안드로이드에서 SQLite 의 질의처리 성능 분석

신민철*, 장용일**, 이준희*, 이준수*, 박상현 *
*연세대학교 컴퓨터과학과
**㈜ LG 전자
e-mail : smanioso@cs.yonsei.ac.kr

Analyzing Performance of Query Processing in SQLite on Android

Mincheol Shin*, Yongil Jang**, Joonhee Lee*, Sanghyun Park*
*Dept. of Computer Science, Yonsei University
**LG Electronics

요 약

스마트폰의 등장과 스마트폰의 기능을 풍부하게 만드는 앱이 제공됨에 따라 우리는 유용한 기능을 일상 생활에서 매우 쉽게 사용할 수 있게 되었다. 이러한 앱은 대부분 SQLite 라는 단순한 DBMS 를 통해 데이터를 저장하고 관리한다. 하지만 SQLite 는 임베디드 장치의 DBMS 라는 초기 목표를 위해, 그리고 특허 등의 라이선스 문제로 인해 다소 단순한 시스템으로 설계 되었다. 하지만 처음 목표 했던 임베디드 시스템에 비해 스마트폰의 높은 성능에서는 SQLite 의 단순한 구조가 오히려 성능저하를 일으킨다. 사용자경험의 증가를 위해 SQLite 의 성능을 향상시키는 연구가 필요하며, 이를 위해 SQLite 의 질의 성능을 분석하는 연구가 필요하다. 본 논문에서는 SELECT, DELETE, INSERT, UPDATE 등의 단순 질의의 성능을 실제 스마트폰에서 측정하고, 이를 CPU 연산과 I/O 연산으로 나누어 분석한다. 이러한 분석결과 SQLite 의 SELECT 질의는 CPU 연산에 큰 영향을 받으며 읽기연산은 리눅스 커널에서 제공하는 기능으로 인해 높은 성능을보인다. 다른 질의의 경우 쓰기연산이 포함되어 있으며 전체 질의처리시간에서 80% 에서 95% 정도가 쓰기연산 시간이다. 따라서 효율적인 CPU 연산을 통해 CPU 연산시간을 줄이고 리커버리 시스템과 같은 쓰기연산에 영향을 미치는 모듈에 대한 연구가 필요하다.

1. 서론

스마트폰의 등장은 사람들의 삶을 많이 바꾸어 놓았다. 우리는 스마트폰의 무료 메신저로 다른 사람과 소통할 수 있고 방금 찍은 사진을 웹에 곧바로 올릴 수도 있으며, 스마트폰의 지도를 통해 현재의 위치와 목적지까지의 길을 안내 받을 수도 있다. 이렇게 다양한 기능은 사용자가 직접 개발자가 되는, 열린 생태계를 통해 다양한 앱이 제공 되기 때문이다.

이러한 유용한 기능을 제공해 주는 앱은 iOS 와 안드로이드를 막론하고 SQLite 를 데이터베이스로서 제공한다. SQLite 는 간편하고 서버가 따로 필요 없는 가벼운 데이터베이스 관리 시스템으로써 간단한 데이터베이스 관리 시스템이 필요한 분야에서 광범위하게 사용된다.

일상적으로 쓰이는 많은 앱들이 SQLite 를 사용하고 있지만 SQLite 는 초기 낮은 사양의 임베디드 장치에서도 돌아갈 수 있도록 구조가 간단하게 설계되어 최근 스마트폰의 향상된 기능을 충분히 활용하지 못하여 이에 대한 개선이 필요하다. 가령, SQLite 는 기존 RDBMS 에서 사용하는 델타로그 방식의 리커버리 시스템이 아닌, 수정된 페이지 전체를 저널링 하는 단순한 리커버리 시스템을 가지고 있다.

이러한 이유로 모바일이라는 특수한 환경에 맞추어 성능을 개선을 할 필요가 있으며, 이에 앞서 SQLite 가 실행하는 질의의 수행시간의 특성을 파악하는 연구가 필요하다.

실제로 학계에서는 저장장치로 인하여 안드로이드의 성능이 저하된다고 밝혀졌으며 [1], SQLite 의 비효율적인 구조에 대한 성능 측정 및 이를 최적화하는 방법에 대한 연구[2]가 진행된바 있다.

하지만 이러한 연구는 SQLite 의 내부구조의 병목현상에 대한 연구가 아니며, 따라서 본 논문에서는 SELECT, INSERT, UPDATE, DELETE 등의 단순질의의 수행시간을 CPU 연산, 읽기 연산, 쓰기 연산 등으로 구분하여 분석을 하였다.

본 논문은 다음과 같이 구성되어 있다. 2 장에서는 SQLite 의 특징에 대해 간략하게 살펴본다. SELECT, INSERT, UPDATE, DELETE 질의의 수행시간을 3 장에서 분석한다. 마지막으로 4 장에서는 이러한 분석결과를 종합하여 현재 SQLite 의 문제점을 진단한다.

2. SQLite 의 특징 [3]

이 연구는 ㈜ LG 전자의 지원을 받아 수행된 것임

SQLite 는 설치가 간편하고 서버가 따로 필요 없는 가벼운 데이터베이스 관리 시스템으로 간편한 데이터베이스가 필요한 분야에서 많이 사용된다. 대표적으로 모바일 운영체제인 iOS 나 안드로이드에서 애플리케이션을 위한 데이터베이스로 제공된다. 이외에도 웹브라우저, 안티바이러스 프로그램, 그래픽 툴 등 많은 분야의 어플리케이션이 사용한다 [4]. SQLite 는 간단한 질의 처리에 사용되기 때문에 단순 질의를 처리하는데 최적화 되어 설계되어 있다. 기존의 RDBMS 가 질의를 트리 형태로 파싱하여 질의를 수행하는 반면, SQLite 는 절차적 형식을 가지고 있는 프로그램을 생성하여 이를 VDBE (Virtual Database Engine) 모듈을 통해 실행한다.

SQLite 는 테이블을 B-tree 로 관리하는 테이블 B-tree 라는 독특한 특징을 가지고 있다. 모든 테이블은 rowid 라는 자동으로 생성되는 속성(attribute)를 가지고 있으며 이 rowid 속성은 테이블 B-tree 의 Key 가 된다. 테이블 B-tree 이외에도 보조 인덱스를 생성할 수 있으며 보조 인덱스는 인덱스 B-tree 라고 하는 변형된 B-tree 를 사용한다.

SQLite 는 일반적으로 사용되는 RDBMS 보다 훨씬 간단한 형태의 리커버리 시스템을 사용한다. 기존 RDBM 는 페이지내의 변경된 부분만 로깅하는 델타 로그 방식을 채용하고 있으나 SQLite 는 모든 변경사항을 페이지 단위로 관리하는 방식을 사용하고 있다. SQLite 는 크게 Rollback Journal (이하 RBJ)과 Write Ahead Logging (이하 WAL)이라고 불리는 리커버리 시스템을 사용한다. 간단하게 RNJ 는 수정된 페이지의 원본을 백업하고 이를 질의가 끝날 때까지 유지하고 WAL 은 변경된 페이지를 WAL 파일에 별도 저장하고 후에 이를 한번에 데이터베이스 파일에 반영한다.

```
CREATE TABLE chat_logs (
  '_id'         INTEGER PRIMARY KEY AUTOINCREMENT,
  'id'           INTEGER,
  'type'         INTEGER,
  'chat_id'      INTEGER,
  'user_id'      INTEGER,
  'message'      TEXT,
  'attachment'   TEXT,
  'created_at'   INTEGER,
  'is_temp'              INTEGER,
  'v' TEXT
, deleted_at INTEGER NOT NULL DEFAULT 0, client_message_id INTEGER);
CREATE UNIQUE INDEX chat_logs_index1 ON chat_logs(chat_id, id);
CREATE INDEX chat_logs_index2 on chat_logs(chat_id);
CREATE INDEX chat_logs_index3 on chat_logs(created_at);
CREATE INDEX chat_logs_index4 on chat_logs(chat_id, type);
CREATE INDEX chat_logs_index5 on chat_logs(chat_id, deleted_at);
```

(그림 1) chat_logs 테이블 스키마

3. 질의 성능 분석

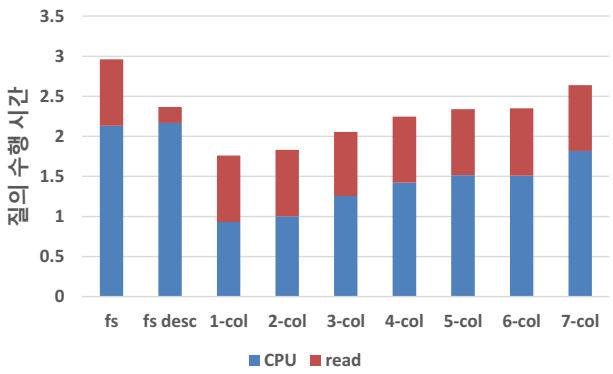
Google Nexus 5 에서 SELECT, UPDATE, DELETE, INSERT 질의를 수행하여 질의 성능 분석을 하였다. 현재 안드로이드에 많이 탑재되고 있는 SQLite 3.7.11 버전을 사용하였다. 질의의 전체 수행시간을 측정하였고, SQLite 에서 호출하는 I/O 관련 시스템 콜이 호출되는 메소드의 수행시간을 누적하여 측정하는 방법을 사용하였다. SQLite 는 buffered I/O 를 사용하기 때문에 읽기 및 쓰기 시스템콜 뿐만 아니라 OS 페이지 캐시를 저장장치와 동기화하는 시스템콜인 fsync 의 수행 시간 또한 측정하였다.

데이터베이스는 카카오톡 데이터베이스를 사용하였으며, 카카오톡 메시지를 저장하는 chat_logs 테이블을 이용하였다. chat_logs 테이블의 스키마는 그림 1 과 같으며, 총 5 개의 인덱스가 생성되어 있다.

3.1. SELECT 질의

SELECT 질의는 크게 테이블 전체 스캔 질의의 성능을 분석하였다. 테이블 전체 스캔의 경우 rowid 를 기준으로 하여 오름차순으로 정렬하는 질의와 내림차순으로 하는 질의를 수행 하였다. 또한 내림차순 질의를 출력하는 속성의 개수를 조절하며 실험하였다. 그림 2 에서 각 실험은 20 회 반복 실험 한 뒤 최대값과 최소값을 제외한 평균으로 계산 하였다. 또한 수행된 모든 질의가 실행되기 전에 OS 페이지 캐시를 제거한 뒤 실험을 진행하였다. 실험에 사용한 질의는 다음과 같다.

```
SELECT * FROM chat_logs;
SELECT * FROM chat_logs ORDER BY rowid DESC;
SELECT {attrs} FROM chat_logs ORDER BY rowid DESC;
```

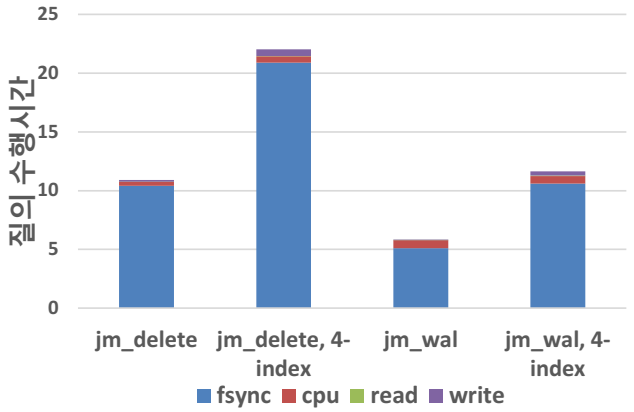


(그림 2) 오름차순 및 내림차순 전체 스캔의 실험결과 및 출력하는 속성의 개수를 조절하며 내림차순 전체 스캔을 수행한 결과

그림 2 는 전체 오름차순 스캔과 내림차순 스캔의 실험 결과를 보여준다. 내림차순 스캔에 비해 오름차순 스캔의 속도가 빠르며 읽기 시간이 더 큰 것을 볼 수 있는데, 이는 OS Prefetch 의 영향이다. 리눅스 커널에서 지원하는 prefetch 는 연속된 페이지를 디스크 주소가 커지는 방향으로 연속으로 읽는 방식으로 동작하는데 오름차순 스캔의 경우 rowid 로 정렬된 테이블 B-tree 를 탐색하므로 prefetch 의 이득을 가져 갈 수 있다.

출력하는 속성의 개수를 조절한 실험결과는 그림 2 에서 볼 수 있다. 그림 2 에서 Y 축은 질의의 수행 시간이며 X 축은 출력하는 속성의 개수를 나타낸다. 가져오는 속성이 많을수록 CPU 연산 시간이 커지며 이로 인해 질의의 수행시간이 커지는 것을 관찰 할 수 있다. 가져오는 속성의 개수가 하나 증가하면 SQLite 의 VDBE 프로그램에서 해당 속성을 가져오는 연산이 하나 추가된다. 즉 많은 속성을 가져오면 하나의

레코드에서 수행하는 VDBE 연산이 많아지고 이로 인해 질의 처리의 전체 수행 시간이 증가하게 된다. 실험 결과를 보면 플래시 메모리에서 데이터 페이지를 읽는 연산이 비중을 차지하지 않으므로 CPU 연산 시간의 증가는 질의 수행시간에 큰 영향을 미친다.



(그림 3) 1000 개의 UPDATE 질의 수행 시간 및 CPU 연산시간과 I/O 연산시간.

3.2. UPDATE 질의

UPDATE 질의는 chat_logs 테이블에 임의의 레코드 1000 개를 rowid 로 검색하여 하나의 속성을 수정하는 질의를 수행하였다. 검색에 사용한 rowid 는 rowid 의 최소 값과 최대값 사이에서 생성하였다. 인덱스가 구축된 속성과 인덱스가 구축되지 않은 속성의 성능 모두를 분석하기 위해서 인덱스 4 개에 영향을 주는 *chat_id* 와 보조인덱스에 영향을 미치지 않는 *is_temp* 속성을 수정하는 질의로 나누어서 실험을 진행 했다. 또한 리커버리 모드에 따라 성능이 변화할 수 있으므로 RBJ 의 DELETE 저널 모드와 WAL 저널 모드 둘로 나누어서 실험을 진행하였다. 실험에 사용한 질의는 다음과 같다.

UPDATE chat_logs **SET** {*attribute*} = {*attribute*} + 10 **WHERE** rowid = {*random value*};

그림 3 은 위에서 기술한 실험의 결과를 보여준다. jm_delete 는 저널모드가 DELETE 로 설정된 것을 의미하며 4-index 는 인덱스 4 개에 영향을 미치는 chat_log 의 실험 결과를 나타낸다. 그래프의 푸른색 영역은 fsync 의 수행시간이고, 붉은색은 cpu 수행 시간, 녹색은 read 의 수행시간, 보라색은 write 의 수행시간을 의미한다.

그래프에서 볼 수 있듯이 UPDATE 질의 수행시간 중 대부분이 수정된 버퍼를 디스크에 쓰기 위해 호출되는 fsync 로 인해서 발생된다. SQLite 는 Buffered I/O 를 사용하기 때문에 read 나 write 시스템콜 같이 디스크에 입출력 하는 경우 디스크에 직접 쓰지 않고 OS 의 페이지 캐시에 읽고 쓰게 된다. 수정된 버퍼의 경우 victim 으로 선택되거나 fsync 가 호출될 때 디스크에 쓰여지게 되며 수정되지 않은 버퍼는 디스크에 쓰이지 않는다. 때문에 fsync 는 쓰기연산으로 간주 할 수 있다.

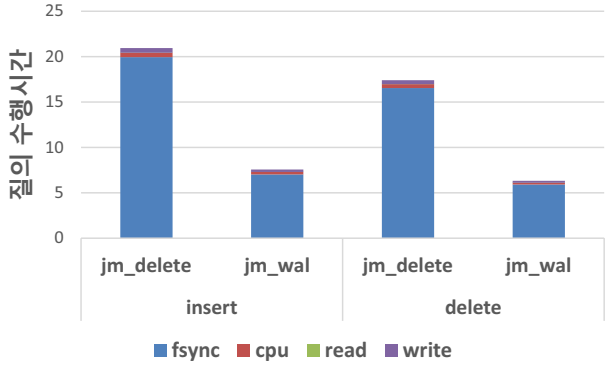
저널모드가 DELETE 인 경우와 WAL 인 경우 약 2 배정도의 성능차이가 나타나는데 이는 RBJ 와 WAL 저널 모드의 차이에서 기인한다. RBJ 방식은 데이터베이스가 수정 될 때마다 원본 페이지를 저널 파일에 복사하고 해당 질의가 종료 되면 저널파일을 삭제한다. 반면 WAL 의 경우 수정된 페이지를 WAL 파일에 저장하며 저널 파일의 크기가 일정 크기 이상이 되면 WAL 파일의 내용을 데이터베이스 파일에 반영한다.

수정되는 속성에 인덱스가 있는 경우 인덱스가 없는 경우에 비해 느린 것을 발견 할 수 있다. 이는 수정 된 속성에 인덱스가 있을 경우 인덱스 또한 수정해야 되기 때문이다.

3.3. INSERT & DELETE 질의

INSERT 및 DELETE 질의는 chat_logs 테이블에 임의의 레코드 1000 개를 삽입하거나 삭제하고 그 때 걸리는 시간을 측정하였다. 이 두가지 질의는 UPDATE 질의와 마찬가지로 데이터베이스의 수정을 발생시키므로 리커버리 시스템에 따른 성능 변화가 발생한다. 때문에 RBJ 의 DELETE 저널 모드와 WAL 저널 모드 둘로 나누어서 실험을 진행하였다. 실험에 사용한 질의는 다음과 같다.

INSERT INTO chat_logs **VALUES**({*values*});
DELETE FROM chat_logs **WHERE** rowid={*rowid*};



(그림 4) 1000 개의 INSERT 질의 수행시간 및 1000 개의 DELETE 질의 수행시간

그림 4 는 INSERT 와 DELETE 질의의 수행 시간 및 각 연산의 수행시간을 나타낸다. jm_delete 는 저널모드가 DELTE 일 때의 실험을 나타내고 jm_wal 은 저널모드가 WAL 일 때의 실험 결과를 나타낸다.

UPDATE 와 마찬가지로 fsync 의 소요시간의 비중이 상당히 크게 나타나고, 또한 WAL 저널모드가 DELETE 저널 모드에 비해 빠른 것을 볼 수 있다.

4. 결론

SQLite 는 리눅스 커널에서 제공하는 기능들을 활용하여 플래시 메모리에서 데이터를 읽는 연산의 부하를 상당부분 줄인다. 리눅스 커널에서 제공하는 페이지 캐시를 통해 플래시 메모리를 읽어야 하는 경우를 상당부분 피한다. 또한 OS prefetch 기능을 통해 순차 읽기 성능을 향상시킨다.

위와 같은 이유로 SELECT 질의에서 읽기연산의 비중은 그리 크지 않으며, CPU 연산 시간의 비중이 상당히 높다. SELECT 질의에서 CPU 연산시간은 출력하는 속성의 개수에 큰 영향을 받는 특징을 가지고 있다. 따라서 불필요한 속성을 가져오는 SELECT 질의는 응용 프로그램의 성능 저하를 불러 올 수 있다.

SQLite 는 리커버리 시스템의 특성상 리눅스 페이지 캐시의 이득을 보기가 어려운 구조로 되어있어 UPDATE, DELETE, INSERT 와 같은 쓰기연산이 포함된 질의의 성능에서는 쓰기연산 시간의 비중이 크게 나타난다. RBJ 의 경우 데이터베이스의 수정이 발생하면 무조건 두 개의 페이지를 쓰게 되며 이는 OS 페이지 캐시에 쓰여지게 되지만, 질의가 끝날 때 fsync 를 이용하여 플래시 메모리에 이를 동기화 시켜야만 한다. WAL 은 데이터베이스의 수정이 발생하면 한번의 페이지만 쓰게 되지만 이 또한 역시 fsync 를 통해 플래시 메모리에 써야만 하고, WAL 파일의 내용을 데이터베이스에 적용해야 한다.

결론적으로, SQLite 의 SELECT 질의의 성능을 향상시키기 위해서는 SQLite 의 CPU 연산 시간을 줄이는 연구가 필요하며, 그 외 쓰기연산이 포함된 질의의 경우 쓰기연산의 시간을 줄이는 연구가 필요하다.

참고문헌

[1] H. Kim, N. Agrawal, C. Ungureanu, “Revisiting Storage for Smartphones”, In ACM Transactions on Storage, Vol.8, No. 4, Article 14, November 2012.

[2] S. Jeong, K. Lee, S. Son, and Y. Won, “I/O Stack Optimization for Smartphones”, In the USENIX Annual Technology Conference, 2013, San Jose, USA.

[3] SQLite, “SQLite Documents”, URL:http://sqlite.org/docs.html

[4] SQLite, “Well-Known Users”, URL:http://sqlite.org/famous.html