SQLite를 통한 플래시 메모리 파일시스템 분석

박혜련*, 오기환, 이상원
*성균관대학교 컴퓨터공학과
elana.park@skku.edu

An Analysis on Flash Memory File System Using SQLite

Hyeryeon Park*, Gihwan Oh, Sang-Won Lee
*Dept of Computer Engineering, Sungkyunkwan University

요 약

전 세계 주요 56개국 성인 인구의 스마트폰 보급률이 평균 약 60%에 달하고, 지난 3월 기준 한국 스마트폰의 보급률은 83.0%에 달해 세계 4위를 기록하였다. 안드로이드, iOS를 포함한 대부분의 모바일 플랫폼은 SQLite 데이터베이스를 기본 데이터베이스로 사용하고 있는 것으로 알려져 있다. 현재 보급된 대부분의 스마트폰의 저장장치는 플래시 메모리를 기반으로 하고 있다. 그러나 현재까지 안드로이드 운영체제의 기본 파일시스템은 Ext4 파일시스템으로 알려져 있으며, 플래시메모리에 최적화되었다고 주장하는 다른 파일시스템에 대한 성능 평가 및 데이터 입출력 특징의 자세한 분석 연구는 존재하지 않았다. 본 논문에서는 가장 잘 알려진 파일 시스템 Ext4, XFS, Btrfs 세 종류에서 실제 안드로이드 애플리케이션 쿼리를 사용하여 성능 측정을 진행하였다. 실험 결과 기본 파일 시스템으로 사용되고 있는 Ext4가 가장 빠른 성능을 나타낸 것을 확인하였고, 각 파일시스템마다 완연히 다른 데이터 입출력 특징을 갖고 있는 것을 확인하였다.

1. 서론

플래시 메모리 기반의 Solid State Drive(SSD)가 시장에 출시되고 점점 주요 저장장치로 자리를 잡고 있으며, 모바일 플랫폼 상에서의 플래시 메모리 SSD 또한 필수 불가결한 존재가 되었다. 전 세계 주요 56개국 성인 인구의 스마트폰 보급률은 평균 약 60%에 육박하고, 지난 3월 기준으로 한국 스마트폰의 보급률은 83%에 달해 세계 4위를 기록하였다 [5]. 가파른 스마트폰의 보급률만큼이나 모바일 플랫폼에 사용되는 플래시 메모리 SSD의 성능 또한 가파르게 증가하고 있다. 하지만 지금까지 알려진 바로는 안드로이드는 기본 운영체제로 Ext4를 사용하고 있으며, SSD의 성장에 따라 발표된 SSD에 최적화된 수많은 파일시스템에 대한 성능평가 및 적용 가능성에 대한 연구는 이루어지지 않고 있다.

성장하는 모바일 플랫폼 시장에서 플래시 메모리 SSD 만큼 중요하게 꼽히는 요소 중 하나가 바로 모바일 플랫폼 상에서의 기본 데이터 관리 소프트웨어이다. 안드로이드뿐만 아니라 iOS를 포함한 대부분의 모바일 플랫폼은 현재까지 임베디드 환경을 위해 개발된 SQLite 데이터베이스를 기본 데이터베이스로 사용하고 있는 것으로 알려져 있다 [1]. SQLite는 그 구현의 이념에 따라 쉽고 빠른 데이터 관리를 라이브러리 형태로 제공하여, 다양한 애플리케이션에서 손쉽게 SQLite를 통한 데이터베이스 관리를 할 수 있는 환경을 제공하고 있다. 또한, 저장되는 데이터의 원자성 및 영속성을 보장하기 위해 저널링 모드를 제공하고 있다.

SQLite가 제공하는 저널링 모드는 크게 Rollback 저널링 모드와 WAL 저널링 모드로 분류된다. 그러나 구현상의 안정성, 역사성 및 다중 파일에 대한 원자성 지원 여부에 따른 차이점으로 인해 현재 모바일 플랫폼에서 사용되는 SQLite는 대부분 Rollback 저널링 모드를 기본 저널링 모드로 사용하고 있다. SQLite는 데이터의 영속성을 제공하기 위해 저널링 모드를 제공할 뿐만 아니라, 트랜잭션 종료 시 버퍼공간에 남아있는 변경된 데이터에 대해 force-write 정책을 취함으로써 데이터 페이지가 온전히 쓰여지도록 보장하고 있다 [2].

SQLite가 제공하는 데이터 관리의 편의성, 데이터 원자성 및 데이터 영속성에도 불구하고, 저널링 모드로 인한 불필요한 데이터 페이지의 다중 쓰기와 force-write 정책으로 인한 데이터 페이지 쓰기는 모바일 플랫폼상에서 기기의 성능 저하를 일으키는 원인으로 알려져 있다[3]. 이러한 문제점을 해결하기 위해 SQLite의 저널링 모드를 변경하거나 새로운 형태의 데이터베이스를 제안하는 연구들이 진행되고 있다 [4, 6, 7]. 그러나 기본적으로 플래시 메모리로 쓰여지는 모든 데이터 페이지들은 운영체제 상의 파일시스템을 통해 블록 단위 입출력 구조로 전달되는 점을 고려하면, SQLite 자체의 변화가 아닌 파일시스템의 변화 또한 필수 불가결한 상황임을 알 수 있다.

이러한 배경지식을 바탕으로 본 논문에서는 실제 안드로이드 기기에서 추출한 유명 애플리케이션들의 SQLite를 통한 데이터 관리 SQL 트레이스를 통해, Ext4, XFS, BTRFS 세 가지의 파일시스템에 대해 SQLite의 성능평가를 진행하였다 [8, 9, 10]. 또한 플래시 메모리에 최적화했다고 주장하는 각 파일시스템들의 실제 데이터 입출력 패턴을 파악함으로써 향후 파일시스템 연구에 대한 방향성을 제시하고자 한다.

2. 플래시 메모리 기반 파일시스템

2.1 Ext4 파일 시스템 [8]

64비트 기억 공간 제한을 없애고 Ext3의 성능을 향상시키며, 하위 Ext2,3와 하위 호환성이 있는 확장 버전으로서, 많은 부분이 본래, 병렬 분산 파일시스템을 위해 클러스터 파일 시스템스사에서 개발되었다. 그러나 다른 커널 개발자들은 안정성을 이유로 이를 반대했으며, 모든 개발에서 Ext3 사용자에게는 영향을 주지 않으면서 Ext3에서 분화하여 Ext4로 이름을 변경하기를 제안했다. 이 제안이 받아들여져 Ext4가 개발되었다. 기본적으로 하드 디스크 드라이브를 기반으로 개발되었으나, 많은 SSD 시스템에서도 기본 파일시스템으로 사용되고 있다.

리눅스 커널 버전 2.6.19부터 초기 버전이 포함되기 시작하였으며 2.6.28코드부터 Ext4 채택 권장이 있어 왔다. 최대 1엑사바이트의 볼륨과 최대 16테라바이트의 파일을 지원한다. 파일에 대하여 extent 단위로 공간이 할당 되며, extent는 인접한 물리적 블록의 묶음으로, 대용량 파일의 접근 성능을 향상시키고 단편화를 줄이기 위해 고안되었다.

지연 할당 기능이라고 알려진 allocate-on-flush 기술을 사용하여 파일 시스템 자체의 성능을 향상시켰다. 이 기술은 데이터가 실제로 디스크에 쓰이기 전까지 해당 파일에 공간 할당을 지연시켜, 실제 데이터가 디스크에 쓰이는 시점에 필요한 공간만큼을 extent 단위로 할당한다. 이를 통해 하나의 파일에 대한 블록이 여러 곳으로 분산되는 현상을 막고 디스크의 이동을 최소화시킴으로써 성능을 향상 시킨다.

2.2 XFS 파일 시스템 [9]

XFS는 1993년 실리콘 그래픽스(SGI)가 만든 고성능 64비트 저널링 파일 시스템으로 2001년부터 리눅스의 커널에 포함되었다. 2014년 6월 기준으로 XFS는 대부분의 리눅스 배포판에서 지원되고 있다.

64비트 파일의 모든 가능성을 사용하여 2GB이상의 파일에 대한 관리를 허용하며, 다른 저널링 파일시스템과 같이 저널링 기술을 이용하여 파일 시스템의 안정성을 보장한다. Ext4 파일 시스템과는 다르게 최소 512bytes 단위의 페이지 사이즈를 기준으로 파일시스템에서 지원 가능한 최대 페이지 사이즈까지 다양하게 선택할 수 있다. XFS에 구현된 저널링 기술은 데이터베이스 분야에서 널리 사용되는 저널링 기술을 채택하였다. 다중 데이터 입출력 요청이 들어올 경우 병렬 입출력 처리를 통해 입출력 성능을 극대화한다. 이러한 입출력 방식을 바탕으로 다중의 데이터 입출력 쓰레드에 대해 파일 시스템의 최대 대역폭 및 하드디스크, SSD와 같은 저장장치의 최대 대역폭을 활용할 수 있도록 한다.

파일시스템의 성능을 극대화하기 위하여 다른 파일시스템과 달리 저널링 영역을 물리적으로 분리된 디스크에 저장할 수 있는 옵션을 제공하여, 실제 데이터의 입출력을 담당하는 모듈과 저널링 입출력을 담당하는 모듈을 분리하여 동작하는 것이 가능하다. 이러한 기술은 logdev 옵션이라고 불리우며, SSD와 같이 데이터 입출력에 대해 매우 짧은 반응 시간을 제공하고 큰 대역폭을 가진 저장장치에 대해서 최대 성능을 이끌어 낼 수 있다고 알려져 있다.

2.3 BtrFS 파일 시스템 [10]

BtrFS(the copy-on-write B-tree file system)는 파일 시스템 2007년 USENIX에서 IBM의 O Rodeh에 의해 제안되었다[11]. 초창기 버전은 2008에 발표되었으며, 2009년도에 리눅스 커널에 포함되었다.

B+tree 데이터베이스 분야에서 on-disk 데이터를 관리하기 위해 유용히 사용된다는 관찰을 바탕으로, B+tree에서 대처하지 못하는 copy-on-write 문제를 해결하는 변경된 B-tree구조를 기반으로 개발되었다. 기본적으로 copy-on-write append only B-tree로 구성되어 있으며, 필요한 영역에 따라 같은 형태의 여러 B-tree로 구성되어 있다. 모든 파일에 대한 파일 정보 및 파일 데이터는 이러한 B-tree구조에 의해 관리 되며, Ext4와 같이 inode의 동적 할당 및 extent 단위의 공간 할당을 지원한다. 또한 SSD 최적화 모드를 지원하여 SSD의 수명 문제를 해결하기 위한 wear levelling을 고려하여 동작한다. 현재까지의 개발 사항에서는 SSD의 wear levelling에 대한 최적화만 진행되어 있으며, 실제 성능에 대한 최적화는 반영되지 않고 있다.

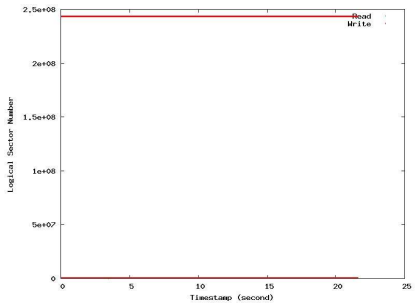
3. 실험 및 결과 분석

<표 1> 실험 환경

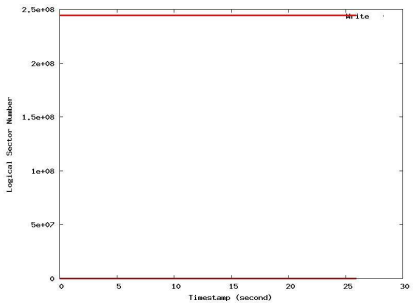
운영체제	Ubuntu 14.04, Linux Kernel 3.13
CPU	Core i3-3220 @ 3.30 GHz(4 core)
Memory	8GB DDR3
SQLite	3.8.11.1 Version
Applications	Androbench, Gmail, KakaoTalk
File Systems	Ext4, XFS, BtrFS

3.1 실험 환경

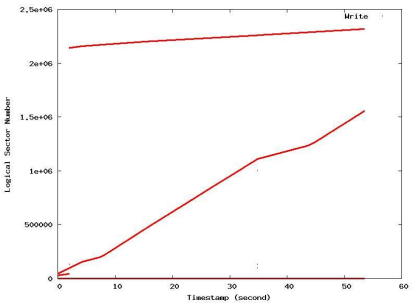
플래시 메모리 기반 파일시스템들에 대한 SQLite 성능 평가를 위하여 <표 1>의 환경을 사용하였다. 운영체제는 Linux Kernel 3.13 기반의 Ubuntu 14.04를 사용하였으며,



(그림 1-a) Ext4 입출력 패턴



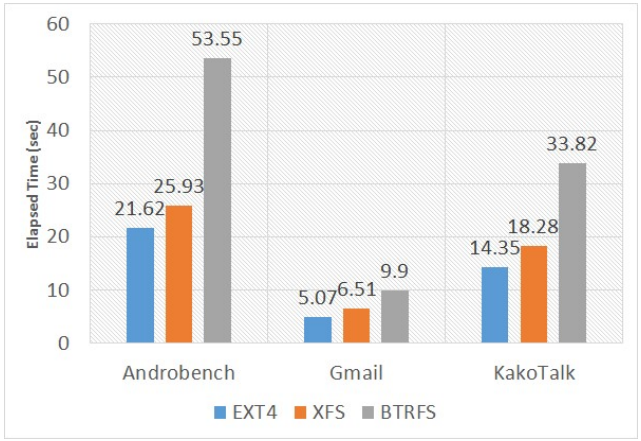
(그림 1-b) XFS 입출력 패턴



(그림 1-c) BtrFS 입출력 패턴

CPU 프로세스는 Core i3-3220 @ 3.30 GHz를 사용하였다. 메모리는 SQLite가 충분히 사용할 수 있도록 8GB DDR3 메모리를 장착하였으며, SQLite의 버전은 실험 당시 가장 최신 버전인 3.8.11.1을 사용하였다.

Ext4, XFS, BtrFS 파일시스템을 모두 사용하기 위해 각각의 파일시스템에 대한 커널 옵션을 사용으로 변경하였다. 각 파일시스템의 형태로 디스크를 포맷하기 위하여 각 파일시스템 개발사에서 제공하는 Linux 기본 패키지를 사용하였다. 모든 애플리케이션은 안드로이드 기본 저널 옵션인 Rollback 저널 모드에서 동작하도록 하였다.



(그림 2) SQLite를 통한 파일시스템 성능 측정

실험 결과를 위해 각 파일시스템 상에서 각 애플리케이션 별 SQLite를 통한 트레이스 수행 시간을 초 단위로 측정하였다. 각 파일시스템의 데이터 입출력 분석을 위해서는 Linux에서 기본적으로 제공하는 blktrace 패키지를 사용하였다.

3.2 성능 측정 결과

실험 결과는 (그림 2) SQLite를 통한 파일시스템 성능 측정에서 확인할 수 있다. 모든 애플리케이션에 대하여 Ext4 파일시스템이 가장 나은 성능을 보여주는 것을 확인할 수 있다. 첫 번째 워크로드인 Androbench 애플리케이션 트레이스에 대해서 Ext4는 XFS 대비 1.2배, BtrFS 대비 2.5배의 빠른 성능을 보임을 알 수 있다. 두 번째 워크로드인 Gmail 애플리케이션 트레이스에 대해서 Ext4는 XFS 대비 1.3배, BtrFs 대비 1.9배의 빠른 성능을 보임을 알 수 있다. 마지막 워크로드인 KakoTalk 트레이스에 대해 Ext4는 XFS 대비 1.3배, BtrFS 대비 2.4배의 빠른 성능을 보임을 알 수 있다. BtrFS의 경우 작은 단위의 업데이트가 잦은 Androbench 혹은 KakaoTalk 트레이스 실험에서 성능 악화가 더 큰 것을 확인할 수 있다. 이를 통해 SSD에 특화된 파일시스템이라고 발표된 XFS, BtrFS는 아직 최적화가 되지 않았으며, 안드로이드의 기본 파일시스템으로 사용되는 Ext4가 가장 빠른 성능을 나타내고, 안드로이드 시스템의 Ext4는 올바른 채택임을 확인할 수 있다.

3.3 파일시스템 데이터 입출력 특징 분석

각 파일시스템에 대하여 첫 번째 워크로드인 Androbench 애플리케이션에 대한 트레이스를 SQLite를 통해 수행하면서, Linux의 blktrace 도구를 통하여 데이터 입출력 로그를 추출하였다. 각 파일시스템의 입출력 특징은 위 (그림 1)에서 확인할 수 있다.

Ext4와 XFS는 비슷한 방식의 in-place update를 지원하는 파일시스템으로 구현되어 있어, 파일에 대한 데이터 입출력에 대해 비슷한 접근 패턴을 보이는 것을 알 수 있다. 두 파일시스템 모두 저널링 방식을 제공하기 때문에 SQLite에서 직접적으로 요청하는 데이터 입출력 외에 추가적인 데이터 입출력 패턴이 발생함을 확인할 수 있다.

BtrFS의 경우는 위의 두 파일시스템과 확연히 다르게, 그 기본 기술 설계인 copy-on-write append only B-tree에서 예상되는 데이터 입출력 패턴을 보임을 알 수 있다. Androbench 애플리케이션에 대한 워크로드 수행중 발생하는 모든 데이터 쓰기 요청이 순차 쓰기와 임의 쓰기가 섞여 있음에도 불구하고, 계속해서 파일시스템 상의 새로운 공간을 할당하여 해당 데이터를 씀으로써 실제 패턴 그래프에서 순차 쓰기 형태의 패턴을 보임을 알 수 있다. SQLite의 데이터베이스 파일 및 그 Rollback 저널 파일에 대해 2줄의 순차쓰기 형태를 확인할 수 있으며, BtrFS가 자체적으로 관리하는 metadata에 대해 1줄의 in-place update 패턴을 확인할 수 있다.

이를 통해 Ext4와 XFS는 비슷한 방식으로 구현된 파일시스템이면서 실제로 동일한 데이터 입출력 패턴을 가짐에도 불구하고 Ext4의 역사와 구현 완성도, Extent 할당 기법 등으로 인해 실제로는 Ext4가 더 빠른 성능을 나타

냄을 알 수 있다. BtrFS의 경우는 그 설계 이념에 맞게 데이터 입출력을 수행하지만, 최근 개발되고 있는 SSD의 순차쓰기/임의쓰기의 성능 격차가 해소됨에 따라 실제 순차쓰기로 바뀐 데이터 쓰기 요청은 성능 향상에 큰 효과를 보지 못함을 알 수 있다.

4. 결론 및 향후 연구

실험을 통해 Androbench, Gmail, KakaoTalk 세 가지의 모든 애플리케이션 쿼리에 대해서 Ext4가 가장 빠른 성능을 나타내는 것을 볼 수 있었다. 이를 통해 플래시메모리에 최적화되었다고 주장하는 파일시스템들도 아직까지는 더 발전이 필요한 것을 알 수 있었다. 본 논문에서의 실험 결과는 이미 개발되었고 공신력 있게 받아들여지고 있는 파일시스템을 기반으로, SQLite 데이터베이스나 운영체제의 소스코드 변화 없이 곧바로 상용 모바일 플랫폼에 적용할 수 있는 변화들을 보여주고 있다. 또한, 실제 파일시스템들의 데이터 입출력 패턴 분석을 통해 해당 파일시스템의 구현 방식에 따른 특성을 분석할 수 있었고, 순차 쓰기/임의 쓰기와 같은 특성에 따른 미래의 파일시스템에 대한 가능성을 확인할 수 있다.

향후 실제 파일시스템 구현 방식에 대한 소스 코드 레벨에서의 분석을 통해, 플래시메모리에 최적화되었다고 주장하는 파일시스템들이 더 느린 이유를 자세히 분석하려고 한다. 이러한 분석을 통해 차세대 파일시스템이 갖춰야 할 구조적 특징을 제시할 것이다.

사사표기

1) 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (R0126-15-1108, FlashSQL:비휘발성 메모리 기반 개방형 고성능 DBMS 개발)
2) 이 논문은 2014년도 정부(산업통상자원부)의 재원으로 산업기술 기술혁신사업의 지원을 받아 수행된 연구임 (10049445, 모바일 저장장치용 UFS 2.0 제어기 SoC 및 임베디드 SW 개발)

참고문헌

[1] SQLite , http://www.sqlite.org/index.html

[2] Won Young Lee, "SQLite Optimization Strategy on Tizen" , SKKU, 2013

[3] Hyojun Kim, Nitin Agrawal, and Cristian Ungureanu. "Revisiting Storage for Smartphones". In USENIX FAST '12, pages 17–29, 2012.

[4] Sooman Jeong, Kisung Lee, Seongjin Lee, Seoungbum Son, and Youjip Won. "I/O Stack Optimization for Smartphones". In USENIX Annual Technical Conference '13, pages 309–320, 2013

[5] KT경제경영연구소, '2015년 상반기 모바일 트렌드' 보고서, 2015년 8월

[6] Gihwan Oh, Sangchul Kim, Sang-Won Lee, Bongki Moon, "SQLite optimization with Phase Change Memory for Mobile Applications", VLDB 2015, Aug, 2015

[7] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Gi-Hwan Oh, Changwoo Min, "X-FTL: Transactional FTL for SQLite Databases", SIGMOD 2013.06, New York

[8] Ext4, https://ext4.wiki.kernel.org/

[9] XFS, http://xfs.org/

[10] BtrFS, https://btrfs.wiki.kernel.org/

[11] Ohad Rodeh, Josef Bacik, and Chris Mason. 2013. BTRFS: The Linux B-Tree Filesystem. Trans. Storage 9, 3, Article 9 (August 2013), 32 pages.