ii) Aggregator – Used to aggregate the data from filesystem and Database

c) Database

The local database is a REDIS database. Redis is a data structure server that works with an **in-memory dataset**. Depending on your use case, you can persist it either by dumping the dataset to disk every once in a while, or by appending each command to a log. Persistence can be optionally disabled, if you just need a feature-rich, networked, in-memory cache.

4.3. Algorithm/Block design

The following Metadata Commands are supported

1. Add Attribute – Adds an attribute to the file or directory

```
addattr <FILE OR DIRECTORY> <ATTRNAME> <ATTRVALUE>
```

Example:

```
addattr system-backup.bin filetype DATA-BACKUP
```

2. Show All Attributes

```
showattr system-backup.bin
```

Example:

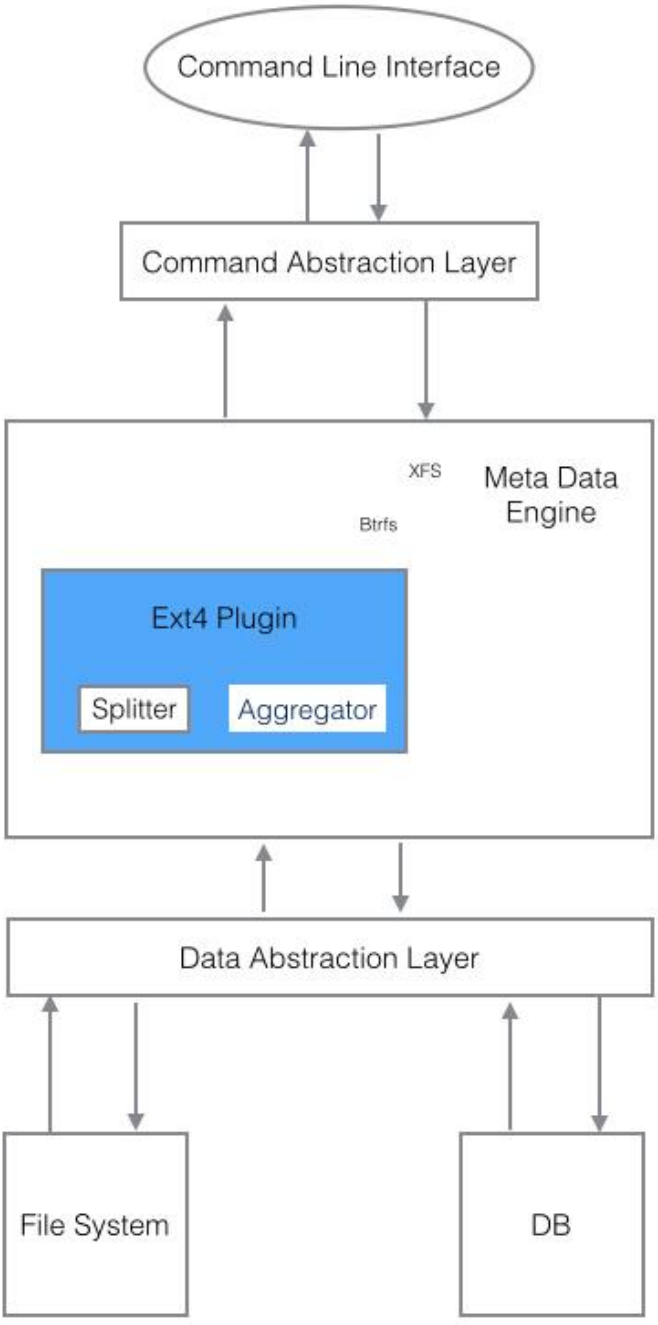
```
showattr system-backup.bin
```

3. Remove Attribute

```
rmattr <FILE OR DIRECTORY> <ATTRNAME>
```

4. Metadata Query Language

```
findattr  <ATTRNAME>[AND OR] <ATTRNAME>[AND OR]….
```



4.4. Language used

- Java
- Unix Shell Script

4.5. Tools used

- libFUSE
  - Filesystem in Userspace libraries that allows to implement a fully functional filesystem in a userspace program
- fuse-jna
  - fuse-jna is a FUSE API library that uses Java Native Access to communicate with the libFuse
- Redis NoSql Database
  - Redis is an open source, BSD licensed, advanced **key-value cache** and **store**.

4.6. How to generate output

Output is generated via automated scripts, which internally invoke the commands that have been defined in the above sections.

Creating files:

In order to load test, the below program creates the files within the Mirror File System.

To Speed up the operations, the program is multithreaded. Create 100000 files in 10 Threads

```
java -cp build/libs/\* coen283.p3.CreateFiles fstest
target/single-thread-files 100000 10
```

Adding Metadata:

OSX:

Using '**xattr'** command

```
# xattr –w <attribute-name> <attribute-value> <filename>
```

Linux:

Using '**setfattr'** command

```
# setfattr -n <KEY> -v "<VALUE>" <FILE>
```

4.7. How to test against hypothesis

The metadata access time in traditional file systems is compared with the metadata management system that we have implemented.

File Creation times are recorded from MAC OSX and Linux. In both Operating systems, the test cases that are executed are

1. Create 100,000 files on Native File System
2. Create 100,000 files on Metadata File System
3. Measure Time taken for Adding attributes to 10,000 files
4. Measure time taken for retrieval of attributes from 10,000 files.

The System Parameters used for testing are

OSX – 10.10/ 2.6 GHz Intel Core i5 / 16GB RAM 1600 MHz DDR3/ 128GB SSD

Linux – Ubuntu 14.14 / Virtual Machine / 8GB RAM / 20GB Virtual HDD

5. Implementation

5.1. Code (refer programming requirements)

The Program is submitted using 'Submit' script as P3 project.

The Program can be run on a linux system with the following packages installed
1. libfuse
2. redis-server
3. openjdk-7 or Oracle JDK 7
4. fattr – Command to process extended arguments

'Autotest' and 'stdin' would not apply to the project as the Design Center Lab workstations do not support libfuse and redis database.

Compiling Source Code:

The source code compiles using gradle wrapper. To compile execute the following command.

```
# ./gradlew rtJar
```

This generates a single JAR file with all the required libraries.

The jar file is generated at **build/libs/coen283-p3-t6.jar**

RUN the file system:
1. Create the mount point
```
# mkdir target/mirrorFS
```
2. Launch the filesystem
```
# ./run.sh
or
# java -cp build/libs/\* coen283.p3.MetadataFileSystem
```

The Source Code is organized as follows. It is written in JAVA

Source Folder:

- `src/main/java` – File System Implementation
- `run.sh` – Start the filesystem mounted in *'target/mirrorFS'* folder
- `./create_file.sh` – Output Generation Script to Create files (Single threaded)

File System Implementation:

The filesystem is implemented in the class **`coen283.p3.MetadataFileSystem`**

Metadata Management Layer:

The metadata management layer interfaces the filesystem and the metadata storage. It is implemented in the class **`coen283.p3.MetadataManager`**.

Data Access Layer:

The DB Connection manager module is implemented in the class **`coen283.p3.ConnectionManager`**

5.2. Design document and flowchart

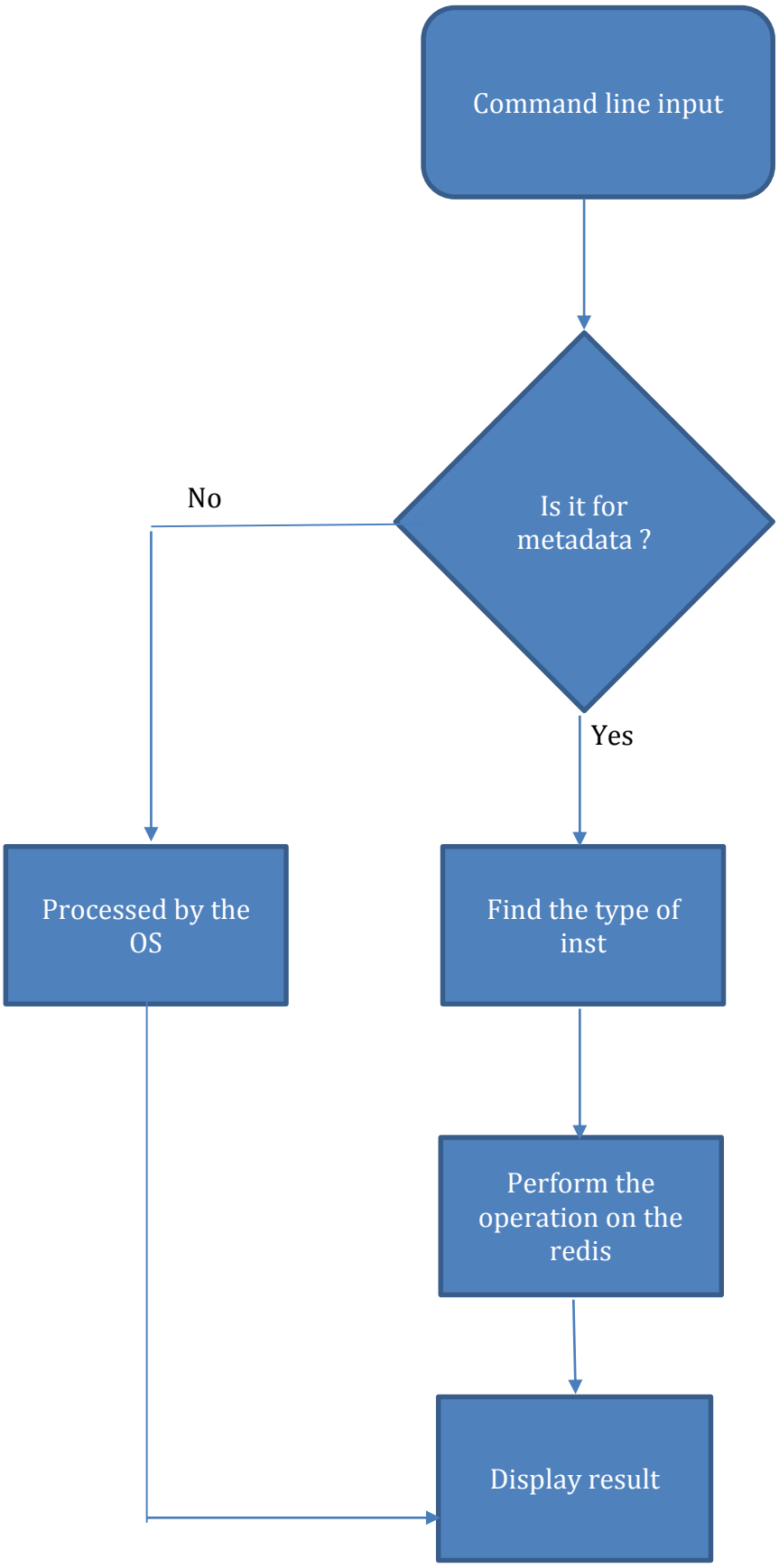
The following are the steps involved.

- The Command is given as an input in the CLI .This command hits the MetaData filesystem which we have mounted using libFuse.

- Here we make a decision whether to redirect it to the OS or the Metadata Engine. If it is Xattr based instruction then we go with the MetaData Engine.

- Based on the decision if the option is MetaData Engine then we go to the module which performs the task for that particular command and display the results back to the Command line.

- Repeat the Steps for the next command.


Command line input
Is it for metadata ?
No
Yes
Processed by the OS
Find the type of inst
Perform the operation on the redis
Display result

6. Data analysis and discussion

6.1. Output generation

Verifying from Command Line:

OSX:

Using **'xattr'** command.

```
# xattr –l target/mirrorFS/mf4*.txt
```

This displays the extended attributes that are added for all the files matching with 'f1*.txt'

Linux:
Using '**getfattr'** command

```
# getfattr -n owner target/mirrorFS/mf4*
# getfattr -n purge-date target/mirrorFS/mf4*
```

Verifying from Database:

The data can also be verified from the Metadata store DB

1. List All the files tagged with owner 'kaushik'

```
redis-cli> LRANGE owner::kaushik 0 -1
```

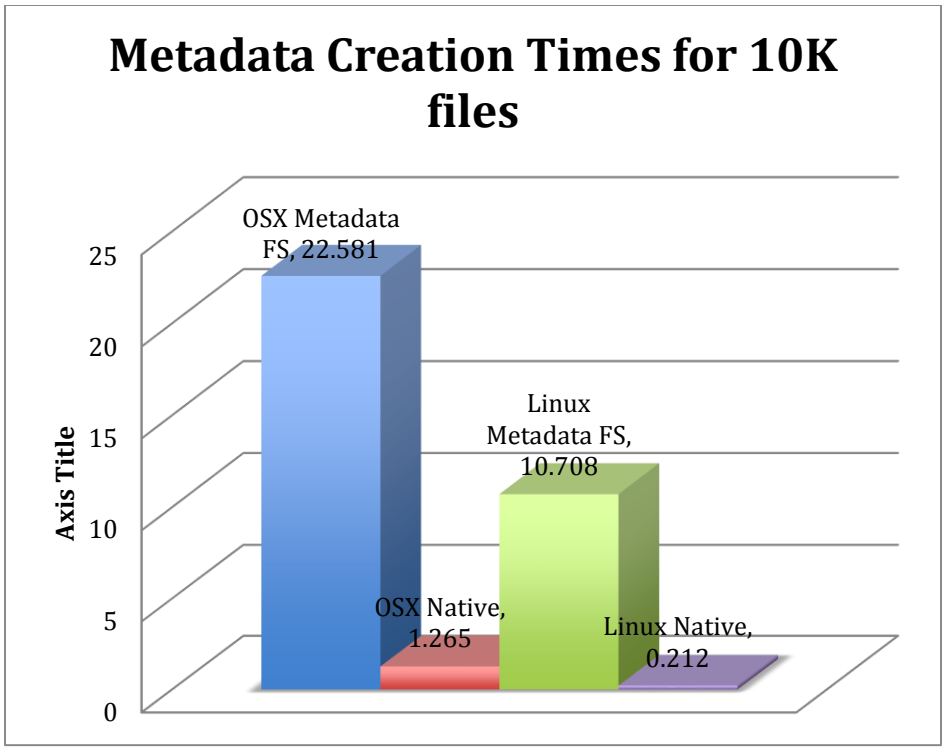
2. List All Files with purge-date '14-dec-2014'

```
Redis-cli> LRANGE owner::kaushik 0 -1
```

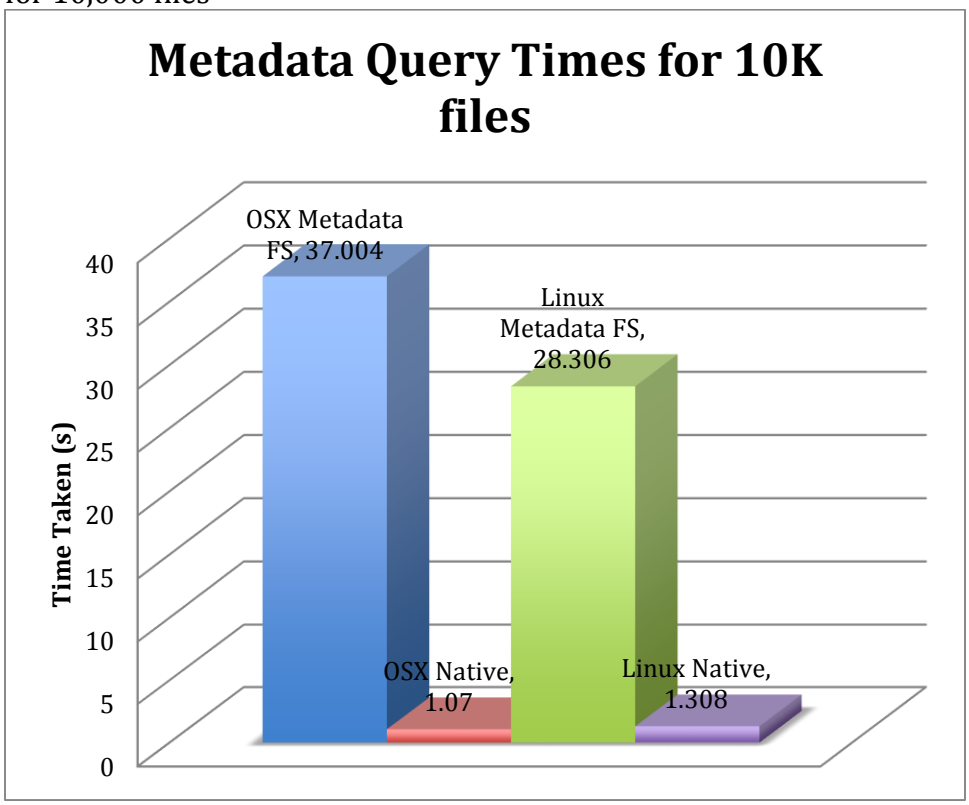
6.2. Output analysis

The below comparison uses the commands native to OS (xattr and fattr) to measure the metadata access/creation times.

The comparison of Metadata Creation times between Native Filesystems (OSX and Linux) and Metadata FileSystem is presented below. The data was measured for 10,000 files



The comparison of Metadata Retrieval times between Native Filesystems (OSX and Linux) and Metadata FileSystem is presented below. The data was measured for 10,000 files



6.3. Compare output against hypothesis

The Metadata Filesystem has been implemented using FUSE Libraries, REDIS Database. The file system has been run on OSX and Linux. The attribute creation times, retrieval times have been compared.

Backward compatibility has also been ensured with our current implementation. The metadata filesystem is implemented as a Mirror Filesystem. It uses the OS's native file system for all the operations except the callback for **setxattr, getxattr, listxattr** method. Hence the metadata file system can be used even for an existing file system and used as a separate metadata store.

6.4. Abnormal case explanation (the most important task)

From the charts shown in Section 6.2, the Metadata Filesystem performs poorly in data add/retrieve operation compared to Native File Systems. The reason can be attributed to two main reasons. The FUSE library is inherently slow as it adds an extra layer of indirection. The Java Interface that is used to communicate with FUSE, fuse-jna is very slow. These inferences can be made because, the data access times from REDIS database is less than 1s(0.5s to 0.8s on average). The only other delay is from the underlying Libraries.

The performance on OSX is also 20% slower than on Linux. The reason being OSX uses 'xattr' utility to perform metadata operation. This utility is based on python and the interpreter introduces a certain amount of delay. However, the linux utility 'fattr' performs much faster.

7. Conclusions and recommendations

7.1. Summary and conclusions

We conclude that metadata management functionality into local filesystem such that it eliminates domain specific metadata management applications from building its their own retrieval system . We can see that this works on both the OSx and also linux which implies that it is portable.

7.2. Recommendations for future studies

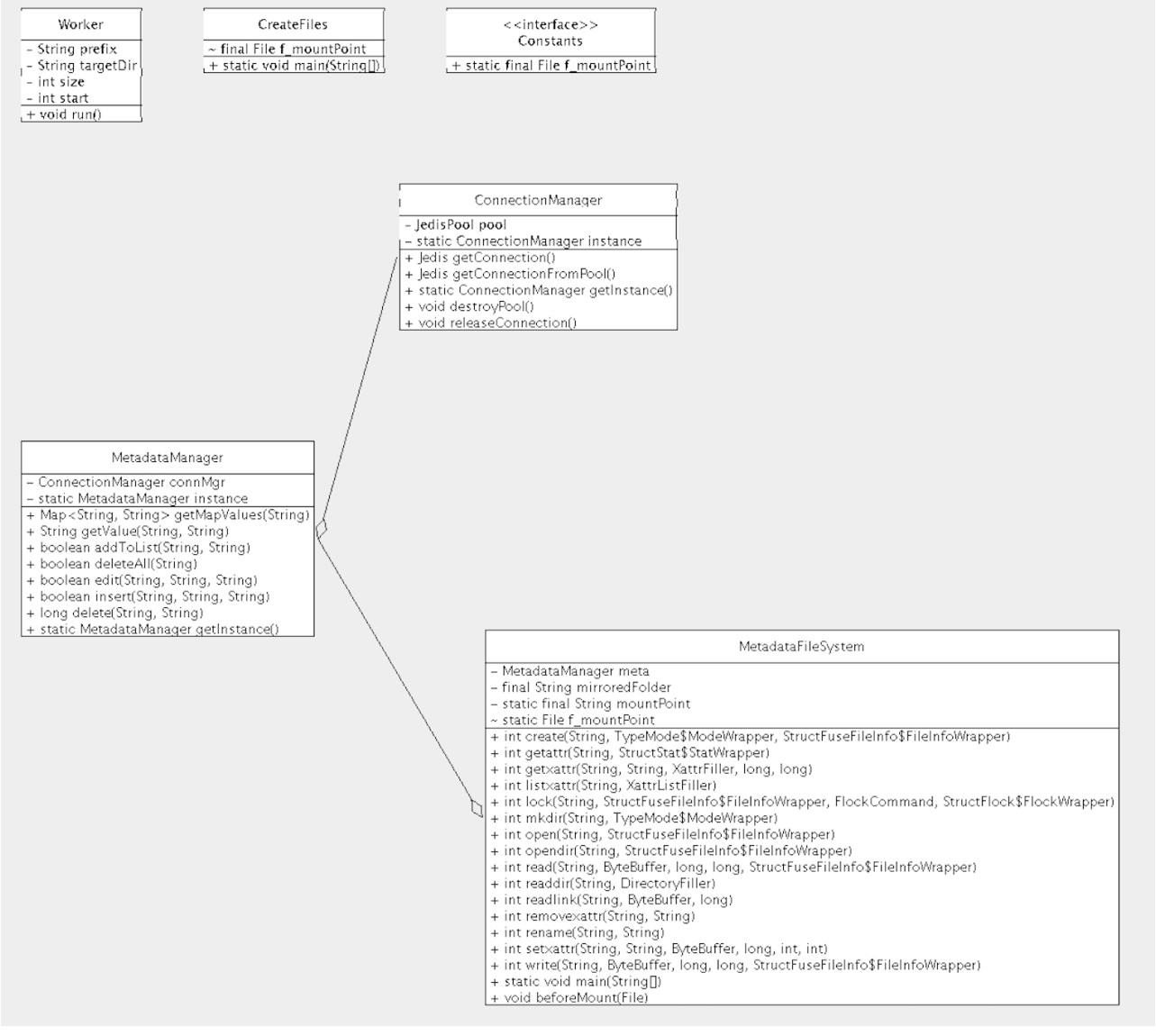
We have seen that we have a long delay due to Fuse filesystem . So, we can check if we can implement it into the kernel space then the results will improve. The Metadata FileSystem can also be used for a group of filesystems maintaining a single redix server.

8. Bibliography

a. van Heuven van Staereling, Richard, et al. "Efficient, modular metadata management with loris." *Networking, Architecture and Storage (NAS), 2011 6th IEEE International Conference on*. IEEE, 2011.
b. Ren, Kai, and Garth Gibson. "TABLEFS: Embedding a NOSQL database inside the local file system." *APMRC, 2012 Digest*. IEEE, 2012.
c. Ren, Kai, and Garth A. Gibson. "TABLEFS: Enhancing Metadata Efficiency in the Local File System." *USENIX Annual Technical Conference*. 2013.

9. Appendices

9.1. Program flowchart



9.2. Program source code with documentation

The Source Code with documentation is hosted in the GIT Repository

https://bitbucket.org/coen283p3/coen283p3/get/8f3a4b93cc11.zip

9.3. Input/output listing \

Linux - Metadata File System

Create 100000 Files using 10 Threads – 3m 33s

```
# time java -cp build/libs/\* coen283.p3.CreateFiles mf
target/mirrorFS/ 100000 10
```

real 3m32.841s
user 1m28.296s
sys 0m15.878s

Add Attribute – 10.7s

```
# setfattr -n owner -v "kaushik" target/mirrorFS/mf*
```

real 0m10.708s
user 0m0.147s
sys 0m0.225s

List Attributes for 10K files – 28s

```
# getfattr -n owner target/mirrorFS/mf4*
```

real 0m28.306s
user 0m0.285s
sys 0m0.525s

OSX – Metadata File System

Add Attribute - 22s

```
# time xattr -w purge-date 14-dec-2015 mf4*
```

real 0m22.581s
user 0m1.012s
sys 0m0.568s

List Attributes for 10K files – 1m 37s

```
# time xattr -l  mf4*
```

real 1m37.004s
user 0m2.382s
sys 0m1.792s

REDIS MEMORY

used_memory:4256832
used_memory_human:4.06M
used_memory_rss:9306112
used_memory_peak:6228424
used_memory_peak_human:5.94M
used_memory_lua:33792
mem_fragmentation_ratio:2.19
mem_allocator:jemalloc-3.4.1

9.4. Other related material

http://www.gnu.org/software/hurd/hurd/libfuse.html