네트워크 스토리지 상에서의 데이터 복구를 위한 스냅샷 개발

석진선[O1], 김선태[1], 노재춘[1], 박성순[2]
[1]세종대학교 컴퓨터 공학부 시스템공학 연구실
durgatm@gmail.com, kimst4444@gmail.com, jano@sejong.ac.kr
[2](주) 글루시스

snap + XFS

Jinsun Suk[O1], Suntae Kim[1], Jaechun No[1], Sungsun Park[2]
[1]System engineering Lab, School of Computer Sci. & Eng., Sejong University
[2]Gluesys Inc.

요 약

다양한 스토리지와 파일 시스템들이 시스템의 신뢰도를 증가시키기 위해 스냅샷을 이용하고 있다. 하지만 현재 널리 사용되고 있는 볼륨 단에서의 스냅샷은 스냅샷 이미지를 생성하는데 필요한 시간이 볼륨의 크기에 비례하고 스냅샷 이미지를 생성하는 동안의 `파일시스템 입출력 성능'이 현저하게 저하되는 단점이 있다. 이러한 단점을 극복하기 위해 파일 시스템 단에서의 스냅샷 기법에 대한 연구가 진행되어 왔으며 “snapFS”, “Ext3Cow” 그리고 “New version of SnapFS” 등의 파일 시스템들이 개발 되었다.
본 논문에서는 네트워크로 연결된 스토리지 상에서의 데이터 복구를 효율적으로 처리하기 위해 대용량 파일을 처리하는데 적합한 XFS에 스냅샷 기능을 추가한 snap+XFS에 대해 언급한다.

1. 서 론

다양한 스토리지와 파일 시스템들은 신뢰도와 효율성을 향상시키기 위해 스냅샷을 이용한 데이터 버전닝을 사용하고 있다. 스냅샷으로 파일 시스템의 어느 한 시점을 보관하는 것은 백업을 위한 일관성 있는 이미지를 유지하고 데이터를 재사용하기에 용이하도록 보관하는 유용한 방법이다. 리눅스에서의 스냅샷은 Logical Volume Manager와 같이 볼륨 단에서 수행되는 것과 snapFS와 같이 파일 시스템 단에서 수행되는 것으로 나뉘어진다.[6] 볼륨 단에서 스냅샷을 수행하는 경우에는 볼륨의 스냅샷 이미지를 저장하기 위한 큰 공간이 필요하고 볼륨에 저장되어있는 많은 양의 데이터를 처리해야 하기 때문에 많은 시간이 필요하다. 또한 스냅샷 이미지를 생성하기 위한 작업의 부하로 인해 전체 시스템의 입출력 성능이 저하된다. 그러나 파일 시스템을 기반으로 스냅샷 이미지를 생성하는 경우에는 볼륨보다 훨씬 작은 단위인 파일 또는 블록 단위로 생성되기 때문에 볼륨을 기반으로 한 스냅샷보다 빠르게 처리된다.
이러한 파일 시스템단에서 수행되는 스냅샷의 장점 때문에 SnapFS[8], xt3cow[4][5], New version of SnapFS[6]과 같은 다양한 연구들이 시도되었다. 본 논문에서는 XFS[1][2][3]에서 데이터 버전닝을 수행하기 위한 스냅샷[7]을 구현했다.

2. 스냅샷

스냅샷은 COW(Copy On Write) 또는 ROW(Redirect On Write)으로 처리된다.[7] COW 스냅샷은 원본 블록의 내용을 수정하기 전에 복사본을 생성하여 스냅샷 파일에 연결하고 원본 블록의 내용을 수정하는 방식이고 ROW 스냅샷은 원본 블록을 스냅샷 파일에 연결하고 수정된 데이터를 기록하기 위한 블록을 새로 할당받아 원본 파일에 연결하여 사용하는 방식이다. COW 스냅샷은 원본 블록을 다른 곳에 복사하여 스냅샷 블록을 생성하고 원본 블록을 수정하는 두번의 쓰기 작업이 필요한 반면 ROW 스냅샷은 원본 블록을 그대로 스냅샷 블록으로 사용하고 새로운 블록을 할당받아 수정된 내용을 기록하기 때문에 한번의 쓰기 작업만으로 스냅샷이 처리 속도가 빠른 장점이 있다.

본 논문에서 구현한 snap+XFS은 두번째 방식인 ROW로 구현되었다.

2.1 ROW

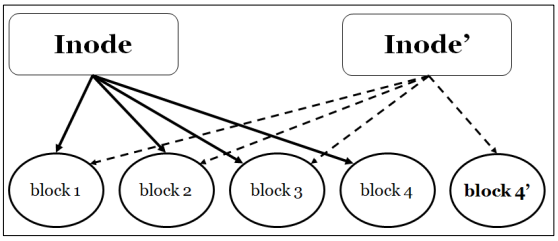

그림1. ROW 알고리즘

그림[1]은 ROW 방식의 snapshot으로 동작한 결과물로 다음과 같은 절차를 거쳐 생성된다. 그림[1]의 Inode와 블록1, 블록2, 블록3, 블록4로 구성된 원본 파일의 스냅샷을 찍으면 그림[1]의 오른쪽에 있는 Inode'가 생성되

본 논문은 Seoul R&BD Program(10557) 지원을 받아 수행되었음

면서 원본 파일과 동일하게 블록1, 블록2, 블록3, 블록4를 가리키게 된다. 이 후 원본 파일의 블록 4번을 수정하면 원본 파일은 블록4번에 관한 연결을 끊고 변경된 데이터의 내용을 담을 새로운 블록을 할당받아 그림[1]과 같은 모습으로 구성된다.

3. XFS

XFS는 IRIX에서 사용하기 위해 SGI가 개발한 저널링 파일 시스템으로 1999년 초에 Open Source 로 공개되면서 리눅스 커널에 포함되었다. XFS는 64bit 기반으로 구현되어 다수의 큰 파일들을 처리하는데 효과적인 파일 시스템이다.

4. XFS+snap의 ROW

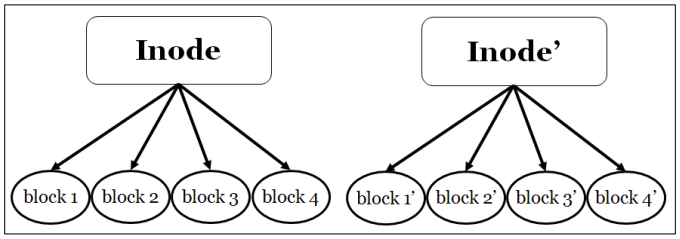

그림 2. snap+XFS의 스냅샷 이미지 생성

본 논문에서 구현한 snap+XFS는 파일 시스템 단에서 그림[2]와 같이 파일 단위로 스냅샷 이미지를 생성하며 생성 절차는 다음과 같다. 스냅샷 파일은 사용자에 의해 snapshot 명령어가 실행되는 시점에서 생성되며 생성된 시점에는 원본 파일과 동일한 블록을 가리킨다. 그러나 스냅샷 파일이 생성된 이 후, 원본 파일의 데이터가 수정되면 원본 파일은 새로운 블록을 할당받아 수정된 내용을 기록하기 때문에 더 이상 블록을 공유하지 않는다. 사용자가 수정한 원본 파일의 데이터는 모두 새로 할당된 블록에 기록되어 원본 파일의 아이노드에 연결된다. 그림[2]는 스냅샷이 존재하는 원본 파일을 수정했을 때 원본 파일의 변화를 표현한 것이다. 그림의 블록1, 블록2, 블록3, 블록4는 원본 파일의 블록을 의미하는 것이고 블록1', 블록2', 블록3', 블록4'는 원본 파일의 블록들을 대신하기 위해 새로 할당된 블록을 의미하는 것으로 원본 파일의 내용을 수정하는 동시에 원본 파일에 있던 원본 블록들이 스냅샷 이미지 파일에 연결되고 원본 파일은 새로 할당된 블록들만으로 구성된다. 파일이 수정되는 경우에 위와 같은 작업을 수행함으로써 스냅샷 이미지 파일이 수정 전 파일의 내용을 유지할 수 있게 된다.

5. snap+XFS의 구현

snap+XFS은 그림[3]처럼 세 가지 요소로 구성되며 각 구성원의 역할은 다음과 같다. 스냅샷 유틸리티는 사용자에 의해 동작하며 원본 파일과 동일한 블록을 공유하는 스냅샷 파일을 생성한다. 스냅샷 데몬은 사용자가 정해준 일정 시간을 주기로 스냅샷 유틸리티를 불러 스냅샷 파일을 생성한다. 커널 영역은 위에서 언급한 ROW 방식의 스냅샷을 수행하기 위해 수정되었다.

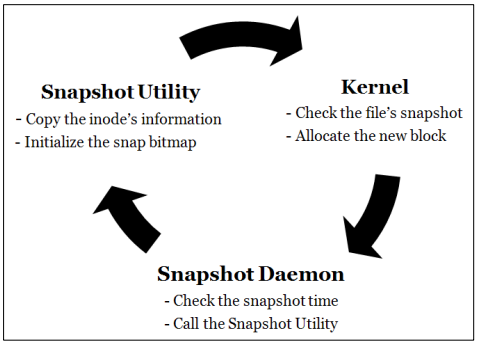

그림3. snap+XFS의 구성

5.1 스냅샷 유틸리티

스냅샷 유틸리티는 사용자가 스냅샷 파일을 만들기 위해 사용되는 사용자 유틸리틸로 입력값으로 스냅샷 이미지 파일을 생성하고자 하는 파일의 경로명을 받아 다음과 같은 절차로 동작한다. 스냅샷 유틸리티는 입력받은 파일의 아이노드 값을 새로 할당한 아이노드에 복사한다. 그리고 새로 할당한 아이노드에 ``원본 파일명_스냅샷 파일의 생성 시간" 형태의 파일 명을 부여하여 스냅샷 이미지 파일을 생성한다.

5.2 스냅샷 데몬

스냅샷 데몬은 스냅샷 이미지 파일을 주기적으로 생성하는 경우에 사용되는 것으로 입력 값으로 초 단위의 시간을 입력받아 입력 값을 주기로 스냅샷 유틸리티를 호출하여 스냅샷 이미지 파일을 생성한다.

5.3 Snapshot 파일의 관리

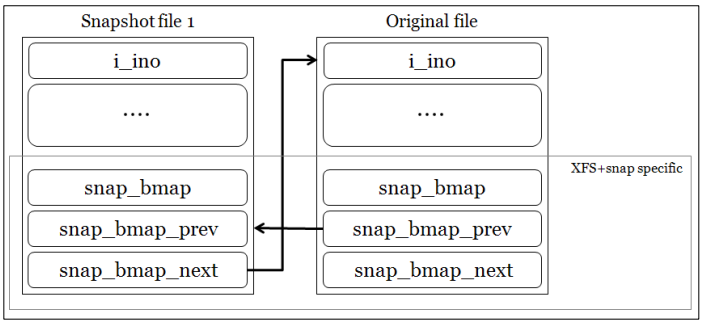

그림4. 스냅샷 파일의 관리

우리는 snap+XFS의 아이노드에 xfs_snap_prev, xfs_snap_next 인자를 추가하여 동일한 원본 파일의 스냅샷 파일들을 하나의 링크드 리스트로 연결, 관리하였다. 이렇게 스냅샷 파일과 원본 파일을 하나로 연결해서 관리한 이유는 수정하고자 하는 파일의 스냅샷 이미지 파일의 존재 유무를 판단하기 용이하게 하기 위한 것이다. 또한 후에 추가될 파일의 삭제 작업을 용이하게 하기 위한 것이다.

5.4 스냅샷 비트맵

원본 파일의 스냅샷 파일이 존재하는 경우에도 스냅샷 파일과 블록을 공유하지 않는 경우에는 블록을 새로 할

당 할 필요가 없다. 그래서 스냅샷 파일과 원본 파일이 블록을 공유하는지 여부를 판단해야 하는데 우리는 이것을 판단하기 위해 XFS의 아이노드에 snap_bmap이라는 인자를 추가하여 블록을 공유하는 경우에는 “0” 값을 가지게 하고 블록을 공유하지 않는 경우에는 “1“ 값을 가지도록 했다.

6. 성능 측정

model name	AMD Athlon(tm) XP 2500+
MHz	1832.754
cache size	12KB
RAM	512M

성능 측정을 위해 위와 같은 환경의 컴퓨터를 사용 했으며 모든 테스트의 결과 값은 동일한 테스트를 10번씩 수행하여 나온 평균값을 사용하였다.

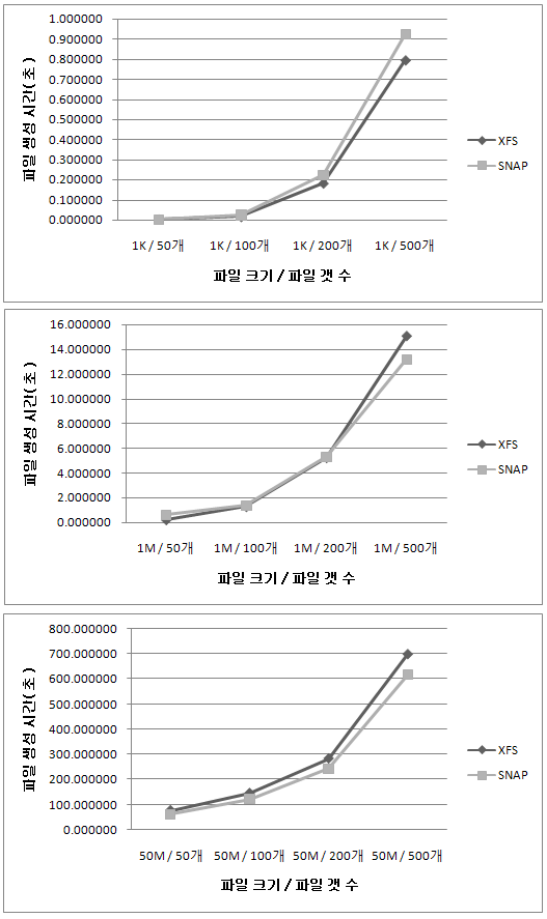

표1. create 성능 비교

표[1]은 파일을 생성한 경우의 성능을 비교한 것으로 단순히 정해진 크기와 갯수로 파일을 생성하는데 걸린 시간을 측정한 것이다. snap+XFS는 XFS에 스냅샷 기능만을 추가한 것이기 때문에 파일을 생성하는 동작에서는 XFS와 거의 동일하게 동작한다. 따라서 표[1]과 같은 서로 유사한 성능을 보이게 된다. 표[2]는 vi와 유사한 방법으로 생성되어 있는 파일을 원래 파일 사이즈만큼의 데이터로 덮어쓰는 경우의 성능을 측정한 것으로 XFS는 단순히 덮어쓰는데 걸리는 시간을 정한 것이고 snap+XFS은 생성되어 있는 파일에 대한 스냅샷을 찍고 원본 파일을 덮어쓰는 경우의 속력을 비교한 것이다. 우리가 vi와 유사한 방법을 사용하여 성능을 측정한 이유는 사용자가 파일을 생성하거나 수정하는 경우에 사용하는 일반적인 유틸이 vi이기 때문이다. vi로 파일을 수정하는 경우, 다음과 같은 절차로 수행된다. 먼저 수정하고자 하는 파일의 복사본인 .swap파일을 생성하고 실질적인 작업은 모두 .swap 파일에서 수행한다. .swap 파일에서의 수정이 모두 끝나면 vi는 원본 파일을 O_TRUNC로 열고 .swap의 데이터를 복사하여 기록한다. 마지막으로 .swap파일이 삭제하고 파일 수정 작업을 완료한다. 우리는 이러한 vi의 동작을 고려하여 성능을 측정하기 위해 스냅샷 파일이 찍힌 파일을 수정할 때 O_TRUNC 옵션으로 파일을 오픈하여 수정된 내용을 기록하였다. snap+XFS는 원본 파일이 vi에 의해서 O_TRUNC로 열리면서 .swap의 데이터를 복사하는 부분의 코드를 수정하여 원본 파일이 스냅샷 파일을 가지고 있는 경우에 데이터 블록에 대한 링크만을 끊고 실제 블록은 삭제하지 않도록 수정했다.

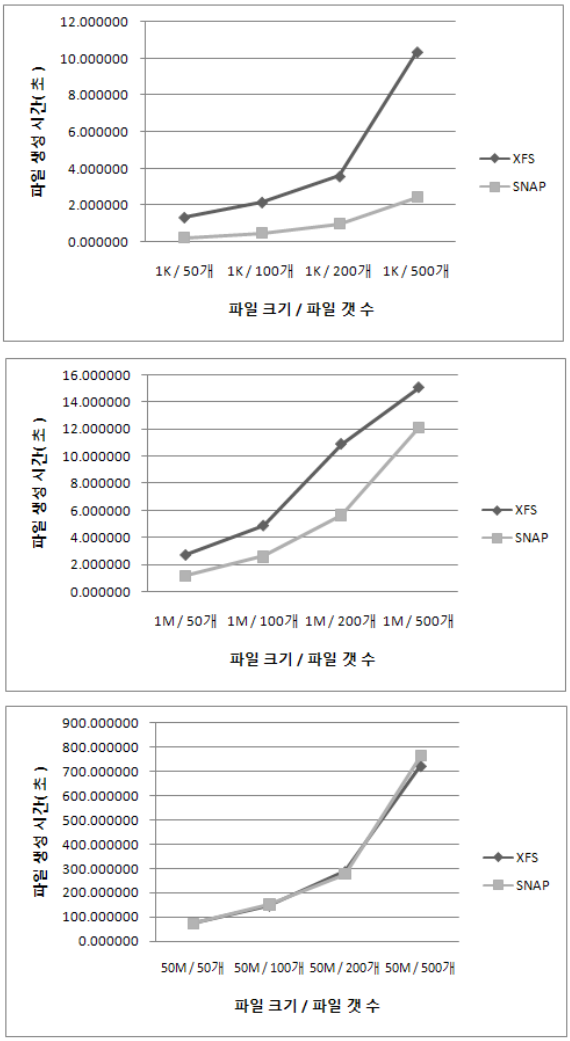

표2. 파일의 변경

그 결과 표[2]의 1K 파일과 1M 파일의 생성 결과처럼 파일의 사이즈가 일정 수준 이하인 경우에는 블록을 삭제하는데 걸리는 시간이 블록을 새로 할당하는 시간보

다 많이 걸려서 snap+XFS의 성능이 XFS보다 우수한 것으로 나타났지만 파일 사이즈가 50M인 경우처럼 파일 사이즈가 커지는 경우, 블록을 삭제하는데 걸리는 시간보다 블록을 새로 할당하는데 걸리는 시간이 더 오래 걸려서 XFS의 성능이 snap+XFS의 성능보다 우수하게 나타났다. 이러한 파일이 일정 수준 이상 커지는 경우에 나타나는 snap+XFS의 성능 문제는 파일 단위로 수행하는 스냅샷의 문제점으로 블록 단위로 스냅샷을 수행한다면 해결될 것이라고 생각한다.

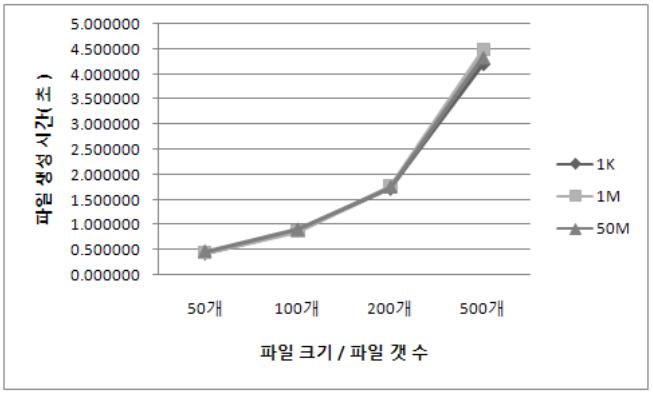

표3. 스냅샷 생성

표[3]은 스냅샷 파일을 생성하는데 걸리는 시간을 측정한 것으로 파일의 크기와 시간은 무관함을 알 수 있다. 다. snap+XFS에서 스냅샷 파일을 생성하는 과정은 원본 파일의 아이노드 정보를 복사하여 스냅샷 파일의 아이노드를 생성하는 작업이기 때문에 파일 크기와는 무관하고 아이노드의 갯수, 즉 생성해야 하는 스냅샷 파일의 갯수에 영향을 받는다.

7. 결론

snap+XFS은 기존의 XFS에 파일 단위로 수행되는 스냅샷 기능을 추가한 것으로 XFS 파일 시스템의 신뢰성을 향상 시키기 위해 제안되었다. 하지만 파일 단위로 수행되기 때문에 파일의 일부만을 수정하는 경우에도 파일의 모든 데이터를 다시 기록하여 디스크 공간을 낭비하는 단점이 있고 파일의 사이즈가 일정 수준 이상으로 커지는 경우 XFS보다 떨어지는 성능을 보였다. 앞으로 파일 단위로 이루어지는 스냅샷의 단점들을 극복하기 위해 블록 단위 스냅샷을 구현 할 계획이다. 블록 단위로 이루어지는 스냅샷은 수정된 블록의 데이터만을 기록하기 때문에 파일단위로 이루어지는 스냅샷 보다 훨씬 향상된 성능을 보여줄 것으로 기대된다.

참 고 문 헌

1. Adam Sweeney, Doug Doucette, Wei Hu, Curtis Anderson, Mike Nishimoto and Geoff Peck,“Scalability in the XFS File System” : USENIX 1996 Annual Technical Conference, 1996.

2. “http://o2000.uibk.ac.at”

3. “http://www-curri.u-strasbg.fr”

4. Zachary N.J.Peterson and RandalC.Burns “Ext3cow : The DesignImplementationAnalysisofMetadata“ : Technical Report HSSL 2003

5. Zachary N.J.Peterson and Randal Burns ``Ext3cow_ATimeShiftingFile''

6. Seungjun Shim, Woojoong Lee and Chanik Park "An Efficient SnapshotTechnique for Ext3 File System in Linux 2.6"

7. Americal Megatrends,"AMI Snapshot Thechnology" : Whitepaper 2005

8. SnapFS: “Http://SOURCEFROG.NET\\/PROJECTS/SNAPFS”