存储技术基础 2019 Spring
Key-Value 存储引擎

作业描述

本次作业题目来自阿里第一届 POLARDB 数据库性能大赛。在给定 C++ 代码框架下，实现高效的并发 Key-Value 存储引擎。需实现的接口有:

```
// name为存储引擎的数据路径，初始化存储引擎，返回指针到*eptr
RetCode EngineRace::Open(const std::string& name, Engine** eptr);

// 将<key, value>插入存储引擎，如果key已经存在，则该操作为更新
RetCode EngineRace::Write(const PolarString& key, const PolarString& value)

//根据key在存储引擎中索引数据，返回数据到*value变量
RetCode EngineRace::Read(const PolarString& key, std::string* value)
```

其中 PolarString 是一个封装的字符串类，详见 `include/polar_string.h`。接口定义在 `include/engine.h` 和 `engine_race/engine_race.{cc|h}`。更详细的接口语义可阅读 `test/single_thread_test.cc`。

作业要求与说明

1. 一人一组完成代码实现工作。
2. 基于课程提供的代码框架进行程序开发（修改 `engine_race.{cc|h}` 文件）。该框架在压缩代码包 engine.zip 中，可从网络学堂下载。编译运行环境为 linux，大家自行搭建虚拟机、docker 或 Windows Subsystem for Linux 环境。
3. engine.zip 解压后有如下文件:

```
[wq@node110 engine]$ ls
bench  engine_example  engine_race  include  lib  Makefile  README.md  test
[wq@node110 engine]$
```

其中 engine_example 里是一个参考版本的 KV 存储引擎。engine_race 中包含 `engine_race.{cc|h}`，**这是本次作业中唯一需要修改的两个文件**。test 中包含正确性测试代码，bench 中包含性能测试代码。

4. 实现的 Key-Value 引擎需保证 Crash Consistency：1) Write 成功返回之后，保证该操作插入的键值对被持久化，即使机器崩溃重启后也不会丢失; 2) Write 操作保证原子性。保证 Crash Consistency 的方法有 Write ahead log和 Shadow paging等。大家需要熟悉 linux 文件系统的一些 posix 接口语义，包括 write, fsync和 mmap; 以及 Buffer IO机制和 Direct IO机制。注意: engine_example 中的参考代码没有保证 Crash Consistency。
5. 实现的 Key-Value 引擎需支持多线程并发执行，并保证Linearizability语义。
6. 编译时，执行 `make` 命令（如果需要编译 engine_example 中的参考代码，执行 `make TARGET_ENGINE=engine_example`)。编译完成后在 lib 目录下生成静态链接库 libengine.a。

```
[wq@node110 engine]$ make
make[1]: Entering directory `/home/wq/engine/engine_race'
  CC       engine_race.o
ar: creating /home/wq/engine/lib/libengine.a
a - /home/wq/engine/engine_race/engine_race.o
make[1]: Leaving directory `/home/wq/engine/engine_race'
[wq@node110 engine]$ ls lib/
libengine.a
```

7. 测试正确性时，进入 test 目录，执行`./build.sh` 来编译测试程序，执行`./run_test.sh` 来运行测试程序。现提供的三个测试程序比较简单，`single_thread_test` 测试单线程正确性；`multi_thread_test` 测试多线程情况下的正确性；`crash_test` 测试进程被 kill 后的系统正确性。

```
[wq@node110 engine]$ cd test/
[wq@node110 test]$ ./build.sh
single_thread_test.cc
multi_thread_test.cc
crash_test.cc
[wq@node110 test]$ ./run_test.sh
====================== single thread test ============================
open engine_path: ./data/test-6498242813469646
====================== single thread test pass :) ======================
------------------------------------
====================== multi thread test ============================
open engine_path: ./data/test-6498247307245798
====================== multi thread test pass :) ======================
------------------------------------
====================== crash test ============================
open engine_path: ./data/test-6498266686465760
====================== crash test pass :) ======================
[wq@node110 test]$
```

8. bench 目录中提供了性能测试程序，执行`./build.sh` 来编译，执行`./bench` 来运行测试程序。./bench 程序有三个参数, thread_num 是并发执行的线程个数，read_ratio 是 Read 操作的比例，isSkew 代表 key 的分布 (0 时为均匀分布，1 时为zipfan 分布)。bench 程序中 key 和 value 的大小分别固定为 8bytes 和 4096bytes。

```
[wq@node110 engine]$ cd bench/
[wq@node110 bench]$ ./build.sh
bench.cc
[wq@node110 bench]$ ./bench
Usage: ./bench thread_num[1-64] read_ratio[0-100] isSkew[0|1]
[wq@node110 bench]$ ./bench 2 50 1
thread_num: 2, read ratio: 50%, isSkew: true
open engine_path: ./data/test-6498405357929621
2 thread, 200000 operations per thread, time: 2907969.731000us
throughput 137553.013615 operations/s
[wq@node110 bench]$
```

附加任务（可选）

实现范围查找接口 `Range`.

```
RetCode Range(const PolarString& lower, const PolarString& upper, Visitor &visitor)
```

思考问题（可选）

1. 如何验证或测试 Key Value 存储引擎的 Cash Consistency?

提交说明

作业提交内容包括：

1. 打包后的 engine 文件夹，含有你实现的 Key Value 存储引擎代码。
2. 作业总结报告，包含：
   - 你的姓名和学号
   - 你的 Key Value 存储引擎的设计与实现细节
   - 你的性能测试结果 (不同的线程数目, 不同的读写比例, 不同的 key 分布)，以及必要的性能分析

参考文献

[1] https://github.com/google/leveldb. (基于 LSM Tree 的 KV 存储引擎)

[2] https://github.com/facebook/rocksdb. (基于 LSM Tree 的 KV 存储引擎)

[3] https://fallabs.com/kyotocabinet/. (基于 B+Tree 或 Hashtable 的 KV 存储引擎)

[4] Lu, Lanyue, et al. “WiscKey: separating keys from values in SSD-conscious storage.” Proceedings of the 14th Usenix Conference on File and Storage Technologies. USENIX Association, 2016.

[5] Raju, Pandian, et al.“Pebblesdb: Building key-value stores using fragmented log-structured merge trees.” Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017.