Lab 4: Data Cache

Hand out: 2020 Dec 2nd

Deadline: 2020 Dec 21st 23:59 (GMT+8)

Please read the whole document before writing any code.

In case you have any question for this lab, please contact TA (Li Mingyu) via Email (maxullee@sjtu.edu.cn) or @him in the CSE2020 WeChat room. You are highly welcome to report bugs and help improve the lab test.

Problem Statement

Alice and Bob are two shopaholics who must visit shopping websites for bargains everyday. Since it is impossible to predict which item will be taken in advance, buyers such as Alice and Bob will have a copy of all available items locally. In reality, a shopping item cannot be physically cloned (otherwise it will break the law of conservation of mass!). So an item that Alice has purchased should be not visible to Bob any more. To serve as many customers as possible, the online seller wishes that Alice and Bob should not visit their website server very frequently. In order to serve more concurrent requests, the online buyer may use more physical resources to improve the server performance (e.g., by adding more CPUs).

Getting Started

At first, please remember to save your lab3 solution:

```
% cd lab-cse
% git commit -a -m "solution for lab3"
```

Then, pull the lab4 branch:

```
% git pull
% git checkout lab4
```

Merge with your lab3:

```
% git merge lab3
Auto-merging fuse.cc
CONFLICT (content): Merge conflict in yfs_client.cc
Auto-merging yfs_client.cc
CONFLICT (content): Merge conflict in ifs_client.cc
Automatic merge failed; fix conflicts and then commit the result
...
```

After merging all the conflicts, you should be able to compile the lab successfully:

```
% make
```

If you wish to run this lab on your native environment (e.g., Debian/Ubuntu), you need to install the libfuse-dev package:

```
% sudo apt-get install libfuse-dev
```

Part-1: Improve your YFS with Data Cache

In Lab1++, you have noticed a large performance gap between your own YFS implementation and the native Linux file system. That's fine. Recall that in lab2, you have integrated RPC with YFS and built a lock server. To reduce the RPC operations, you have grasped the power of lock cache. Likewise, in Part-1, you are requested to build another cache---data cache in the client side to improve the performance.

Your goal is to devise your own caching algorithm inextent_client.cc, and make it as efficient as possible.

Hints: the marshall/unmarshall procedures in the RPC library can incur high transmission costs. You are suggested to try best efforts to reduce the amount of RPC traffics as best as you can. Batching can be a good idea.

Test

Use FxMark to profile your original YFS in Lab-1:

```
% git checkout lab1
% make
% sudo ./start.sh
% ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --ncore=1 --duration=10
% sudo ./stop.sh
% make clean
```

Use FxMark to profile your cache-based YFS, and see if client-side data cache helps:

```
% git checkout lab4
% make
% sudo ./start.sh
% ./fxmark/bin/fxmark --type=YFS --root=./yfs1 --ncore=1 --duration=10
% sudo ./stop.sh
% make clean
```

Part-2: Design a Cache Coherence Protocol

In reality, there should be many clients concurrently accessing the same server. So the data cache should be extended to to work correctly between multiple concurrent clients and the one server. For example, when client Alice modifies a data item that client Bob has cached, the server must invalidate the cached data of Bob by sending an RPC to Bob, or Bob must synchronize with the server periodically to prevent himself from using a stale version of the data item.

Your goal is to design a distributed cache coherence protocol. This protocol should be able to deal with such a complicated circumstance where RPC requests and replies may not be delivered in the same order in which they were sent :(

We suppose that you have completed lab1-lab3 on your own, and should have learnt lessons on how to design a qualified cache system (especially from lab2 and lab3). So, good luck and go ahead :)

Test

Step One: Once you have your distributed data cache coherence protocol implemented, you can run it using the grade_lab2.sh script, just as in Lab 2:

```
% export RPC_LOSSY=0
% grade_lab2.sh
```

Step Two: Now that it works without loss, you should try testing withRPC_LOSSY=5. You may encounter problems with reordered RPCs and responses.

```
% export RPC_LOSSY=5
% grade_lab2.sh
```

Part-3: Squeeze More Performance Out of Servers (Optional)

We are going to re-run FxMark benchmarks to measure the performance of your Part-2 implementation.

You are suggested to review Lec-21 and Lec-22 for common optimization techniques, and please feel free to use any powerful techniques you know to help build a high-throughput server.

Hints:

Sharding might be of great help as it partitions data on different servers, given that each server has a limited resources (compute, network, storage). It is intuitive to leverage a consistent hashing algorithm for load balancing between multiple extent servers.

Another bottleneck might come from the big lock in the extent server implementation. You may remove this big lock and add fine-grained locks to avoid contention.

Evaluation Criteria

We are going to run your code on a TA's server and fairly compare their performances one by one (excluding any noisy neighbors).

Correctness Evaluation

Run grade_lab2.sh once you have your distributed data cache coherence protocol implemented. Do not bother testing with loss:

```
% export RPC_LOSSY=5
% export RPC_COUNT=25
% ./grade_lab2.sh
```

Performance Evaluation

Use FxMark to measure your insanely improved YFS:

```
% ./grade_lab4.sh
```

Unlike prior correctness evaluation, we are going to generate a sorted list according to the submitted code performance, namely, the works value. Those who win top 10 will get 1 bonus point in the final, while others remain the same.

Handin Procedure

Like what you did in previous labs, just type:

```
% make handin
```

This should produce a file called lab4.tgz in the directory. Change the file name to your student id:

```
% mv lab4.tgz lab4_[your student id].tgz
```

Please make sure the submitted file is less than 4MB, otherwise it won't be considered. Your commits to Lab-4 will be manually reviewed once it passes the tests on TA's machine.

Then upload lab4_[your student id].tgz to ftp://maxulleepublic@public.sjtu.edu.cn/upload/cse/lab4/ before the deadline.

You are only given the permission to list and create new file, but no overwrite and read. So pleasse make sure your implementation has passed all the tests before the final submit.