Week#10 Introduction to SQLite and AndroBench

Sangeun Chae
2018314760

1. PROPOSAL

1- Compare with the SQLite and client-server RDBMS

- **Advantage of SQLite**

1) SQLite 는 serverless 한 DB engine 이기 때문에, zero-configuration 한 특성을 지니고 있습니다. 따라서 client-server RDBMS 에 비해 사용하는 공간이 현저히 작고, SQLite 설치과정에 있어서 특정 어플리케이션과의 통합과정이 간소화됩니다.

2) SQLite 는 Virtual Filesystem 을 가지고 있어서, 특정 operation system 에 존재하는 Filesystem 과의 효용성이 문제가 되지 않고, 모든 operation system 위에서 구동할 수 있습니다.

3) 네트워크 환경이 필요 없고, 로컬에서만 구동되는 DB 의 구성에 용이합니다. 따라서 단일 사용자가 사용하는 로컬 애플리케이션에서는 시스템에 heavy load 되는 client-server RDBMS 에 비해, SQLite 가 적합합니다.

- **Disadvantages of the SQLite**

1) 동시적으로 많은 쓰기 작업이 발생했을 경우, SQLite 는 client-server RDBMS 에 비해 효율이 떨어집니다. 그 이유는, 하나의 file 에 데이터를 저장하기 때문에, 오직 하나의 프로세스만이 데이터베이스를 변경할 수 있습니다. 따라서 여러 개의 쓰기 작업이 발생하는 경우, concurrency 제어에 이점을 둔, client-server RDBMS 가 성능적인 측면에서 유리합니다.

2) 많은 데이터를 저장해야 하는 경우, SQLite 에 비해 server-client RDBMS 가 유리합니다. SQLite 는 하나의 disk file 로 데이터를 저장하기 때문에, 하나의 file 에 많은 데이터들이 저장되었을 시, 데이터를 처리하는 과정에서 overhead 가 발생하게 됩니다. 따라서 데이터를 많이 저장해야 하는 workload 에서는 server client RDBMS 가 유리하게 작용합니다.

3) 네트워크 접근이 필요한 경우, SQLite 는 client-sever RDBMS 에 비해 적합성이 떨어집니다. SQLite 는 단일 사용자가 사용하는 로컬 환경에서 최적화된 데이터베이스기 때문에, 데이터에 대한 직접적인 네트워크 접근을 제공하지 않습니다. 따라서 이러한 경우에는, client-server RDBMS 가 적합합니다.

2. How SQLite supports cross-platforms database

다양한 operating system(운영체제)은, 각각의 파일 시스템을 사용합니다. 또한, 하나의 시스템에서도 하나의 파일 시스템만 사용하지 않습니다. 따라서, 특정 데이터가 저장되어 있는 데이터 블록을 읽기 위해서는, 해당 데이터가 저장되어 있는 파일 시스템에 맞게 함수를 호출해야 하는 어려움이 있습니다. SQLite 는 이러한 문제점을 해결하기 위해, VFS(Virtual File System)을 사용합니다. VFS 는 실제 저장되어 있는 공간의 파일 시스템에 접근하기 전에, 가상적인 file system layer 를 통해서, 특정 파일 시스템에 맞는 함수를 사용하게 합니다. 이런 특성에 의해서, SQLite 는 여러 플랫폼에서 사용할 수 있습니다.

3. Compare the SQLite and Filesystem

SQLite 는 하나의 큰 파일에 데이터를 저장하는 구조를 가진다. 하지만 일반적인 파일 시스템의 경우에는, 데이터를 여러 개의 파일에 나눠서 저장하는 구조를 가진다. 따라서 SQLite 의 관점에서, 작은 데이터를 읽는 과정에 있어서는, 파일을 open 하고 close 하는 system call 함수를 호출하는 횟수가 1 번으로 제한됩니다. 하지만 일반적인 파일 시스템에서는, 데이터가 여러 파일에 나눠서 저장되는 경우, 해당 파일의 수만큼 파일을 open 하고 close 하는 system call 함수를 호출해야 합니다. 따라서 system call 함수를 호출하는 오버헤드가 일반적인 파일 시스템에서는 크게 작용됩니다.

2. INTRODUCTION

What is SQLite

SQLite 는 MySQL 나 RocksDB 같은 데이터베이스 엔진이지만, 서버가 아니라 serverless 로 구동되어지는, 비교적 가벼운 데이터베이스 엔진이다. 즉, 대규모 데이터를 저장하는 데이터베이스가 아닌, 임베디드 환경에서 구동되어지는 가벼운 데이터베이스 엔진이다. SQLite 는 다른 server-client RDBMS 와는 다르게, 하나의 파일로 구성되며, 단순한 API 기능만을 제공하는 것이 특징이다. 또한 단일 사용자가 사용하기에 편리한 데이터베이스 엔진으로서, zero configuration 의 특성을 지니고 있고, stand alone 한 특성도 지니고 있다. 하지만 다른 데이터베이스 엔진과 유사하게 transaction 의 ACID 를 지원하며, index 구조와 table 구조, 그리고 multi-layer index 구조를 지원하는 등, 기본적으로 데이터베이스 엔진이 가져야할 특성을 지니고 있다.

What is Androbench

Androbench 는 Android 기기의 스토리지 성능을 측정하는 벤치마크이다. Androbench 에서 제공하는 SQLite 벤치마크는 앱에서 사용하는 것과 유사한 데이터베이스 테이블에 대한 생성(Create), 삽입(Insert), 업데이트(Update) 및 삭제(Delete) 쿼리의 성능을 테스트한다. 또한 해당 쿼리 수행에서 발생하는 random I/O 성능, sequential I/O 성능에 대한 throughput 을 테스트한다.

이번 랩에서는, SQLite 의 구조와 기능에 대해서 알아보고, Androbench 를 SQLite 에서 구동하는 방법을 알아볼 것이다. 또한, 다양한 configuration 을 통해서 나오는 결과를 확인 및 분석하는 과정을 거칠 것이다.

3. METHODS

우선적으로, SQLite 를 빌드하는 과정을 거친다. SQLite 가 성공적으로 빌드 되면, SQLite 의 configuration 파일을 변경하여 반복적으로 androbench 벤치마크를 수행한다. 수행 후 나온 결과를 다른 파일로 저장하여, 각각의 configuration 을 비교 및 분석한다. 변경되어지는 configuration 요소는 다음과 같다. Journal mode, Page size, Cache size, Synchronous mode, Locking mode.

4. Performance Evaluation

4.1 Experimental Setup

Type	Specification
OS	Ubuntu 20.04.3 LTS
CPU	AMD Ryzen 7 5800X 8-Core Processor (VMware support 4 Core)
Memory	4GB
Kernel	Linux ubuntu 5.11.0.34 -generic
Disk	VMware Virtual 80GB

Table 1: System setup

Type	Configuration (Default)
Journal mode	Delete
Page cache	2000
Page size	4096
Locking mode	NORMAL
Synchronous	1

Table 2: Configuration (Default)

4.2 Experimental Results

1. Configuration Factor

1.1 Journal mode

SQLite 에서 사용하는 journal 은 atomic commit & rollback 을 지원하기 위한 임시 파일이다. 즉, 저널 파일은 예기치 않은 에러에 의해 데이터 유실 혹인 파일 손상을 방지하기 위해, 문제 발생 시 원래 데이터로 복원하고 자 하는 것에 목적이 있다. SQLite 는 이러한 저널에 다양한 방식을 두고 있다. 우선적으로 default 로 사용하는 delete journal mode 는 하나의 트랜잭션이 시작되고 종료될 때마다, 저널 파일이 생성되고 삭제되는 과정을 수행한다. 따라서 매우 무거운 연산에 해당한다. 그리고, off 모드는 저널파일을 사용하지 않기 때문에, 데이터의 유실과 파일 손상에 매우 취약하다. 하지만 저널파일에 쓰이는 I/O 가 일어나지 않기 때문에, 성능적으로는 뛰어나다. 마지막으로 wal journal mode 는, 기본적인 저널 방법과는 반대의 방법을 사용한다. 기본적인 저널 방식은, 변경되지 않은 원본 데이터를 저널 파일에 쓰고, 변경하는 데이터는 즉시 데이터베이스에 기록하는 방식을 취한다. 하지만 wal journal mode 는 변경하는 데이터를 저널에 먼저 쓰고, 기존의 데이터는 데이터베이스 내부에서 복제하지 않는다. 또한, commit record 를 따로 wal 파일 내부에 두어, 특정 프로세스가 읽기 연산을 수행할 때, commit 이 완료된 데이터 부분만 읽을 수 있도록 처리한다. 그리고, check point 를 두어, wal 파일의 크기가 특정

임계점을 넘어갔을 때, 데이터베이스 파일에 쓰여지는 방식을 통하여 sequential 하게 쓰이며, random I/O 를 최소화하는 방식을 통해 I/O 에서 발생하는 오버헤드를 최소화하고, del mode 에 비해 파일에 쓰여지는 횟수가 적기 때문에, 성능적인 측면에서 del mode 보다 뛰어나다.

1.2 Page size

SQLite 에서 파일에 저장되는 최소한의 단위이다. Page size 는 configuration file 의 수정을 통해서 변경이 가능하다.

1.3 Cache size

SQLite 에서 사용되어지는 cache 의(Memory buffer) 크기이다. 데이터베이스에서 수행되는 쿼리에 의해서 발생하는 디스크 I/O 의 연산을 줄이기 위해, 일정 메모리 공간을 캐시로 사용한다. 따라서 이 용역의 크기에 따라서 성능이 좌우된다. 해당 요소도 위의 요소와 같이 configuration file 의 수정을 통해서 쉽게 변경이 가능하다.

1.4 Synchronous mode

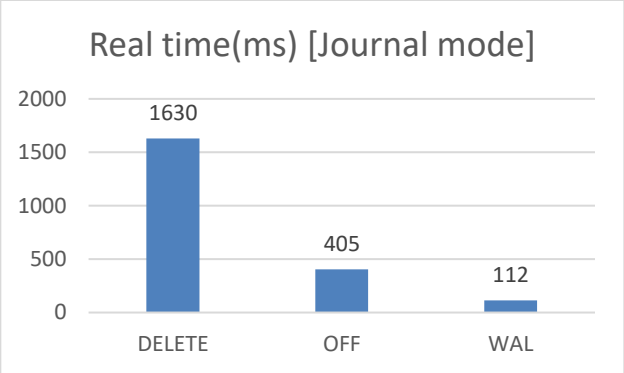
SQLite 에서는 데이터베이스에 데이터를 입력하거나 데이터베이스에 담긴 정보를 수정하게 되면, 먼저 내부 메모리에서 변경 작업을 한 후, 디스크에 쓰는 작업을 한다 (Dirty Cache). 이러한 과정을 수행하는 요소가 synchronous 요소이며, SQLite 에서는 총 3 개의 mode 를 지원한다. FULL(2), NORMAL(1), OFF(0). FULL(2) mode 는, 메모리상의 변경사항이 디스크에 완전히 기록된 것을 확인할 때까지 진행을 멈추고 대기한다. NORMAL(1) mode 는 디스크 동기화 작업을 수행하지만, FULL mode 보다 동기화 수준이 낮다. 그리고 마지막으로 OFF(0) mode 는 별도의 디스크 동기화 작업 없이 메모리에서 변경된 사항이 디스크에 기록되는 것을 확인하지 않고 계속 진행한다. 따라서 FULL(2) mode 가 성능 차원에서는 가장 낮은 성능을 보이고, 순차적으로 성능의 측면에서는 개선된다. 하지만 FULL(2) mode 에서 멀어질수록, 데이터 손실의 위험을 증가한다.

1.5 Locking mode

SQLite 는 두가지의 locking mode 를 지원한다. Default locking mode 는 Normal locking mode 이다. 이는 트랜잭션의 동작에 맞춰 데이터베이스 파일의 locking level 이 자동적으로 조절된다. 가령, 특정 process 가 읽기 연산을 수행한다면, shared lock 으로 조정되고, 특정 process 가 write 를 한다면 exclusive lock 으로 조정된다. 또한 해당 process 가 데이터 파일에서 수행을 종료한다면, 해당 파일은 lock 이 해제된다. 반면, Exclusive locking mode 에서는, 특정 process 에 의해 locking level 이 증가한다면, 해당 process 가 작업을 끝나고 난 후에도 locking level 을 유지한다. 따라서 lock 을 처리하는 system call 함수의 호출이 Normal locking mode 보다 적어서, 성능적인 측면에서는 Exclusive locking mode 가 우수하다.

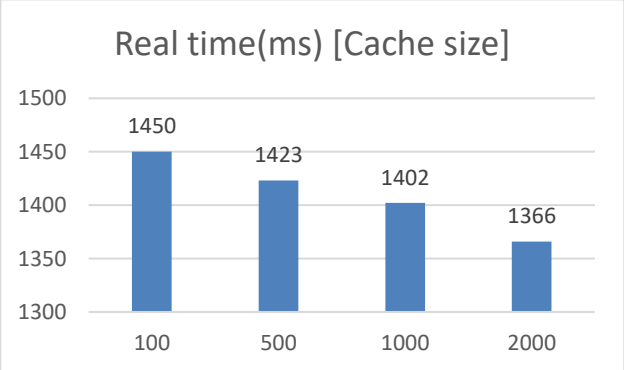
2. Analyze Result

2.1 Journal mode



Journal mode configuration 은 Delete mode, Off mode, Wal mode 순서대로 성능 측면에서 우수하다. 그 이유는 앞에서 언급한 바와 같이, delete mode 는 roll back transaction 의 방식을 취하기 때문에, 변경되는 데이터의 I/O, 기존 데이터의 I/O 총 2 번의 disk write 가 일어난다. 하지만, Off mode 의 경우에는 저널 파일을 생성하지 않기 때문에, 변경되는 데이터의 I/O 만 일어난다. 따라서 성능적인 측면에서 Delete mode 보다 우수하다. 그리고 마지막 Wal mode 는, 변경하는 데이터만 따로 Wal 파일에 저장을 하기 때문에, check point 에 도달하기 전까지는 한번의 disk I/O 만 발생하고, check point 에 도달했을 경우에도, sequential 하게 disk I/O 가 발생하기 때문에 random I/O 를 최소화한다. 따라서 성능적인 측면에서 앞 두개의 mode 에 비해 우수하다.

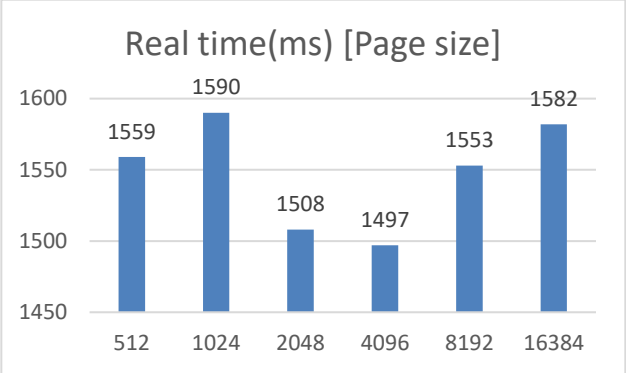
2.2 Cache size



위의 그래프의 양상은, cache size 가 증가할수록 성능적인 부분에서 우수함을 보인다. 이는 memory buffer 의 크기가 커질수록, memory buffer 에 올라가 있는 데이터가 많아져서

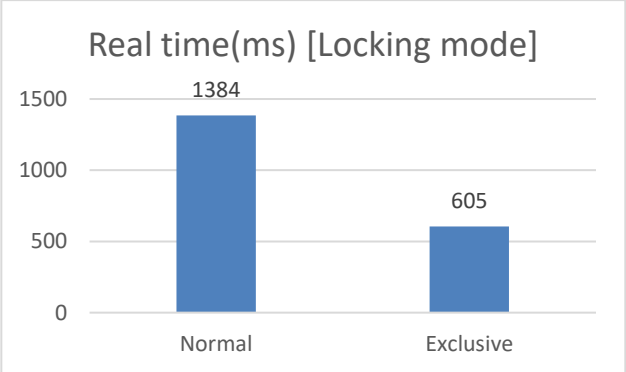
disk I/O 의 횟수가 줄어들게 된다. 따라서 memory buffer(cache)의 크기가 커질수록, 성능적인 측면에서 우수하다.

2.3 Page size



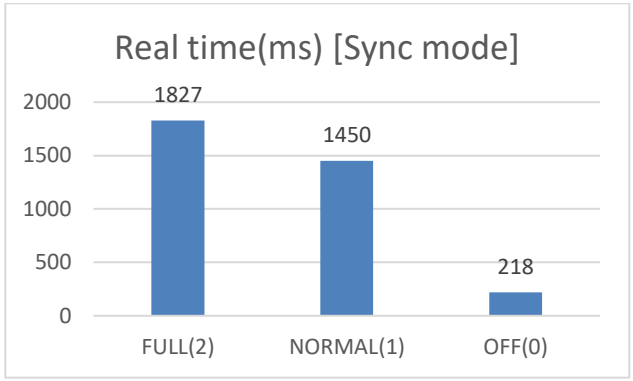
위의 그래프 양상은, page size 에 별로 상관되지 않은 성능 분포를 보인다. 따라서 page size 와 SQLite 의 성능은 무관해 보인다.

2.4 Locking mode



Locking mode configuration 은 크게 normal mode 와 exclusive 모드가 있고, exclusive mode 가 성능적인 측면에서는 우수하다. 그 이유는 앞서 언급한 바와 같이, exclusive mode 는 lock 을 제어하는 system call 함수의 호출을 최소화하기 때문이다. 따라서 system call 함수의 호출에서 발생하는 오버헤드를 최소화할 수 있어서 성능적인 측면에서 normal mode 보다 exclusive mode 가 우수하다.

2.5 Synchronous mode



Synchronous mode 는 크게 FULL(2), NORMAL(1), OFF(0)의 mode 로 구성되어 있고, 순서대로 성능적인 측면에서 개선된다. 그 이유는, 메모리상의 변경사항이 디스크에 flush 되는 과정에서 발생하는 연산의 비용이 크기 때문이다. 따라서 동기화 작업을 수행하지 않는 OFF(0) mode 가 성능적인 측면에서는 가장 월등하다.

3. Best configuration

Type	Configuration
Journal mode	Wal
Cache size	2000
Page size	4096 (Default)
Locking mode	Exclusive
Synchronous mode	1

Table 3: Best configuration

내가 생각하는 best configuration 은 위의 표와 같다. 우선 journal mode 에서는 wal mode 가 성능적으로 가장 우수하고, 트랜잭션 처리에 있어서도 다른 mode 와 성능적으로 차이가 없다. 다음으로, cache size 는 2000 을 적용했을 때, 성능적으로 가장 우수하다. 그리고 locking mode 는 system call 호출 횟수를 최소화하는 exclusive mode 가 성능적으로 가장 우수하다. 마지막으로, synchronous mode 는 OFF(0) mode 가 성능적으로 가장 우수하지만, 데이터의 손실 우려가 가장 크기 때문에 성능적인 측면과, 데이터 손실 정도를 모두 고려하는 NORMAL(1) mode 를 선택했다.

5. Conclusion

SQLite 에는 다양한 configuration factor 가 있다. 이번 랩에서는 총 5 개의 configuration factor 를 고려하여 실험을 진행했다(Journal mode, Cache size, Page size, Locking mode, Synchronous mode). 우선 journal mode 에 있어서는, default mode 인 delete mode 의 성능이 가장 낮았고, 그 이유는 하나의 트랜잭션이 처리될 때 마다 저널파일의 생성과 삭제를 반복하기 때문이다. 그리고 off mode 의 성능은 저널 파일을 따로 생성하지 않아, disk I/O 가 한번 만 일어나기 때문에 delete mode 보다 우수한 성능을 보였다. 하지만 데이터 손실과 파일 손상에 있어서 고려를 하지 않는 journal mode 이다. 마지막으로 wal mode 는 check point 에 도달하기 전까지는 disk I/O 가 wal 파일에서 일어나며, wal 파일에 append 하는 형태로 I/O 가 일어나기 때문에, 데이터베이스 파일에 병합될 때 sequential 하게 I/O 가 일어난다. 따라서 성능적인 측면과 데이터 보존의 측면에서 고려하면 wal mode 가 가장 좋은 configuration mode 로 고려된다. 다음으로는 cache size 이다. Cache size 의 크기가 증가할수록, cache 에 올라가 있는 데이터들이 많기 때문에 disk I/O 의 발생 빈도가 줄어든다. 따라서 일반적으로는 cache size 가 증가할수록 성능적인 측면에서 우수함을 보인다. 다음으로는 locking mode 이다. Locking mode 에는 두가지 mode 가 있었고, 자동적으로 locking level 을 조정하는 normal mode 이 비해, 특정 process 에 의해 설정된 locking level 을 유지하는 exclusive mode 가 system call 호출을 최소화해 성능적으로 우수하다. 마지막으로는 synchronous mode 이다. Synchronous mode 에는 FULL(2), NORMAL(1), OFF(0)이 있고, 순차적으로 메모리와 디스크를 동기화 시키는 작업이 줄어들기 때문에 성능적으로 우수하다. 하지만 OFF(0)의 경우 동기화 작업을 수행하지 않기 때문에, 데이터 손실의 우려가 있다.

6. REFERENCES

[1] https://github.com/meeeejin/SWE3033-F2021/tree/main/week-10

[2] https://www.sqlite.org/fasterthanfs.html

[3] https://news.ycombinator.com/item?id=27897427

[4] https://blog.devart.com/increasing-sqlite-performance.html

[5] http://delab.yonsei.ac.kr/assets/files/publication/legacy/2014-%EC%A0%95%EB%B3%B4%EC%B2%98%EB%A6%AC%ED%95%99%ED%9A%8C-%EC%B6%98%EA%B3%84-dglee.pdf