Project 4: Paxos-based Key-Value Service

Chen Xiaoqi, Ku Lok Sun, Wu Yijie, Zhang Hanrui

Design

Three servers in library `kvpaxos` runs with underlying `paxos` library.

Paxos

The `paxos` library will create paxos instances, which decide operations for every slot. The decision will contain the data (key/value) and operation type (Put, Update, etc.) and is consistent for majority.

Note: This library has been fully tested using the original paxos test. Use `make test_Paxos` to run the original test.

KVPaxos

The `kvpaxos` library will use its underlying `paxos` instance to achieve consistent database service. For each request (including both KV service and KVMAN service), one decision slot is obtained and the operation is logged into paxos history. All data queries can be recovered from the database log.

In order to improve performance, we used a random-backoff scheme (similar to that in CSMA/CD) and optimized the inter-arrival time of paxos decision queue, such that each decision can be quickly made even if the system is under high load pressure.

The client provide an operation ID optionally, and the server will not repeat operations with the same ID. This will ensure database consistency in the case of server temporary partition and client/server unreliable communication.

Each server will periodically create snapshots of database log (and let paxos forget old decisions), to reduce memory comsumption and improve performance (of scanning the log). The threshold to trigger snapshot update can be modified in the configuration. In order to pass the original memory consumption test, the threshold must be less than 50 (because

there's only about 50 operations in the test).

Note: This library has been fully tested using the original kvpaxos client/server test, before migrating to the tcp-based RPC platform. Use `make test_kvPaxos` to run the test (using Unix socket). (The test has been modified since `PutHash` operation is not included in this project; also, since `Put` operation here has different semantic from `insert` of project 3, it is recorded as `Naive_PUT` rather than `PUT` in the database journal)

RPC Interface

To comply with the multi-machine scenario, we changed the communication between different paxos instances to TCP-based (from Unix-socket based). This does not affect the normal working of paxos, since the RPC and the network transportation is fully layered; however, now we cannot control the partition in our new test cases.

HTTP Interface

Each kvpaxos instance will listen to a HTTP port, to provide the following service:

Data service

Put `/kv/insert` or `/kv/put`

Insert new key into the database; will succeed only if it's a new key. Require the key to be nonempty.

The parameter is provided in `key` and `value` field.

Update `/kv/update`

Update key in the database; will succeed only if it's an existing key. The old value will be returned.

The parameter is provided in `key` and `value` field.

Delete `/kv/delete`

Delete key in the database; will succeed only if it's an existing key. The old value will be returned.

The parameter is provided in `key` field.

Get `/kv/get`

Look up a key in the database; will succeed only if it's an existing key. The value will be returned.

The parameter is provided in `key` field.

Note: Each HTTP request is treated as independent requests, since the HTTP protocol is stateless; if consistency in unreliable network is desired, the client should provide a unique increasing operation ID in `opid` field, and the server will not repeat requests with the same ID or an smaller ID.

Management service

CountKey `/kvman/countkey`

Returns the number of distinct, existing keys in the database.

This operation will succeed only if the server can obtain an agreement (i.e. not partitioned into minority) such that the data is guaranteed to be up to date.

Dump `/kvman/dump`

Returns a list of existing key-value pairs in the database.

This operation will succeed only if the server can obtain an agreement (i.e. not partitioned into minority) such that the data is guaranteed to be up to date.

Shutdown `/kvman/shutdown`

Kills the server and release the listening ports.

Note: We may have to wait at most 10 milliseconds, before the ports are released and a new instance can be started. Listening to the port immediately after shutdown may cause an error. This issue is handled in our server initialization process, by waiting 10ms before starting any port; however it may affect a subsequent group's project if projects of many groups is tested in batch automatically.

Performance

By using appropriate random timeout settings similar to CDMA/CD, we improved the performance of `kvpaxos` in various test cases by approximately 10x faster, compared with simple random timeout implementation. Currently, the server cluster can response requests within approximately 80ms (without partitioning).

The delay and timeout parameters are set primarily for ethernet or same-machine testing, which is similar to the actual enviroment between different machines in one data centers; the performance may suffer, if the delay between machines are larger than design limits. Therefore, if the test is performed in high-latency network (e.g. cross-pacific), the performance might become suboptimal.

Build, Run and Test

Command

The files `compile.sh` , `bin/start_server` , `bin/stop_server` , `bin/test` are as required in Project 3/4.

For remote testing, use the same config files (`conf/settings.conf` and `conf/test.conf`), in each server, run the corresponding auxiliary tester

```
bin/test < n01|n02|n03 > &
```

or

```
bin/test < 1 | 2 | 3 > &
```

Then in any machine, run the main tester:

```
bin/test -m
```

For single-machine testing, run `make test_Paxos` and `make test_kvPaxos` (the original go test) or `make tester` using our tester and test cases in the directory `test/` .

Tester Description

In this project we use a similar tester as in previous Project 3, which initializes HTTP requests to different servers and send different requests according to test case files; the result of each request is automatically deduced from previous request sequence and compared with actual result. The test will be passed only if all results matched.

Normally, the servers are started before each case and shut after the test finishes. Optionally, a test case can shut down one or more servers during the test. In this case the

correct results are also automatically deduced using the majority consensus requirement. Due to aforementioned difficulty, the test case does not implement partition.

To help testing on multiple servers, we implementd auxiliary testers, who runs on different machines and help main tester to start and stop kvpaxos instances. The main tester will wait for all three auxiliary testers until they are alive, and then start testing.

Each test case is specified by the `*.test` file under `test/` as in Project 3. After each case finished, the result as well as the ellaped time will be printed on the screen if the `with_err_mesg` flag is `true` (set in `test.conf`).

The number of test cases (`test_total`) is also set in `test.conf` . If you wait too long you can call `http://127.0.0.1:3086/main?op=finish&forced=< true | false | report >` (modify to the ip of main tester and the port `test_port00` in `test.conf`):

- `forced=report` will allow the tester to finish the current round and report the infomation;
- `forced=true` will immediately shutdown all testers;
- `forced=false` will set timeout and might allow the current test case to finish without reporting.) This hack is helpful in case the tester stuck since we do not time out all the serial request.

We have also carried out the real remote testing on three machines (one in Beijing, one in Tsinghua and one in USA). It took about 40~50 minutes to pass all the ten cases. (While it takes only 1~2 minutes for the same-machine test using out tester.)

Note:

1. We assume the ports in `settings.conf` are available and do not check for that. In case the specified ports are already occupied (partly due to previous failed run), the program may crash.
2. We also assume all the conf file and `.test` file are well-format and do not check for that. So please follow the format if you try to modify these files, otherwise the program may crash or panic. Some existing test cases are generated by python script.
3. As long as the client HTTP operations are the same, this tester can be used to test against other group's program.