HW12

2017314712 김우진

John Ousterhout와 Mendel Ronsenblum이 로그 기반 파일 시스템이라는 새로운 파일 시스템을 개발하였습니다. 메모리 크기가 증가 추세, 임의 I/O와 순차 I/O의 성능 간의 격차가 크게 벌어짐, 많은 일반적인 워크로드에서 기존 파일 시스템들의 성능이 좋지 않음, 파일 시스템이 RAID를 고려하지 않음이 동기 부여가 되어 연구를 하게 되었다.

이상적인 파일 시스템은 쓰기 성능에 초점을 둔다. Rosenblum과 Ousterhout가 생각한 새로운 종류의 파일 시스템은 로그 기반 파일 시스템 또는 LFS라고 불린다. 디스크 기록시, LFS는 모든 갱신 정보를 세그멘트에 저장한다. 세그멘트가 가득차면 디스크에 기록한다. LFS는 기존의 내용을 덮어쓰지 않고 비어있는 곳에 쓴다. 즉 디스크 성능을 최대한 활용할 수 있다.

데이터 블록을 기록하면 데이터만 갱신되는 것이 아니라, 메타데이터도 같이 갱신되어야 한다. 아이노도(I)도 디스크에 기록해야되고 아이노드가 데이터블럭을 가르키도록 해야한다. 모든 갱신을 디스크에 순차적으로 기록하는 것이 LFS의 핵심이다.

디스크에 순차적으로 쓰는 것만으로 효율적인 쓰기를 보장할 수 없다. 다수의 순차쓰기를 한 번에 디스크에 내려 보내야 빠른 쓰기 성능을 얻을 수 있다. LFS는 쓰기 버퍼링이라는 기법을 사용한다. 디스크에 쓰기 전에 LFS는 갱신 내용을 모두 메모리에 보관한다. 디스크에 데이터를 기록할 때 쓰기 내용들을 세그멘트 버퍼에 유지하고, 시크멘트가 차면 세그멘트를 버퍼를 한 번의 쓰기 연산으로 디스크에 기록한다.

디스크의 물리적 특성 및, 전송 속도 대비 위치 잡기 오버헤드에 의해 세크멘트 버퍼에 담는 내용의 크기가 달라진다.

D = F 1 − F × Rpeak × Tposition 계산하면 이런 식을 얻을 수 있다.

전통적인 파일 시스템에서는 각 아이노드의 위치가 정해져있다. 배열을 이용하면 아이노드의 위치를 빠르게 찾을 수 있다. LFS의 경우 아이노드가 디스크 전역에 흩어져 있을 뿐만 아니라 원 위치에 덮어쓰기를 하지 않기 때문에 찾기가 복잡하다. 이를 찾기 위해서 LFS 개발자들은 아이노드맵이라는 자료 구조를 개발하였다. 아이노드 번호를 입력으로 하여 가장 최신 아이노드의 디스크 위치를 구한다. 디스크에 아이노드가 기록될 때 imap은 새로운 위치를 가리키게 바뀐다.

Imap은 자주 갱신되기 때문에 디스크의 고정된 위치에 배치하게 된다. LFS에서는 아이노드 맵을 새로이 기록된 데이터와 아이노드를 옆에 함께 기록한다. 하지만 결론적으로는 아이노드 맵도 찾아야한다. 정해진 위치에 검색을 시작하는 데 필요한 정보를 저장한다. LFS는 디스크의 약속된 위치에 imap 블록들의 위치를 기록하며 이를 체크포인트 영역이라 부른다. 체크포인트 영역은 주기적으로 갱신이 되기 때문에 성능에 약영향을 주지 않는다. 일반적인 Unix 파일 시스템과 같이 아이노드 맵 조각들은 아이노들들의 주소들을 포함하며, 아이노드들은 파일들을 가리킨다.

디렉터리는 매핑 정보로 구성되어 있다. 파일을 생성할 때 LFS는 새로운 아이노드와 데이터 그리고 파일을 가리키는 디렉터리 데이터와 디렉터리 아이노드도 같이 써야 한다. LFS는 이 정보들을 순차적으로 기록한다. 아이노드맵은 재귀 갱신 문제라는 심각한 문제를 해결한다. 아이노드 위치가 변경되더라도 변경 내용은 디렉터리 내에 직접 반영하지 않는다.

LFS는 갱신된 파일을 계속 디스크의 새로운 위치에 쓴다. 남아있는 예쩐값을 가비지라 부른다. 파일의 여러 번을 관리하는 파일 시스템을 버전 파일 시스템이라 부른다. 하지만 LFS는 파일의 최신 버전만을 우지한다. LFS는 주기적으로 이전 버전의 데이터와 아이노드 그리고 다른 자료 구조들을 찾아 제거한다. 만약에 LFS의 가비지 컬렉터가 데어터 블록과 아이노드 등을 하니씩 순회하며 해제하면 몇 개의 구멍과 할당 된 공간이 섞여 있는 상태가 된다. 그래서 LFS의 가비지 컬렉터는 주기적으로 오래된 세그멘트들을 읽은 후 최신 블록들을 새로운 세그멘트로 이동한다.

LFS는 각 테이터 블록에 속하는 파일의 아이노드 번호와 파일 내에서 오프셋을 저장한다. 이 정보는 세그멘트의 첫 머리에 세크멘트 요약 블록이라 부르는 자료 구조에 기록된다. 하지만 성능의 문제가 있어 파일을 삭제했을 때 LFS는 버전 번호를 증가시키고 새로운 버전 번호를 imap에 기록해 둔다.

어느 블록들을 가비지 컬렉션할지 정하는 것은 도전적이다. 세그멘트를 핫과 콜드로 구분하는 방법이 있다. 하지만 이 방법도 문제가 있고 더 좋은 방법들이 많이 소개되었다.

LFS는 쓰기 데이터를 세그멘트 버퍼에 먼저 기록하고 해당 세크멘트 버퍼를 디스크에 기록하고 해당 세그멘트 버퍼를 디스크에 기록한다. 이것들은 로그로 구성된다. 전체 세그멘트들이 일종의 링크드 리스트로 연결되는데 체크포인트 영역의 해당 정보가 갱신될 때 크래시가 발생할 수 있다.

이것을 방지하기 위해 LFS는 두 개의 체크포인트 영역을 둔다. 체크포인트 영역을 디스크 양 끝에 위치시키고 교대로 갱신한다. LFS는 먼저 체크포인트 헤더를 기록한 후에 체크포인트 영역에 내용을 쓰고, 그리고 최종적으로 체크포인트 영역의 마지막 블록을 갱신한다. 크래시가 발생할 경우 LFS는 헤더와 마지막 영역에 저장된 시간값을 비교한다. 헤더에 저장된 시간값이 더 크다면, 해당 체크포인트 영역을 갱신하는 도중 크래시가 발생한 것이다.

재부팅 시 복구 과정에서 LFS는 체크포인트 영역을 읽어서 imap 조각들이 가리키는 곳을 확인하여 파일들과 디렉터리들을 복원한다. 하지만 마지막 수초간의 갱신 내용들이 손실되는 것을 피할 수 없다. 이 문제의 개선을 위해 LFS는 데이터베이스에서 사용되는 롤 포워드 기법을 적용한다. 체크 포인트 영역을 읽어들여 세크멘트 리스트의 마지막 세크멘트 위치를 파악한다. 세그멘트가 가리키는 다음 세그멘트의 존재 여부를 검사하고, 필요하면 체크포인트의 마지막 세그멘트를 가리키는 포인터를 갱신한다. 세그멘트 리스트를 순회하며 마지막 체크포인트 이후의 데이터와 메타데이터를 복구한다.