ISSN(Print): 2287-5905

http://dx.doi.org/10.3745/KTSDE.2013.2.2.091

MAHA-FS : A Distributed File System for High Performance Metadata Processing and Random IO

Young Chang Kim[†] · Dong Oh Kim[†] · Hong Yeon Kim[††] · Young Kyun Kim[†††] · Wan Choi[††††]

ABSTRACT

The application field of supercomputing systems are changing to support into the field for both a large-volume data processing and high-performance computing at the same time such as bio-applications. These applications require high-performance distributed file system for storage management and efficient high-speed processing of large amounts of data that occurs. In this paper, we introduce MAHA-FS for supercomputing systems for processing large amounts of data and high-performance computing, providing excellent metadata operation performance and IO performance. It is shown through performance analysis that MAHA-FS provides excellent performance in terms of the metadata processing and random IO processing.

Keywords : Supercomputing, Distributed File System

MAHA-FS : 고성능 메타데이터 처리 및 랜덤 입출력을 위한 분산 파일 시스템

김 영 창[†] · 김 동 오[†] · 김 홍 연[††] · 김 영 균[†††] · 최 완[††††]

요 약

바이오 응용과 같은 대용량 데이터 처리와 고성능 계산을 동시에 지원하는 발판으로 슈퍼컴퓨팅 시스템의 활용 분야가 변화하고 있다. 이러한 응용에서는 발생하는 대용량의 데이터를 고속으로 처리하고 효율적으로 저장 관리하기 위한 고성능의 분산 파일 시스템이 요구된다. 본 논문에서는 대용량의 데이터 처리와 고성능 계산을 동시에 지원하는 슈퍼컴퓨팅 시스템을 위해 우수한 메타데이터 연산 성능 및 입출력 성능을 제공하는 MAHA-FS를 소개한다. 아울러 성능 분석을 통해 MAHA-FS가 메타데이터 연산 처리 성능 및 random 입출력 성능이 우수함을 보인다.

키워드 : 슈퍼 컴퓨팅, 분산 파일 시스템

1. 서 론

바이오, 디지털 콘텐츠 렌더링과 같은 고속 처리를 요구하는 미래 선도 산업 및 서비스가 도래하면서 슈퍼컴퓨터의 활용 분야가 복잡한 과학 및 산업 문제 해결을 위한 연구 개발 활동에서 대용량 데이터 처리와 고성능 계산을 동시에 지원하는 시스템으로 활용 분야가 변화하고 있다. 특히 유전체 분석과 같은 응용에서는 분석에 필요한 한 사람의 유전체 데이터 크기가 수 기가바이트에 달하기 때문에, 분석할 유전체 데이터의 수가 수천 건 이상으로 증가하게 되면 입력 데이터의 크기만 수 테라바이트 이상으로 증가하게 된다. 아울러 분석 과정에서 막대한 양의 데이터가 생성되기 때문에 이런 대용량의 데이터를 고속으로 처리 및 효율적인 저장 관리 기능을 제공하고, 스토리지 공간을 무제한으로 확장할 수 있으며 데이터의 안전한 공유를 제공할 수 있는 스토리지 시스템이 요구된다.

이러한 요구를 만족시키고자 다양한 분산 파일 시스템에 대한 연구 개발이 진행되어 왔다. 대표적인 분산 파일 시스템으로는 국외의 경우, Lustre[1], GFS[2], PVFS[3], Ceph[4] 등이 있으며, 국내에서는 GLORY-FS[5]가 개발되어 인터넷 서비스 분야에서 널리 사용되고 있다.

GLORY-FS는 저비용 서버들을 이용하여 저장 공간 구축에 드는 비용을 최소화하면서도 장애에 대한 효율적인 통제

※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업원천기술개발사업의 일환으로 수행하였음[10038768, 유전체 분석용 슈퍼컴퓨팅 시스템 개발].
† 정 회 원 : 한국전자통신연구원 선임연구원
†† 정 회 원 : 한국전자통신연구원 책임연구원
††† 정 회 원 : 한국전자통신연구원 저장시스템연구팀장
†††† 정 회 원 : 한국전자통신연구원 클라우드컴퓨팅연구부장
논문접수 : 2013년 1월 8일
수 정 일 : 1차 2013년 1월 22일
심사완료 : 2013년 1월 22일
* Corresponding Author : Dong Oh Kim(dokim@etri.re.kr)

능력과 높은 입출력 성능을 제공한다. 또한, 응용의 중단없이 온라인으로 공간을 확장할 수 있는 기능을 제공하며, 확장에 따라 선형적으로 입출력 성능이 향상되는 장점을 제공한다. 그러나 GLORY-FS는 높은 sequential 입출력 성능에 반해 메타데이터 연산 처리 성능과 random 입출력 성능이 떨어지는 단점이 존재한다. 본 논문에서는 대용량의 데이터 처리와 고성능 계산을 동시에 지원하는 슈퍼컴퓨팅 시스템을 위해 GLORY-FS의 장점을 유지하면서 단점을 보완한 분산 파일 시스템인 MAHA-FS를 소개한다. MAHA-FS의 설계 목표는 다음과 같다.

- 페타바이트(Petabyte)급 이상의 스토리지 공간 제공
- 고속의 단일 메타데이터 서버 연산 성능 제공
- DBMS를 탈피한 메타데이터 저장 엔진 개발
- 향상된 random 입출력 성능 제공
- 안정성, 확장성, 사용편이성 제공
- POSIX 표준 API 호환 제공

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구인 FUSE와 GLORY-FS를 소개하고, 3장에서는 MAHA-FS를 소개한다. 4장에서는 MAHA-FS의 성능 분석 결과를 기술하고 마지막으로 5장에서 결론을 언급한다.

2. 관련 연구

본 절에서는 관련 연구로서 MAHA-FS 클라이언트에서 사용하는 FUSE(Filesystem in Userspace)와 분산 파일 시스템인 GLORY-FS 에 대해 기술한다.

2.1 Fuse

FUSE[6]는 유닉스 계열의 운영체제에 적재할 수 있는 커널 모듈로써, 커널 레벨이 아닌 사용자 영역에서 실행되는 파일 시스템을 만들 수 있게 해준다. FUSE는 커널 모듈과 라이브러리로 구성되며, Fig. 1에 나타난 바와 같이 VFS(Virtual File System)를 통해 전달되는 파일 시스템 요청을 사용자 영역에서 실행되는 프로세스에서 파일 시스템 코드를 실행할 수 있도록 전달해주는 역할을 수행한다.

FUSE 는 파일 시스템이 사용자 레벨에서 실행되도록 지원하기 때문에 안정성과 개발 편의성 및 POSIX 표준 인터페이스 호환성을 제공하는 장점을 갖는다. FUSE는 Linux, FreeBSD, NetBSD, OpenSolaris, OS X 등을 지원하며, FUSE를 이용한 대표적인 분산 파일 시스템으로는 GLORY-FS[5], GlusterFS[7], Ceph[4] 등이 있다.

2.2 GLORY-FS

GLORY-FS[5]는 국내에서 개발된 분산 파일 시스템으로서 저비용 서버들을 이용하여 저장 공간 구축에 드는 비용을 최소화하면서도 장애에 대한 효율적인 통제 능력과 높은 입출력 성능을 제공한다.

GLORY-FS는 다수의 클라이언트, 메타데이터를 저장하고 관리하는 메타데이터 서버(MDS: Metadata Server) 클러스터, 데이터를 저장하는 데이터 서버(DS: Data Server)들로 구성되며 구조는 Fig. 2와 같다.

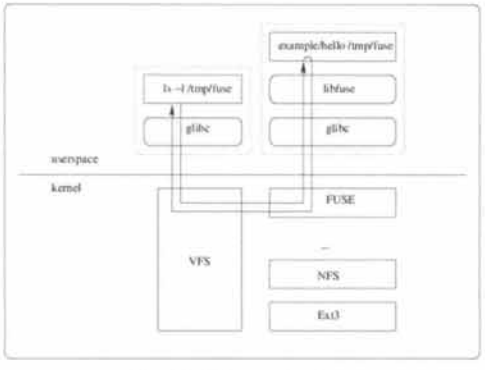

Fig. 1. FUSE_structure

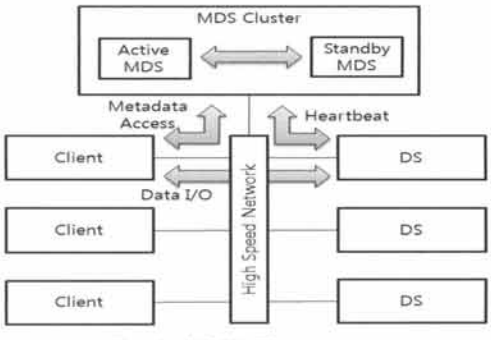

Fig. 2. GLORY-FS_structure

클라이언트는 FUSE를 기반으로 사용자 레벨에서 동작하며 네임스페이스 연산 및 파일 입출력 연산을 수행한다. MDS는 DBMS(Database Management System)를 활용하여 메타데이터를 저장하고 관리하며 Active-Standby 형태의 MDS 클러스터링을 통해 MDS의 장애 발생에 대처하고 시스템의 안정성을 높인다. DS는 사용자의 파일을 일정한 크기의 청크(chunnk)들로 나누어 로컬 시스템의 파일로 저장한다. 이때 데이터 가용성을 위해 일정 개수의 복제본을 생성하여 서로 다른 DS에 분산 저장하고, 각 청크에 대한 버전 관리를 통해 데이터의 무결성을 제공한다.

GLORY-FS는 높은 가용성, 온라인 공간 확장과 확장에 따른 선형적인 입출력 성능 향상, POSIX 호환 API 제공, 네트워크 구조를 인식한 데이터 배치, Hot Spot 회피 기능 등의 장점을 제공한다. 특히 대용량 파일에 대한 sequential 입출력 성능이 매우 우수하다. 그러나 파일의 전체 경로명 기반으로 메타데이터를 관리하여 네임스페이스 연산 처리시 DBMS에서의 문자열 비교에 따른 지연으로 메타데이터 연산 처리 성능이 저조하다. 이를 해결하기 위해 여러 대의 MDS를 클러스터로 구성하여 메타데이터 연산을 분산 처리하지만 MDS를 추가로 구성해야 하기 때문에 비용이 증가하는 문제가 있다. 또한 sequential 입출력에 최적화된 IO 프로토콜로 인해 random 입출력 성능이 떨어지는 단점이 존재한다.

3. MAHA-FS 소개

본 절에서는 GLORY-FS의 단점인 메타데이터 연산 처리 성능 및 random 입출력 성능 한계를 해결하여 슈퍼 컴퓨팅 시스템을 지원하기 위한 MAHA-FS를 소개한다.

3.1 전체적인 구조

MAHA-FS의 전체적인 구조는 Fig. 3과 같다. MAHA- FS는 GLORY-FS와 유사하게 다수의 클라이언트와 메타데이터 서버(MDS) 클러스터, 다수의 데이터 서버(DS)로 구성된다.

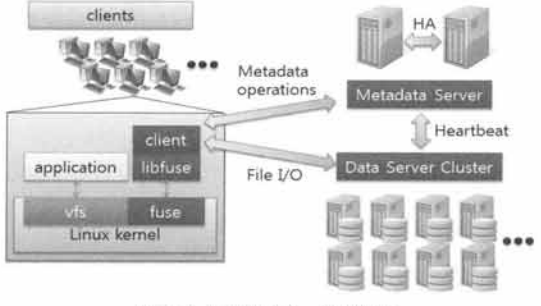

Fig. 3. MAHA-FS_structure

클라이언트는 MAHA-FS를 마운트하여 로컬 파일 시스템처럼 사용할 수 있도록 하며, POSIX 호환 API를 제공함으로써 기존의 응용을 그대로 수용할 수 있는 장점을 갖는다. 메타데이터 서버 클러스터는 분산 파일 시스템에서 필요로 하는 모든 메타데이터를 관리하며, MAHA-FS를 구성하는 각 모듈 및 HW 구성 요소들에 대한 모니터링 및 그 관리 정보를 유지하는 기능을 제공한다. 데이터 서버는 파일의 데이터를 저장 관리하며, 클라이언트에게 파일 입출력 기능을 제공한다.

MAHA-FS의 동작 구조는 Fig. 4와 같다. 먼저, 클라이언트는 사용자의 파일 입출력 요청을 FUSE를 통해 전달받는다. 그리고 전달받은 파일의 inode 정보를 MDS로 전송하여 해당 파일에 대한 레이아웃 정보를 획득한다. 사용자의 파일은 일정 크기의 청크 단위로 분할되어 여러 DS에 분산되어 저장되는데 레이아웃 정보에는 파일이 몇 개의 청크로 이루어져있으며, 또한 데이터 가용성을 위해 복제된 청크들이 몇 개인지, 어느 DS가 해당 청크를 저장하고 있는지에 대한 정보를 포함하고 있다. Read의 경우 클라이언트는 접근할 청크의 복제본 중에서 random으로 선택한 청크에 해당하는 DS와 네트워크 연결을 설정하고 청크에 대한 읽기를 수행한다. 쓰기의 경우에는 복제본 청크 중에서 첫 번째 청크에 해당하는 DS로 네트워크 연결을 설정하고 쓰기를 수행하며, DS는 변경된 청크에 대한 정보를 MDS에 전달하고 해당 청크의 복제본을 갱신하도록 복제본을 저장하고 있는 DS에 변경된 데이터를 전달한다.

3.2 클라이언트

MAHA-FS 클라이언트는 GLORY-FS와 마찬가지로 FUSE를 기반으로 사용자 영역에서 동작하며, 네임스페이스

Fig. 4. MAHA-FS_processing structure

연산 처리, 파일 입출력을 담당하지만 두 가지 측면에서 다른 구조로 동작한다.

첫째, MAHA-FS는 GLORY-FS와 달리 FUSE low level API를 이용하여 동작한다. GLORY-FS의 경우 파일의 전체 경로명을 기반으로 한 FUSE high level API를 사용하여 네임스페이스 연산을 처리하기 때문에, MDS로의 전체 경로명 전달 및 DBMS에서의 경로명 문자열 비교에 따른 비용 증가로 전체적인 메타데이터 연산 처리 성능이 저조한 단점을 갖는다. 이를 해결하기 위해 MAHA-FS는 Fig. 5에 나타난 바와 같이 inode 기반의 FUSE low level API를 이용하여 동작한다.

둘째, MAHA-FS는 request-and-reply의 입출력 프로토콜로 파일 입출력을 처리한다. GLORY-FS는 파일 입출력을 위해 DS와 입출력 채널을 설정하고 read/write 요청을 전달한 후 해당 채널에 대해 read/write를 수행한다. 이때 읽기의 경우 DS는 클라이언트가 요청한 크기에 상관없이 128K 바이트 단위로 데이터를 미리 읽어 네트워크를 통해 클라이언

```
struct fuse_lowlevel_ops {
  void (*init) (void *userdata, struct fuse_conn_info *conn);
  void (*destroy) (void *userdata);
  void (*lookup) (fuse_req_t req, fuse_ino_t parent, const char *name);
  void (*forget) (fuse_req_t req, fuse_ino_t ino, unsigned long nlookup);
  void (*getattr) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*setattr) (fuse_req_t req, fuse_ino_t ino, struct stat *attr, int to_set, struct
                   fuse_file_info *fi);
  void (*readlink) (fuse_req_t req, fuse_ino_t ino);
  void (*mknod) (fuse_req_t req, fuse_ino_t parent, const char *name, mode_t mode,
                 dev_t rdev);
  void (*mkdir) (fuse_req_t req, fuse_ino_t parent, const char *name, mode_t mode);
  void (*unlink) (fuse_req_t req, fuse_ino_t parent, const char *name);
  void (*rmdir) (fuse_req_t req, fuse_ino_t parent, const char *name);
  void (*symlink) (fuse_req_t req, const char *link, fuse_ino_t parent, const char
                   *name);
  void (*rename) (fuse_req_t req, fuse_ino_t parent, const char *name, fuse_ino_t
                  newparent, const char *newname);
  void (*link) (fuse_req_t req, fuse_ino_t ino, fuse_ino_t newparent, const char
                *newname);
  void (*open) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*read) (fuse_req_t req, fuse_ino_t ino, size_t size, off_t off, struct
                fuse_file_info *fi);
  void (*write) (fuse_req_t req, fuse_ino_t ino, const char *buf, size_t size, off_t off,
                 struct fuse_file_info *fi);
  void (*flush) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*release) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*opendir) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*readdir) (fuse_req_t req, fuse_ino_t ino, size_t size, off_t off, struct
                   fuse_file_info *fi);
  void (*releasedir) (fuse_req_t req, fuse_ino_t ino, struct fuse_file_info *fi);
  void (*statfs) (fuse_req_t req, fuse_ino_t ino);
  void (*create) (fuse_req_t req, fuse_ino_t parent, const char *name, mode_t mode,
                  struct fuse_file_info *fi);
};
```

Fig. 5. fuse_lowlevel_ops structure

트에 전달한다. 이러한 구조는 sequential read의 경우 클라이언트가 DS로 read 요청을 전달한 후 DS가 데이터를 전송할 때까지 채널로부터 데이터가 도착할 때까지 대기해야 하는 지연 시간이 필요없기 때문에 sequential 입출력의 성능이 극대화 될 수 있는 장점을 갖는다. 그러나 random read의 경우 필요한 크기보다 큰 데이터를 DS가 전송해야 하기 때문에 네트워크 대역폭이 비효율적으로 활용되고, 해당 채널을 해제하고 다시 채널을 설정하여 random 입출력을 수행해야하기 때문에 성능이 저하되는 단점이 존재한다.

이를 해결하기 위해 MAHA-FS 클라이언트는 각 DS에 대한 소켓 풀을 관리한다. 파일 입출력을 위해 DS와 입출력 채널이 생성되면 입출력이 끝난 후 이를 해제하지 않고 소켓 풀에 반환하고 다음 입출력 요청이 발생하면 이를 재사용한다. 이를 통해 random 입출력 시 채널을 항상 새로 생성하고 해제해야하는 비용을 줄임으로써 입출력 성능을 향상시킨다. 또한, 파일 입출력 요청 및 크기를 DS에 전달하고 DS는 요청된 크기만 클라이언트에 전달하는 request-and-reply 입출력 프로토콜을 사용하여 네트워크 대역폭을 효율적으로 사용한다.

3.3 MDS

MAHA-FS의 MDS는 클라이언트로부터 요청된 네임스페이스 연산을 처리하며, 파일 시스템의 볼륨 정보, DS, 디스크, 클라이언트, MDS 통계 정보 관리, 청크 할당/해제, 자원 관리, 고장 회복 관리등의 기능을 수행한다.

MAHA-FS는 GLORY-FS와 달리 메타데이터 저장을 위해 DBMS를 사용하지 않고 inode 비트맵, inode 데이터, 블록 비트맵, 블록 데이터를 기반으로 메타데이터를 관리하는 자체 메타데이터 관리 엔진을 제공한다. Inode 비트맵 파일은 inode가 할당되었는지 여부에 대한 정보를 나타내며, inode 파일은 4KB 블록 단위로 구성된 inode에 대한 정보를 저장한다. 블록 비트맵은 블록의 할당 유무를 나타내며, 블록 데이터 파일은 4KB 단위로 파일 메타데이터의 청크 레이아웃 정보를 저장하는데 사용한다. MDS는 하나의 메타데이터 연산을 트랜잭션 형태로 처리하여 ACID(Automic, Consistency, Isolation, Durability)를 지원함으로써 DBMS와 같은 수준의 데이터 안정성을 제공한다.

MDS는 모든 DS들로부터 주기적으로 heartbeat을 받아 DS들의 상태를 파악하고 장애가 감지된 DS에 저장된 청크의 복제본을 다른 DS에 새로운 복제본을 생성하여 저장하도록 함으로써 DS 장애에 따른 데이터 가용성을 제공한다. 또한 Active-Standby 구조의 MDS 클러스터를 통해 active 상태의 MDS는 모든 메타데이터와 관련된 연산을 처리하고, standby 상태의 MDS로 지속적으로 메타데이터 동기화를 수행한다. Standby MDS 는 heartbeat을 통해 MDS의 상태를 모니터링하고 active MDS에 장애가 발생했을 때 active 상태로 전환하여 MDS 장애에 대처하고 시스템의 안정성을 높인다.

3.4 DS

DS는 데이터의 무결성 보장과 안전한 보관, 저장 공간의 효율적 관리, 최적 입출력 성능 보장을 담당한다. DS는 클라이언트로부터의 파일 입출력 요청을 받아 처리하고, 서버의 상태나 부하 등을 모니터링하고 그 결과를 주기적으로 heartbeat을 통해 MDS에 전달한다.

사용자의 파일은 일정 크기의 청크 단위로 분할되어 DS에 분산 저장된다. 이때, 데이터의 가용성을 보장하기 위해 청크는 일정한 수의 복제본을 생성하여 서로 다른 DS에 분산되어 저장된다. 또한, 데이터의 무결성을 보장하기 위해 데이터를 기록할 때마다 이를 MDS에 전달하여 해당 청크의 정보가 변경되었음을 알리고 해당 청크의 버전 정보를 갱신하며, 청크의 복제본이 위치한 다른 DS에 변경된 청크 내용을 전달한다. 청크는 잘 구성된 디렉토리의 임의 위치에 청크의 식별자를 이름으로 하여 로컬 파일 시스템 상에 일반 파일로 저장된다.

4. 성능 분석

4.1 시험 환경

성능 분석을 위해 사용한 시스템 환경은 Table 1과 같다. 시험은 Redhat Enterprise Linux 6.1을 설치한 5대의 시스템을 사용하였으며, MDS 1대, DS 4대를 운영하였다. 각 시스템은 Infiniband 40Gbps 네트워크를 통해 연결되어 있으며, IPoIB(IP over InfiniBand)를 이용하여 TCP/IP 환경에서 성능을 분석하였다.

성능 분석은 GLORY-FS와 MAHA-FS와의 성능을 비교하였으며, 메타데이터 연산 성능과 파일 입출력 성능으로 나누어 수행하였다. 메타데이터 연산 성능 분석은 각 파일 시스템을 각 클라이언트에서 마운트하고, 파일 생성을 수행하여 초당 실행된 명령어의 횟수를 비교하였으며, 파일 입출력 성능은 성능 분석 도구인 IOzone[8]을 사용하여 sequential 입출력과 random 입출력 성능을 각각 분석하였다.

Table 1. Experiment environment

experiment environment	model
CPU	Intel Xeon QuadCore 2.4Ghz * 2ea
RAM	24GB
RAID Controller	LSI MegaRAID 9265-8i * 2ea
SSD	OCZ Veretx3 120GB * 16ea
Network	Infiniband 40GB IPoIB
OS	Redhat Enterprise Linux 6.1
Experiment Tool	IOzone

4.2 메타데이터 연산 성능 분석 결과

먼저, 메타데이터 연산 성능 분석 결과에 대해 기술한다. 메타데이터 연산 성능 분석을 위해 MAHA-FS와 GLORY-FS를 마운트한 클라이언트에서 5개의 쓰레드를 생성하고 각 쓰레드마다 100,000개의 파일을 생성하도록 테스트 프로그램을 작성하였다. 이때 클라이언트의 수에 따른 성능 변화를 분석하기 위해 클라이언트의 수는 1, 4, 8대로 증가하며 실험을 수행하고, 파일 생성에 따라 발생하는 네임스페이스 연산 중 getattr, create 명령의 수와 전체 네임스페이스 연산 처리 횟수를 분석하였다. Fig. 6은 메타데이터 연산 성능 분석 결과를 나타낸다.

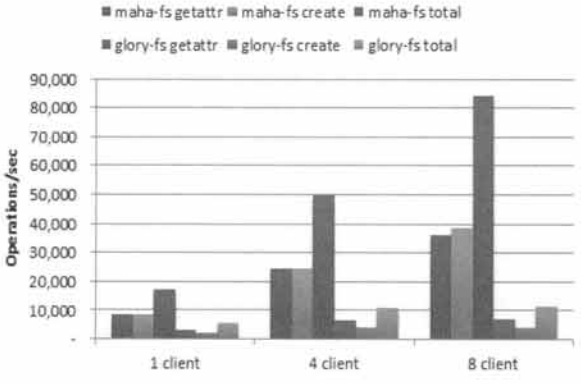

Fig. 6. Metadata operation performance

Fig. 6에 나타난바와 같이 전체적으로 MAHA-FS의 메타데이터 연산 성능이 GLORY-FS에 비해 매우 우수함을 나타내었다. 클라이언트가 1대일 경우에는 초당 처리한 명령의 수가 MAHA-FS는 약 17,000건, GLORY-FS는 약 5,800건으로 약 3배의 차이가 발생하였으며, 이는 클라이언트의 수가 증가할수록 더 벌어졌다. 클라이언트의 수가 8대일 경우에는 MAHA-FS가 약 84,000건, GLORY-FS의 경우 약 12,000건으로 7배의 차이가 발생하였다. 특히 MAHA-FS의 경우 클라이언트의 수가 증가할수록 메타데이터 연산 성능이 선형적으로 증가하는 반면, GLORY-FS의 경우 클라이언트의 수가 4대일 경우와 8대일 경우의 차이가 거의 발생하지 않았다. GLORY-FS의 경우 메타데이터 저장을 위해 DBMS를 사용하고 경로명 기반으로 파일의 메타데이터가 저장되어 있어 파일이 생성될 부모 디렉토리의 위치를 찾기 위한 문자열 비교에 따른 지연과 경로명을 기반으로 클라이언트와 MDS간의 네트워크 통신 지연이 발생하여, 메타데이터 연산 처리가 초당 약 12,000건이 한계로 보임을 나타내었다. MAHA-FS의 경우에는 최대 메타데이터 연산 처리 성능을 확인하기 위해서는 보다 많은 클라이언트를 이용한 실험이 수행되어야 할 필요가 있다.

4.3 Random 입출력 성능 분석 결과

다음으로 random 입출력 성능 분석 결과에 대해 기술한다. random 입출력 성능 분석은 IOzone을 이용하여 실험하였으며, 입출력 단위를 4K 바이트로 고정하고 초당 처리한 입출력 명령 횟수를 측정하였다. Fig. 7은 random 입출력 성능 분석 결과를 나타낸다.

Fig. 7에서 MAHA-FS의 경우 쓰레드의 수가 증가할수록, 그리고 클라이언트의 수가 증가할수록 random 입출력 성능의 향상이 있음을 나타내었다. 반면 GLORY-FS의 경우 read는 약 12,000건, write는 약 1,200건 이하로 비슷한 성능을 나타내었다. 두 파일 시스템 모두 read에 비해 write의 성능이 저조함을 나타내었다. 이는 두 파일 시스템 모두 write에 따른 청크에 대한 변경이 발생할 때마다 청크의 변경을 MDS로 전달하고 다른 복제 청크들과의 동기화를 수행하기 때문에 나타나는 현상이다.

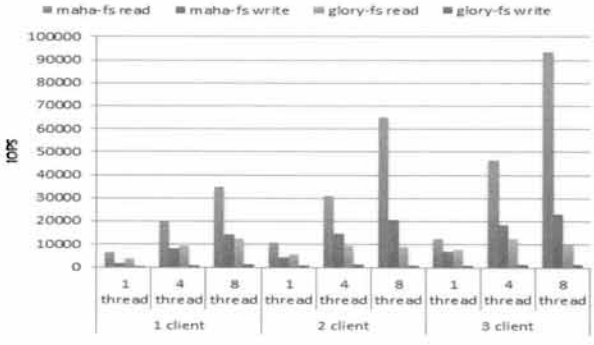

Fig. 7. Random IO performance

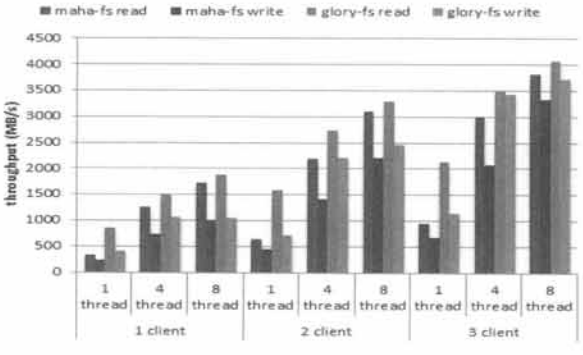

Fig. 8. Sequential IO performance

4.4 Sequential 입출력 성능 분석 결과

마지막으로 파일 입출력 성능 중 sequential 입출력 성능 분석 결과에 대해 기술한다. Sequential 입출력 성능 분석은 random 입출력 성능 분석과 동일한 환경에서 IOzone을 이용하여 실험하였으며, 입출력 단위는 128K 바이트이고, 사용된 파일의 크기는 1G 바이트이다. 실험은 클라이언트의 수를 1, 2, 3대로 증가시키며 측정하였고, 각 클라이언트마다 쓰레드의 수를 1, 4, 8개로 변화시키며 실험을 수행하였다. Fig. 8은 sequential 입출력에 대한 성능 분석 결과를 나타낸다.

Fig. 8에서 두 파일 시스템 모두 클라이언트의 수가 증가할수록, 그리고 쓰레드의 수가 증가할수록 입출력 성능이 선형적으로 증가하였다. 전체적으로 GLORY-FS가 MAHA-FS보다 sequential 입출력 성능이 항상 우수함을 나타내었으며, 쓰레드의 수가 적거나 클라이언트의 수가 적을 때에는 약 2.5~3배 차이가 발생하였다. 그러나 쓰레드의 수가 증가할수록 성능 차이는 줄어드는 경향을 나타내었다. 클라이언트의 수가 3대이고 클라이언트당 쓰레드의 수가 8일 때, 즉, 총 24개의 쓰레드에서 입출력이 발생하였을 때에는, MAHA-FS의 경우 read 가 약 3.8GByte/sec, write 가 약 3.3GByte/sec 이고, GLORY-FS의 경우 read 가 약 4.0GByte/sec, write 가 약 3.7GByte/sec를 측정되어 MAHA-FS가 GLORY-FS에 비해 read, write 성능이 각각 95%, 90%정도로 차이가 줄어드는 것을 보였다. MAHA-FS의 입출력 프로토콜은 random 입출력 성능은 크게 향상시켰으나, sequential 입출력에서 성능차이가 발생하였으며, 이를 반영한 입출력 프로토콜의 개선이 필요함을 나타내었다.

5. 결 론

본 논문에서는 대용량의 데이터 처리와 고성능 계산을 동시에 지원하는 슈퍼컴퓨팅 시스템을 위한 분산 파일 시스템인 MAHA-FS를 소개하고 그 성능을 분석하였다. MAHA-FS는 inode 기반의 자체 메타데이터 관리 엔진과 FUSE low-level API를 이용하여 우수한 메타데이터 연산 성능을 제공하며, request-and-reply IO 프로토콜을 이용하여 개선된 random 입출력 성능을 제공한다. 그러나 GLORY-FS에 비해 sequential 입출력의 성능이 약 90~95% 정도로 저조한 성능을 나타내었다.

향후 연구로는 sequential 입출력을 고려하도록 MAHA-FS의 입출력 프로토콜을 개선하여 sequential 입출력 성능을 향상시키는 연구가 필요하며, 실제 바이오 응용에 MAHA-FS를 적용하여 실험을 수행하고 MAHA-FS가 슈퍼컴퓨팅 시스템을 위한 분산 파일 시스템으로서 적용 가능함을 입증하는 것이다.

참 고 문 헌

[1] Lustre File System [Internet], http://www.lustre.org

[2] S. Ghemawat, H. Gobioff, S. Leung. "The Google file system," In Proc. of ACM Symposium on Operating Systems Principles, Lake George, NY, pp.29-43, 2003.

[3] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. "PVFS: A parallel file system for Linux clusters," in Proc. of 4th Annual Linux Showcase and Conference, pp.317-327, 2000.

[4] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI '06), pp.307-320, 2006.

[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] FUSE: Filesystem in Userspace [Internet], http://fuse.sourceforge.net

[7] GlusterFS: Clustered File Storage that can scale to petabytes [Internet], http://www.glusterfs.org/.

[8] IOzone Filesystem Benchmark [Internet], http://www.iozone.org

김 영 창

e-mail : zerowin@etri.re.kr

2001년 전북대학교 컴퓨터공학(학사)

2003년 전북대학교 컴퓨터공학과(석사)

2009년 전북대학교 컴퓨터공학과(박사)

2009년~현 재 한국전자통신연구원 선임연구원

관심분야 : Database & File System

김 동 오

e-mail : dokim@etri.re.kr

2000년 건국대학교 컴퓨터공학(학사)

2002년 건국대학교 컴퓨터·정보통신공학과(석사)

2006년 건국대학교 컴퓨터·정보통신공학과(박사)

2006년~년2009년 건국대학교 강의교수

2009년~현 재 한국전자통신연구원 선임연구원

관심분야 : 데이타베이스, 병렬 파일 시스템, 공간 데이터 관리

김 홍 연

e-mail : kimhy@etri.re.kr

1992년 인하대학교 통계학과(학사)

1994년 인하대학교 전자계산학과(석사)

1999년 인하대학교 전자계산학과(박사)

1999년~현 재 한국전자통신연구원 책임연구원

관심분야 : 스토리지 시스템, 파일시스템, 데이터베이스 시스템

김 영 균

e-mail : kimyoung@etri.re.kr

1991년 전남대학교 전산통계학과(학사)

1993년 전남대학교 전산통계학과(석사)

1995년 전남대학교 전자계산학과(박사)

1999년~현 재 한국전자통신연구원 저장시스템연구팀장

관심분야 : 스토리지 시스템, 파일시스템, 데이터베이스 시스템

최 완

e-mail : wchoi@etri.re.kr

1981년 경북대학교 전자공학과(학사)

1985년 KAIST 전산학과(석사)

1985년~2003년 ETRI, TDX/CDMA 전전자교환기용 실시간OS/DBMS/MW/컴파일러 개발책임자

2000년~2011년 한국정보처리기술사회 이사

2004년~2007년 ETRI, 클라우드서비스 기술 개발팀장

2007년 불교명예철학 박사

2008년~2010년 ETRI, SW콘텐츠미래기술연구부장

2011년~현 재 한국전자통신연구원 클라우드컴퓨팅연구부장

관심분야 : Cloud Computing, High Performance Computing