# SSD 캐시를 이용한 분산파일시스템의 성능 향상

김재열, 박정숙, 김영창, 김영균
한국전자통신연구원
e-mail:gauri@etri.re.kr

# Enhancing Distributed File System Performance Using SSD Cache

Chei-Yol Kim, Jeong-Sook Park, Young-Chang Kim, Young-Kyun Kim
Electronics and Telecommunications Research Institute

요 약

분산 파일시스템의 클라이언트 측에 SSD 장치를 캐시장치로 사용하여 분산파일시스템에 부족한 랜덤 입출력 성능을 향상시키고, Back-end 데이터 서버의 부하를 줄일 수 있다. 본 논문은 국내에서 개발된 분산파일시스템인 MAHA-FS의 클라이언트 측에 읽기 캐시로 SSD 장치를 지원함으로써 캐시 히트시에 읽기 성능을 향상 시킬 수 있음과 더불어 읽기 캐시의 기능 추가로 인한 쓰기 성능의 저하가 없음을 보여준다. 본 논문에서 제안한 SSD 캐시를 이용하여 분산파일시스템의 활용 분야을 넓힐 수 있을 것으로 기대한다.

## 1. 서론

폭발적으로 증가하는 디지털 데이터 서비스는 데이터 저장을 위해 대용량의 저장장치를 요구한다. 이를 위해 고비용의 전용 스토리지 장치가 아닌 저사양의 서버를 활용하여 대용량의 스토리지 공간을 제공할 수 있는 분산파일시스템이 다양한 분야에서 사용되고 있다. 구글 파일시스템[1]은 가장 널리 알려진 분산파일시스템이다. 또한 빅데이터 분석에 사용되는 하둡 파일시스템[2]과 슈퍼 컴퓨팅에 주로 사용되는 러스터 파일시스템[3] 모두 구글 파일시스템과 같은 분산파일시스템을 기반으로 하고 있다.

이러한 분산파일시스템은 스토리지 전용의 네트워크와 프로토콜을 사용하지 않고 일반적인 네트워크인 이더넷을 기반으로 하기 때문에 입출력 처리에 있어 지연시간이 많이 걸리는 단점이 있는 반면에 이를 높은 처리율(throughput)을 제공하는 장점으로 극복한다.

분산파일시스템을 이용하면 일반적인 클라우드 환경에서의 순차적 읽기/쓰기 요청은 높은 처리율만으로도 충분한 서비스를 제공할 수 있으나 짧은 지연시간을 요구하는 랜덤 요청이 많은 응용이나 서비스에는 적절히 대응하기 어렵다. 짧은 지연시간 요구를 만족시키기 위한 방법으로 클라이언트 측에 다양한 형태의 캐시를 지원하는 방법이 있다.

SSD는 디스크 기반의 HDD가 가지는 성능상의 문제를 플래쉬 메모리를 사용함으로써 지연시간과 처리율을 모두 향상시켜 최근 HDD를 대체하고 있는 저장장치이다. 하지만 HDD에 비해 상대적으로 적은 용량과 높은 가격으로 인하여 HDD를 완전히 대체하지는 못하고 있다. 최근의 많은 연구들이 이러한 SSD를 캐시로 사용하여 HDD의 저성능을 극복하기 위한 분야에 관해 진행되고 있다.

본 논문은 국내에서 개발된 대표적인 분산파일시스템인 MAHA-FS[4]의 클라이언트 측에 SSD 기반 읽기 캐시 기능을 지원하여 분산파일시스템의 성능을 향상 시킬 수 있음을 보여준다. MAHA-FS[4]는 Glory-FS[5]의 메타데이터 연산처리 성능과 랜덤 입출력 성능을 향상시킨 파일시스템이다.

## 2. 관련연구

SSD를 로컬의 캐시 장치로 사용하기 위해 리눅스를 기반으로 하는 다수의 연구와 결과물이 있다.

CacheFS[6]는 NFS처럼 네트워크 파일 시스템에서 로컬의 블록 장치를 캐시로 사용할 수 있도록 지원한다. CacheFS[6]는 리눅스와 같은 유닉스 기반의 운영체제에서 사용가능한다.

DM-Cache[7]는 최신의 리눅스 커널에서 지원하는 Device Mapper target으로 로컬의 메모리와 블록 디바이스를 이용해 캐시를 구성할 수 있도록 해준다. DM-Cache[7]는 캐시 데이터와 메타데이터를 각각 서로 다른 디바이스에 구성할 수 있도록 지원하는 것이 특징이다. Bcache[8]는 리눅스에서 SSD를 HDD의 캐시장치로 사용할 수 있도록 지원해 주는 커널 모듈로 리눅스 커널 3.1부터 지원된다. Bcache[8]을 사용하기 위해서는 Backing 장치인 HDD를 초기화 해서 등록해야 하기 때문에 기존의 데이터를 유지한 상태로 캐시를 추가할 수 없는 단점이 있다.

Flashcache[9]는 2010년에 Facebook에 의해서 개발된 디

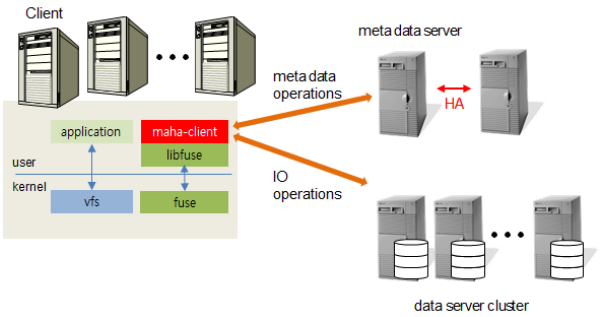
스크 캐시로, DM-Cache[7]과 같이 리눅스의 Deivce Mapper 기반으로 만들어 졌다. FlashCache[9]는 주로 Random IO의 성능을 높이기 위한 것이 주목적 이었으며 따라서 순차 IO의 성능이 떨어지는 단점이 있기도 하다.

## 2. MAHA-FS

MAHA-FS[4]는 [그림 1]과 같이 클라이언트, 메타데이터 서버, 데이터 서버로 이루어진 분산파일시스템이다. 클라이언트는 MAHA-FS[4]를 마운트하여 로컬 파일시스템처럼 사용할 수 있다. MAHA-FS[4]는 POSIX 호환 API를 지원함으로 기존의 응용 프로그램을 컴파일이나 수정 없이 그대로 사용할 수 있다. MAHA-FS[4]의 클라이언트는 리눅스의 FUSE[10]를 이용함으로써 리눅스 커널의 의존성을 탈피하여 다양한 리눅스 배포판에서 사용할 수 있다. 메타데이터 서버와 데이터 서버는 단일 혹은 다수대로 구성할 수 있다. 메타데이터 서버는 파일시스템의 모든 메타데이터를 관리하며 데이터 서버는 파일의 데이터를 청크 단위로 관리하여 클라이언트의 파일 입출력을 처리한다. 데이터 서버의 개수를 증가시킬수록 파일시스템의 용량과 성능을 향상시킬 수 있다.

MAHA-FS[4]의 특징은 아래와 같이 정리될 수 있다.

- 페타바이트급 이상의 스토리지 공간 제공
- 고속의 단일 메타데이터 서버 연산 성능
- DBMS를 탈피한 메타데이터 저장 엔진
- 향상된 랜덤 입출력 성능
- POSIX 표준 API 호환

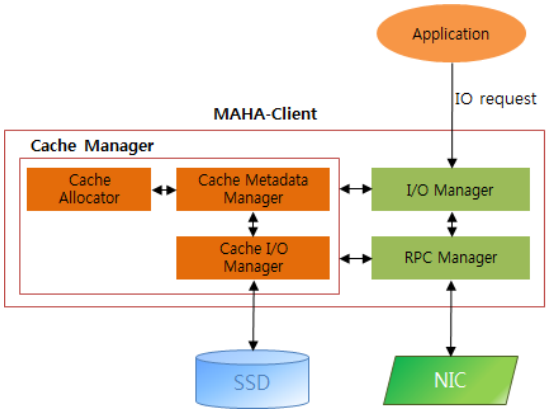



[그림 1]. MAHA-FS 구조

## 2. SSD 캐시

MAHA-FS[4]는 파일 입출력을 하기 위해서 네트워크를 통해 메타데이터 서버와 메타데이터 정보를 교환한 후 데이터 서버와 파일 데이터를 주고 받는 과정을 거쳐야 한다. 이러한 네트워크 기반 파일시스템의 경우 순차 입출력 성능은 상당히 우수하나 랜덤 입출력 성능은 네트워크의 지연시간으로 인해 성능이 떨어질 수밖에 없다. 이는 네트워크의 대역폭은 큰 폭으로 향상되고 있는 반면 네트워크 지연시간은 크게 줄어들지 않고 있기 때문에 발생한다고도 할 수 있다. 이러한 네트워크 통신으로 인한 지연시간을 줄일 수 있는 방법 중의 하나는 자주 사용되는 데이터를 클라이언트에 저장해 두는 것이다. 리눅스를 포함한 대다수의 시스템은 메모리를 버퍼 캐시로 사용하여 메모리 수준에서 파일시스템의 캐시를 지원하고 있다. 하지만 시스템의 메모리는 디스크와 비교하여 상대적으로 용량이 작아 많은 양의 파일 데이터를 보관할 수 없다는 단점이 있다.

본 논문에서는 네트워크를 기반으로 하는 분산 파일시스템의 성능 문제점을 해결하는 방법으로 클라이언트측에 SSD 디스크를 캐시로 사용하는 방법을 제안한다. [그림 1]과 같이 MAHA-FS[4]의 클라이언트에서 실제 입출력을 수행하는 부분은 MAHA-CLIENT로 이는 사용자 레벨에서 동작한다. SSD 캐시는 입출력 요청 중 읽기 요청에 대해서만 캐싱을 지원하며 쓰기 요청은 Write-through 모드로 데이터 서버에서 바로 처리한다. 이는 클라이언트 시스템의 오류가 발생할 때의 데이터 손실 방지와 다수의 클라이언트가 하나의 파일을 공유할 때의 데이터 일관성을 보장해 주기 위해서이다.




[그림 2]. SSD 캐시 구조

[그림 2]는 MAHA-Client에 구현된 캐시의 구조를 보여준다. MAHA-FS[4]의 클라이언트 부분은 [그림 2]의 IO Manager와 RPC Manager로 간략히 표현하였다. Application의 입출력 요청은 IO Manager에서 받아서 요청의 종류를 판별한다. 이 때 캐시와 관련된 요청은 읽기와 쓰기 요청이다. 읽기 요청은 Cache Manager로 보내져서 처리된다. 쓰기 요청의 경우 Cache Manager에게 해당 데이터가 캐시 되어 있다면 이를 무효화 시키기를 요청한다. Cache Metadata Manager는 캐시된 블록의 정보를 관리하며, Cache Allocator는 SSD 디스크 블록의 사용을 관리한다. Cache Allocation 알고리즘으로는 외부 단편화를 줄일 수 있는 버디 알고리즘을 적용하였다. Cache IO manager는 RPC Manager를 통해 데이터서버에서 읽어온

데이터를 SSD에 저장하거나 SSD에 저장된 캐시 데이터를 읽어 주는 역할을 한다. 데이터를 SSD에 저장할 때는 지연쓰기 기법을 이용하여 입출력 응답성능을 높였다.

SSD 캐시 관리자의 특징을 정리하면 아래와 같다.

- Write-through mode
- Buddy allocation 적용으로 외부단편화 최소화
- 캐시 데이터 지연 쓰기로 응답성능 향상

## 2. 실험

본 장에서는 SSD 캐시를 적용한 MAHA-FS[4]의 성능 시험 환경과 시험 결과를 기술한다.

### (1) 실험 환경

본 실험을 통하면 MAHA-FS[4]에 SSD 캐시가 적용되었을 때의 읽기 요청의 성능 향상과 쓰기 시의 성능저하 발생여부를 확인할 수 있다. 기본적인 MAHA-FS[4]의 성능을 비교하기 위하여 네트워크 파일시스템인 NFS를 비교대상으로 삼았다. NFS는 분산파일시스템이 아니기 때문에 동일한 환경의 비교를 위해서 MAHA-FS[4]도 메타데이터 서비스와 데이터서비스를 동일한 서버 한 대에서 실행하였다. 실험에 사용된 HW구성은 아래 [표 1]과 같다.

[표 1]. 하드웨어 실험 환경

| | HW | Specifications |
|---|---|---|
| Client | CPU | Intel Xeon Quadcore 2.67Ghz * 2 socket |
| | RAM | 64GB |
| | SSD | Samsung 840Pro 256GB |
| | OS | CentOS-6.3 |
| Network | | 1Gbps ethernet |
| Server | CPU | Intel Xeon Quadcore 2.67Ghz * 2 socket |
| | RAM | 64GB |
| | Disk | Seagate Momentus 750GB * 4 |
| | OS | CentOS-6.3 |

성능측정 도구로는 입출력 성능 측정에 일반적으로 사용되는 IOzone[11]을 사용하였다. 성능 측정시 사용된 IOzone 옵션은 아래 [표 2]와 같다.

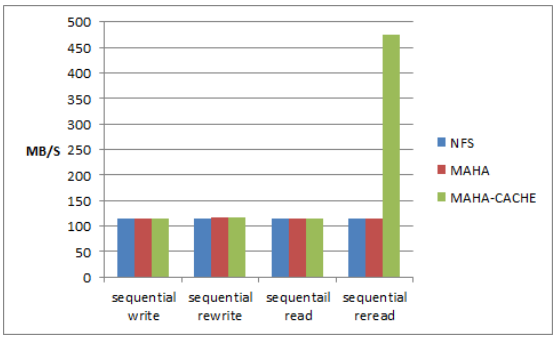
[표 2]. IOzone 옵션

| Test | Type | Value |
|---|---|---|
| Sequential I/O test | Record size | 64KB |
| | Direct IO | enabled |
| | Test file size / thread | 500MB |
| | Test thread | 10개 |
| Random I/O test | Record size | 4KB |
| | Direct IO | enabled |
| | Test file size / thread | 500MB |
| | Test thread | 10개 |

위 IOzone 옵션에서 Direct IO를 선택한 이유는 클라이언트의 메모리 캐시 효과를 제거하기 위해서이다. Direct IO 옵션을 활성화 하지 않으면 각 테스트의 성능은 클라이언트의 메모리 버퍼 효과로 인해 파일시스템의 성능만을 제대로 측정할 수 없다.

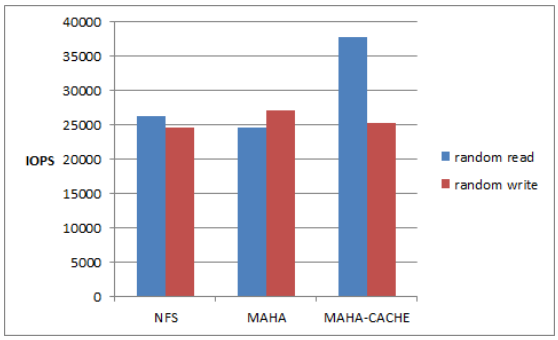
### (1) 실험 결과

실험은 10개의 쓰레드가 각각 500MB의 파일에 대하여 순차 Write, 순차 Rewrite, 순차 Read, 순차 Reread를 수행 후 다시 해당 파일에 대해서 랜덤 Read, 랜덤 Write를 수행하는 것으로 구성하였다.




[그림 3]. Sequential IO 성능

[그림 3]의 결과는 NFS와 MAHA-FS[4]의 순차 읽기, 쓰기 성능이 비슷한 것을 보여준다. MAHA-CACHE의 reread 성능이 월등히 높은 것은 이전의 read에서 읽어온 데이터가 모두 SSD에 캐싱되어 있어 모든 데이터가 캐시 히트 되었기 때문이다. 즉 MAHA-CACHE의 월등한 성능은 SSD 디스크의 성능에 따라 결정된다. 그리고 [그림 3]에서 확인할 수 있는 것은 SSD 캐시가 write 시나 최초 read시의 성능을 떨어뜨리지 않는다는 점이다. 이는 데이터를 캐시 디스크에 넣은 작업이 IO 성능에 큰 영향을 주지 않고 효율적으로 동작하고 있음을 의미한다.




[그림 4]. Random I/O 성능

[그림 4]는 [그림 3]의 sequential 테스트가 끝난 후 바로 수행한 랜덤 테스트 결과이다. [그림 4]의 결과에서도 NFS와 MAHA-FS[4]의 성능은 크게 차이나지 않으며, MAHA-CACHE의 읽기 성능만 13000 IOPS 정도가 높은

것을 알 수 있다. 이는 순차 Reread에 비해서는 상대적으로 떨어진다. 이는 SSD도 순차 IO에 비해 랜덤 IO 성능이 떨어지기 때문으로 판단된다.

## 2. 결론 및 향후 연구

최근의 클라우드 서비스의 확대는 저비용의 대용량 스토리지의 필요성을 높이고 있다. 분산파일시스템은 고가의 스토리지를 쓰지 않고도 대용량의 데이터를 저비용으로 지원할 수 있으며, 전용스토리지에 비해 우수한 확장성을 가지고 있는 장점이 있다. 클라우드 서비스가 다양화 됨에 따라 기존의 순차 IO의 성능뿐만 아니라 랜덤 IO의 성능이 필요한 분야까지 응용이 확대되고 있다.

본 논문에서 제안하는 분산파일시스템의 클라이언트 SSD 캐시는 분산파일시스템의 랜덤 IO의 성능상의 단점을 매워줄 수 있을 것으로 생각된다. [표 3]은 각 파일시스템과 SSD 캐시의 장단점을 보여준다. FS-CACHE[6]는 NFS에서 클라이언트의 SSD 캐시를 지원한다. [표 3]에서 볼 수 있듯이 각 파일시스템과 SSD 캐시는 서로 다른 특징을 가지고 있음을 알 수 있으며, MAHA-FS[4]의 SSD 캐시 지원은 분산파일시스템에서 SSD 캐시를 지원한다는 차별점을 가진다고 볼 수 있다.

[표 3]. 파일시스템 및 SSD CACHE 별 특징

| | NFS | MAHA | MAHA-CACHE | DM-CACHE |
|---|---|---|---|---|
| File 시스템 지원 | O | O | O | X |
| 블록 디바이스 지원 | X | X | X | O |
| 네트워크 파일시스템 지원 | O | O | O | X |
| 분산 파일시스템 지원 | X | O | O | X |
| SSD Cache 지원 | O (FS-Cache) | X | O | O |

본 논문에서 제안한 SSD 캐시는 캐시의 메타데이터를 메모리에 저장함으로써 시스템이 재부팅되거나 파일시스템을 다시 마운트 하는 경우 캐시를 재사용할 수 없는 단점이 있다. 이러한 캐시의 재사용성을 높이기 위해서는 메타데이터도 디스크에 저장하는 기능이 필요하다. 향후 연구에서는 이와 같은 기능을 도입하여 캐시의 재사용성을 높이는 기능의 추가가 필요할 것으로 보인다.

## ACKNOWLEDGEMENT

본 연구는 지식경제부 및 한국산업기술평가관리원의 IT 산업융합원천기술개발사업 “[10041730] 10,000 사용자 이상 동시 접속 가상 데스크톱 서비스를 지원하는 클라우드 스토리지용 파일 시스템 개발” 의 일환으로 수행하였습니다.

## 참고문헌

[1] Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[2] Shvachko, Konstantin, et al. "The hadoop distributed file system." Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010.

[3] Schwan, Philip. "Lustre: Building a file system for 1000-node clusters." Proceedings of the 2003 Linux Symposium. Vol. 2003. 2003.

[4] 김영창, 김동오, 김홍연, 김영균, 최완. “MAHA-FS : 고성능 메타데이터 처리 및 랜덤 입출력을 위한 분산 파일 시스템.” 정보처리학회 논문지, 소프트웨어 및 데이터 공학 2(2), pp.91-96, 2013

[5] Y.S. Min, H.Y. Kim, Y.K. Kim, "Distributed File System for Cloud Computing." Communications of the Korean Institute of Information Scientists and Engineers, Vol.27, No.5, pp.86-94, 2009.

[6] Howells, David. "Fs-cache: A network filesystem caching facility." Proceedings of the Linux Symposium. Vol. 1. 2006.

[7] Van Hensbergen, Eric, and Ming Zhao. "Dynamic policy disk caching for storage networking." URL: http://visa. cs. fiu. edu/ming/dmcache (2006).

[8] http://en.wikipedia.org/wiki/Bcache

[9] http://en.wikipedia.org/wiki/Flashcache

[10] FUSE, http://fuse.sourceforge.net

[11] http://www.iozone.org