fsync 지연시간 개선을 위한 저널링 기법 연구

박대준°, 신동군
성균관대학교 전자전기컴퓨터공학과
pdaejun@skku.edu, dongkun@skku.edu

Fine-Grained journaling for Reducing Fsync System Call Latency

Daejun Park°, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

저널링은 시스템 장애 상황에서 fsck의 느린 파일시스템 복구 시간을 단축하기 위해 제안된 일관성 관리 기법이다. 현재의 저널링은 효율적인 쓰기를 위해 여러 파일 연산의 결과를 모아서 한꺼번에 기록하는 컴파운드 트랜잭션을 사용하기 때문에 fsync와 같이 응답성이 중요한 파일 연산이 다른 파일 연산과 함께 처리되어 늦어지는 현상이 발생한다. 본 논문에서는 일반적인 상황에서는 기존 저널링의 컴파운드 트랜잭션을 그대로 사용하여 쓰기 증폭 문제를 해결하면서도, fsync가 발생하였을 때는 컴파운드 트랜잭션을 대신해 파일 수준의 저널을 기록하고 복구 과정에서 파일시스템의 일관성을 보장하는 방법으로 fsync 연산의 응답성 문제를 해결하였다. 제안한 기법을 ext4 파일시스템을 기반으로 구현하여 스마트폰에서 수행한 데이터베이스 벤치마크에서 52% 더 향상된 성능을 얻었다.

1. 서 론

버퍼를 활용한 입출력은 스토리지의 느린 속도를 보완하기 위해 주기억 장치에 쓰기 버퍼를 할당하여 빠른 입출력을 가능하게 하는 기법이다. 그러나 주기억 장치로 사용되는 DRAM은 휘발성이기 때문에, 갑작스런 시스템 장애 발생시 버퍼링 된 쓰기가 스토리지에 적용되지 않을 수 있다. 파일시스템의 메타데이터와 데이터 중 한 정보라도 스토리지에 적용이 누락된다면, 파일 시스템은 잘못된 정보를 가지게 되고, 이 상태를 일관성이 깨졌다고 한다.

시스템 장애가 발생 했을 때, 일관성을 복구하기 위해 fsck를 통해 파일시스템에 기록된 메타데이터와 데이터를 읽어 들여 문제가 있는 정보를 교정할 수 있다. fsck는 파일시스템의 오류 상태를 복구하는 유틸리티로, 스토리지의 크기가 커질수록 복구시간이 길어지는 단점이 있다. 그래서 복구 시간을 줄이기 위해 저널링이 제안되었다[1].

저널링에서는 파일 연산이 일어날 때 발생하는 쓰기를 실제 스토리지의 위치에 기록하기 전에, 쓰기의 복사본을 저널 영역에 먼저 기록한다. 이 기록 과정을 커밋(commit)이라고 하고, 이러한 순서를 지키는 것을 WAL(Write Ahead Log)이라고 한다. 저널링을 사용하면 WAL을 통해 파일시스템의 최근 변경사항이 저널 영역에 기록되어 있다는 것을 보장 받는다. 그래서 시스템 장애가 일어나게 되면 저널 영역에 커밋된 변경사항이 담긴 기록을 읽어 실제 스토리지의 위치에 기록하는 간단한 과정을 통해 복구 할 수 있다. 그러나 저널링을 사용할 경우, 파일 연산 별로 변경된 메타데이터를 기록하기 위해 커밋 시 로깅 되는 양이 많아지는 쓰기 증폭 현상이 발생한다. 이는 파일시스템의 다른 파일 연산을 방해하여, 전체적인 성능에 악영향을 끼친다. 그런데 일반적으로 파일 연산들은 서로 공유하는 메타데이터를 변경하는 경우가 잦다. 그래서 ext4에서 사용하는 JBD2에서는, 5초안에 발생하는 파일 연산을 묶어서 커밋하는 컴파운드(compound) 트랜잭션 기법을 사용한다. 이를 통해 빠른 복구를 유지하면서도 쓰기 증폭 현상을 완화시켜, 저널링이 파일 시스템의 전체적인 성능에 미치는 영향을 최소화 시킬 수 있다. 그런데 컴파운드 트랜잭션에는 여러 파일 연산이 함께 존재하므로, 기존 방식에 비해 각각 발생하는 단일 커밋 지연시간이 길어 질 수 있는 단점이 있다. 그러나 JBD2가 발생시키는 주기적인 커밋은 비 동기 적으로 처리되므로 커밋의 지연 시간이 성능에 영향을 끼치지는 않는다.

일관성 문제와 별개로, 버퍼를 활용한 입출력을 사용하면 수행한 파일 연산에 대해 바로 내구성이 보장되지 않으므로, 필요한 경우 어플리케이션은 fsync 시스템 콜(system call)을 사용하여 특정 파일에 대한 즉시 내구성을 요구해야 한다. 많은 데이터베이스 관리 시스템에서는 데이터베이스 파일에 대해 fsync를 사용하여, 수행한 파일 연산에 대해 즉시 내구성을 보장받는다. 또한 안드로이드와 같은 모바일 플랫폼에서는, 시스템 설정이나 어플리케이션의 설정을 저장하기 위해, XML파일에 fsync를 사용한다. 그런데 fsync는 즉시 내구성이 보장 될 때 반환되는 동기적 방식을 사용하는 시스템 콜이므로 fsync의 지연시간은 어플리케이션의 성능과 밀접한 관계가 있다. 저널링을 사용하면 fsync를 처리할 때 일관성을 유지하기 위해, fsync의 대상이 되는 파일이 포함된 컴파운드 트랜잭션을 커밋하여 즉시 내구성을 보장한다.

그런데 이때 기록되는 컴파운드 트랜잭션에는 fsync와 관련 없는 파일의 파일 연산도 포함되므로 커밋 시간이 길어져 fsync의 지연시간에 영향을 끼칠 수 있다. 특히 Ext4에서의 기본 저널링 모드인 순차(ordered) 저널 모드를 사용할 경우에는, 일반데이터를 실제 스토리지 위치에 기록하고 이를 기다리는 시간이 커밋 시간에 포함된다. 그러므로 fsync와 관련 없는 파일의 버퍼된 데이터들이 커밋의 응답시간을 지연시킬 수 있다. 만약 각각의 파일에서 발생하는 파일 연산들을 별도의 트랜잭션으로 관리하여 저널링을 수행한다면 fsync의 파일과 관련 없는 데이터로 인한 fsync 응답시간 증가 문제는 발생하지 않을 것이다. 그러나 주기적인 커밋에서 공유 메타데이터에 대한 쓰기 증폭을 발생시켜 전체적인 입출력 성능을 떨어뜨릴 수 있다.

본 논문에서는 이 문제를 해결하기 위해, 주기적인 커밋에서는 일반 저널링을 통해 일관성을 유지하고, fsync 발생 시에는 파일수준 저널링을 사용한다. 파일 수준 저널링에는 fsync와 관련된 파일의 메타데이터만을 로깅 하여 컴파운드 트랜잭션으로 인한 fsync의 응답시간을 줄인다. 그러나 컴파운드 트랜잭션 중에서 일부의 메타데이터만 복구한다면 파일시스템의 일관성이 깨지게 된다. 그래서 제안된 기법에서는 파일시스템 복구 시에 저널링에서 사용하는 단순한 복사 과정 후에, fsync 시 기록한 파일레벨 저널을 바탕으로 다른 파일시스템 메타데이터도 복구하여 파일시스템이 일관성 있는 상태를 유지할 수 있게 한다. 제안된 기법을 사용하면 파일 수준 저널링을 통해, 복구

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 서울어코드활성화지원사업의 연구결과로 수행되었음
(IITP-2015-R0613-15-1062)
본 연구는 미래창조과학부 및 정보통신기술진흥센터의 SW중심대학-ICT/SW창의연구과정 지원사업의 연구결과로 수행되었음
(IITP-2015-R2215-15-1005)

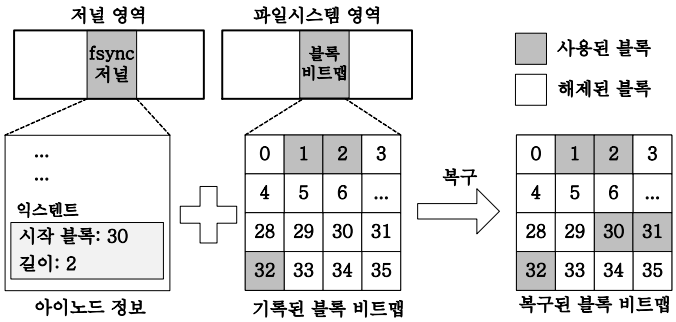

그림 1 블록 비트맵 복구 과정

시 스토리지 전체 메타데이터의 비교과정 없이 일부 메타데이터의 비교만으로 교정 할 수 있다. 본 논문에서는 제안한 기법을 ext4 파일시스템을 기반으로 구현하여 스마트폰에서 수행한 데이터베이스 관리 시스템에서의 성능 평가에서는 52% 더 향상된 성능을 얻었다.

2. 관련 연구

파일시스템의 성능 고립화를 통해 fsync의 지연시간을 줄이기 위한 연구로 IceFS[2]와 spanFS[3]가 있다. IceFS에서는 디렉터리를 기반으로 한 큐브(cube)라 불리는 단위를 통해 파일시스템의 고립성을 제공한다. 큐브 마다 메타데이터를 물리적으로 분리시켜, 저널링 또한 큐브 별로 구분되어 작동한다. 그러므로 서로 다른 큐브 간에는 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생하지 않는다. 그러나 동일 큐브 내에서는 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생한다. 또한 큐브 간에는 서로 디렉터리 구조를 공유할 수 없어, 사용자가 파일시스템을 사용하는데 제약사항이 발생한다.

spanFS는 IceFS의 큐브와 유사한, 도메인(domain)이라는 고립화 단위를 제공한다. 도메인은 큐브와 달리 도메인간의 디렉터리 구조 연결을 지원하므로, 사용자가 파일시스템을 사용하기에 제약사항이 없다. 그러나 spanFS 또한 같은 도메인 내에서는 컴파운드 트랜잭션으로 인한 fsync 지연 문제가 발생한다는 문제점이 있다.

reconFS[4]는 낸드 플래시 기반의 스토리지를 위한 파일시스템으로 디렉터리와 데이터에 대한 트리를 즉시 기록할 때 발생하는 지연시간을 줄이기 위해 트리를 기록하는 대신, 낸드 플래시 페이지의 여분 공간에 트리에 대한 역 인덱스를 기록한다. 역 인덱스에는 해당 페이지가 아이노드인 경우 부모 디렉터리에 대한 정보가 담겨 있고, 데이터인 경우 소속 아이노드에 대한 정보가 담겨 있다. 그러므로 역 인덱스를 통해 트리 정보를 재구성 할 수 있다. reconFS는 일관성을 유지하기 위해 필요한 모든 데이터를 로깅하지 않고도 최소한의 정보를 스토리지에 기록하고, 이를 이용해 일관성을 복구한다는 점에서 논문에서 제안하는 기법과 연관성이 있다. 그러나 저널링을 사용하는 환경에서 발생하는 fsync 지연문제에 대해서는 해결할 수 없다.

FFSCK[5]는 fsck의 비효율적인 입출력 패턴을 개선하고, 기존 파일시스템의 메타데이터 구조를 변경하여 복구 시간을 줄이고자 한 기법이다. 그러나 이 기법은 기존의 fsck에 비해 빠르지만 저널링에 비해서는 느린 복구 성능을 보여준다. 그리고 컴파운드 트랜잭션으로 인한 fsync 지연문제에 대해서는 해결할 수 없다.

4. fsync 저널

fsync 저널을 구현하기 위해, 본 논문에서는 대부분의 리눅스 배포판에서 사용되는 ext4 파일시스템을 선택하였다. fsync 저널에서는 컴파운드 트랜잭션에 포함되는 fsync와 관련 없는 파일 연산의 기록을 피하기 위해, fsync에 해당하는 파일의 일

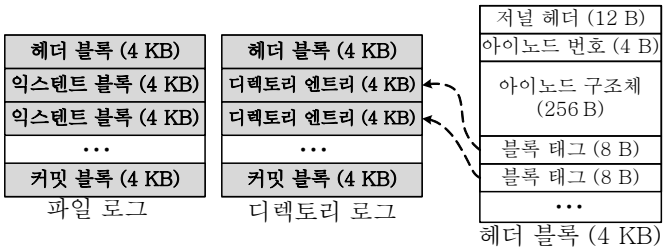

그림 2 fsync 저널의 구조

반데이터와 메타데이터만을 기록하고자 한다. 그러나 블록 비트맵과 아이노드 비트맵, GDT(group descriptor table)와 같은 **파일시스템 메타데이터**는 여러 파일에 대해 연관되어 있다. 그러므로 fsync 저널에서는 파일시스템 메타데이터들이 현재 가진 정보를 그대로 저널링 하는 것이 아니라 fsync에 해당하는 파일이 변경시킨 정보만을 추출해 기록해야 한다.

그러나 현재 구조에서는 해당 메타데이터들에서 fsync에 해당하는 파일이 변경시킨 정보만을 빠르게 추출 할 수 있는 방법이 없다. 이를 구별하기 위해서는 컴파운드 트랜잭션을 포기해야 하는데, 이는 fsync 지연시간의 향상을 위해 일반적인 입출력 성능을 포기해야 하므로 비효율적이다.

한편 아이노드와 익스텐트 블록은 하나의 파일에만 속하는 **파일 메타데이터**이다. 그러므로 파일이 변경시킨 정보를 추출하기 위해서는, 이전 버전과의 비교만으로 가능하다. 그래서 파일 메타데이터를 따로 로깅해 둔다면, 시스템 장애 발생 시 기존에 기록된 메타데이터와 비교를 통해 한 파일에 대해 어떤 파일 연산이 있었는지 파악 할 수 있다. 또한 이 파악된 정보를 통해 파일시스템 메타데이터 또한 한 파일에 대한 파일 연산을 적용시킬 수 있다. 그림 1은 아이노드와 익스텐트 블록만을 저널링 한 뒤, 하나의 파일에 해당하는 블록할당 정보를 추출하여, 기존의 블록 비트맵에 반영되지 못한 추가 정보를 나중에 반영하는 과정에 관한 것이다. 이러한 방법을 통해 파일시스템은 파일 메타데이터의 로깅을 통해 일관성을 유지할 수 있다. 이때 각 메타데이터의 정보를 비교하여 수정하는 방법은 fsck의 일관성을 유지하기 위해 비교하는 방법과 유사하다.

fsync 저널은 fsync 발생 시 하나의 파일에 해당하는 아이노드와 익스텐트 블록들을 저널링 한다. 그림 2는 fsync 저널의 구조에 대한 것이다. 저널 헤더에는 현재 fsync 저널의 버전을 기록되어 있으므로, 주기적인 커밋으로 기록되는 기존 저널과의 선후 관계를 파악 할 수 있다. 아이노드는 fsync를 발생시킨 파일에 대한 것이고, 블록 태그는 뒤이어 저널 되는 익스텐트 블록에 대한 주소 정보를 가지고 있다. 익스텐트 블록은 하나의 파일의 블록 할당에 대한 정보로, 아이노드 내에서 표현이 불가능할 경우 아이노드 외부에 기록한다. 그런데 큰 용량을 가지는 파일일 경우 기록해야 하는 익스텐트 블록의 수가 많아질 수 있으므로, 본 기법에서는 현재 변경 중인 익스텐트에 대해서만 fsync 저널에 기록한다. 그리고 fsync 저널의 마지막에는 커밋 블록을 기록하여 해당 fsync 저널이 유효하다는 것을 나타낸다. 또한 디렉터리 접근성을 위해 fsync에 해당하는 변경 중인 모든 조상 디렉터리에 대해, 디렉터리 엔트리를 포함한 fsync 저널을 기록한다.

제안된 기법에서 fsync가 일어날 경우, 먼저 해당 파일에 해당하는 일반데이터만을 스토리지에 기록한다. 그 다음, 컴파운드 트랜잭션을 기록하는 대신 fsync 저널을 기록하여 빠르게 커밋을 완료한다. fsync 저널은 트랜잭션에 해당하는 일부의 메타데이터만을 저널링한다. 그러나 주기적으로 발생하는 커밋의 경우, 기존 컴파운드 트랜잭션을 그대로 사용한다. 그래서 fsync 지연시간을 줄이면서도 쓰기 증폭 현상을 줄이는 기존 저널링의 장점을 유지시킬 수 있다.

복구 과정은 다음과 같다. 우선 기존 저널에 대해서만 복구를 수행한다. 그 후 fsync 저널이 있을 경우, 해당 저널을 복구해야 할지 기존 저널과의 버전 비교를 통해 확인한다. fsync

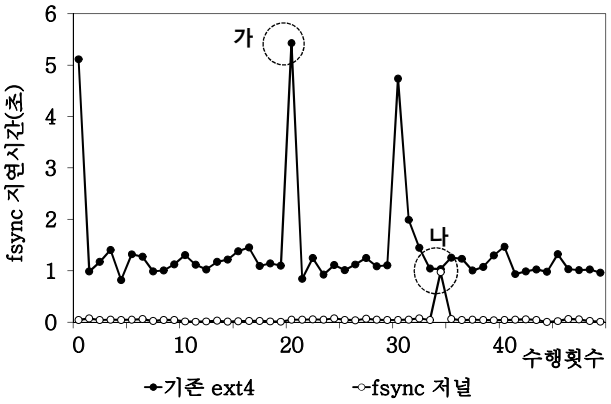

그림 3 기법 별 fsync 지연시간 측정

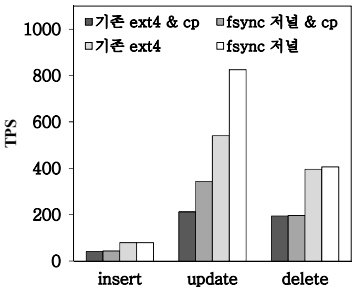

그림 4 저널링 기법 별 롤백저널 성능 평가

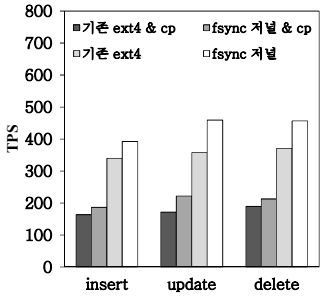

그림 5 저널링 기법 별 WAL 성능 평가

저널에 기록된 메타데이터는 기존 컴파운드 트랜잭션의 부분집합이므로, 주기적으로 발생하는 커밋은 이전에 발생한 fsync 저널을 포함하고 있다. 그러므로 fsync 저널의 버전이 기존 저널보다 최신일 경우에만, fsync 저널을 위한 복구를 시작한다. fsync 저널을 위한 복구에서는 fsync 저널에 기록된 아이노드와 익스텐트 블록에 대해 파일시스템에 기록된 버전과 서로 비교하는 과정을 거친다. 이를 통해 블록 할당이나 해제에 대한 정보를 획득할 수 있고, 이를 통해 블록 비트맵이나 아이노드 비트맵과 같은 파일시스템 메타데이터를 수정한다. 모든 수정이 끝난 후에, 파일시스템의 아이노드 속성은 fsync 저널에 기록된 정보로 갱신하고, 익스텐트 또한 fsync 저널에 기록된 것으로 갱신한다.

3. 실 험

제안된 저널링의 실험을 위해 실험 기기로 eMMC를 스토리지로 사용하는 Galaxy S4(SHV-E300s)를 선택하였다. 기기의 사양은 exynos5(ARM Cortex- A15 Quad 1.7GHz + Cortex- A7 Quad), 32GB eMMC, 2GB DRAM이다. 안드로이드 버전은 4.2.2(Jelly bean)이고 리눅스 커널 버전은 3.4.5이다. 그리고 ext4 파일시스템은 순차 저널 모드를 사용하였다.

제안된 기법을 사용하여 복구가 제대로 되는 지, 시나리오에 따라 인위적으로 시스템 장애 상황을 만들어 확인하였다. 첫 번째 시나리오는 파일을 생성하고 4KB씩 블록을 연속적으로 할당받아 이어서 쓰기를 수행하였다. 그리고 마지막으로 fsync를 발생시켜 완료 한 직후 시스템 장애를 발생시켰다. 그 결과 복구 과정에서 fsync 저널을 통해 해당 파일이 올바르게 복구되어 파일시스템의 일관성을 유지하였다. 두 번째 시나리오에서는 무작위 파일에 대해 mkdir, create, write, truncate 연산을 무작위로 수행하고 fsync를 완료 한 직후 시스템 장애를 발생시켰다. 복구 과정이 끝난 후 살펴본 결과, 모든 파일에 대해 파일 연산들이 적용된 상태로 파일시스템의 일관성을 유지하였다. 두 시나리오 모두 fsck를 활용해 파일시스템의 일관성을 확인하였다.

그림3은 기존 저널링과 fsync 저널을 사용하였을 때 fsync 지연시간을 비교한 것이다. 컴파운드 트랜잭션으로 인한 문제를 재현하기 위해 지속적으로 쓰기 연산을 수행하는 쓰레드를 4개 만들고, 별도 쓰레드로 fsync를 1초마다 반복하여 수행하면서 지연시간을 측정하였다. 그래프를 보면, 전체적으로 fsync 저널을 사용했을 때, 기존의 저널링에 비해 매우 낮은 fsync 지연시간을 보이는 것을 알 수 있다. 특히 "**가**" 부분을 보면 fsync 지연시간이 급격히 증가한 것을 볼 수 있는데, 이는 커밋 시 먼저 기록되어야 하는 일반데이터의 쓰기가 다른 쓰기 연산에 의해 지연되어 발생하는 현상이다. fsync 저널을 사용할 때도 비슷한 현상이 발생하는데 이는 "**나**"에서 볼 수 있다. 그러나 fsync 저널은 커밋 시 기록해야 하는 일반데이터의 크기가 기존기법에 비해 매우 적기 때문에 매우 큰 지연시간은 보이지 않는다.

그림 4와 5는 데이터베이스 벤치마크 프로그램인 Mobibench[6]를 수행한 결과이다. Mobibench는 8개의 thread를 통해 트랜잭션을 1000개 수행하도록 하였고 SQLite를 DBMS로 사용하였으며, SQLite에서의 저널 모드는 롤백 저널과 WAL을 사용하였다. 또한 데스크톱에서 3GB의 파일을 디바이스로 USB를 통해 복사하는 명령을 병행하였다.

그림 4에서는 fsync 저널이 기존 저널기법에 비해 최대 52% 높은 성능을 보여주었는데, 이는 데이터베이스 관리 시스템의 잦은 fsync 호출로 인한 것이다. 데이터베이스 파일과 롤백 저널 파일에 쓰기를 수행 할 때마다 fsync를 발생시키는데, fsync 저널은 fsync에 해당하는 데이터만 기록하므로 지연시간이 기존 기법에 비해 상대적으로 낮다. 그래서 fsync 저널을 사용할 때, 데이터베이스 관리 시스템이 높은 성능을 보여주게 된다. 그림 5는 WAL 모드를 사용한 성능 측정 결과이다. WAL 모드는 롤백 저널에 비해 전체적으로 좋은 성능을 보이고 fsync 저널이 기존 저널기법에 비해 높은 성능을 보여주고 있다.

4. 결 론

fsync 저널은 컴파운드 트랜잭션으로 인해 발생하는 다른 파일 연산이 발생시키는 fsync 지연문제를 해결하기 위해 제안되었다. 파일시스템의 메타데이터 중 하나의 파일에만 연관되어 있는 아이노드와 익스텐트를 활용하여, 컴파운드 트랜잭션에서 하나의 파일에 해당하는 파일 연산만을 적용시킬 수 있었다. 복구 과정에서는 기존 저널링 기법과는 다르게 fsck에서 사용하는 메타데이터 간 확인과 수정 과정을 일부 적용하여, 아이노드와 익스텐트 이외의 다른 메타데이터도 최신의 일관성 있는 정보를 가질 수 있도록 하였다. 향후 연구로는 데스크탑에서 fsync 저널을 적용하여 fsync의 성능과 부하에 대해 관찰하고자 한다.

참고문헌

[1] S. C. Tweedie. Journaling the Linux ext2fs File System. In The Fourth Annual Linux Expo, 1998.
[2] Lanyue Lu, *et al.* Physical disentanglement in a container-based file system. In Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation, 2014
[3] Kang, Junbin, et al. SpanFS: a scalable file system on fast storage devices. Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference, 2015.
[4] Lu, Youyou, Jiwu Shu, and Wei Wang. ReconFS: a reconstructable file system on flash storage. FAST, 2014.
[5] Ma, Ao, et al. Ffsck: The Fast File-System Checker. ACM Transactions on Storage, 2014
[6] Jeong, Sooman, et al. AndroStep: Android Storage Performance Analysis Tool. Software Engineering, 2013.