클러스터 파일 시스템 기술 동향

Trends of Cluster File Systems Technology

김영철 (Y.C. Kim) 저장시스템연구팀 선임연구원
박근태 (K.T. Park) 저장시스템연구팀 Post-Doc.
이상민 (S.M. Lee) 저장시스템연구팀 연구원
김홍연 (H.Y. Kim) 저장시스템연구팀 선임연구원
김영균 (Y.K. Kim) 저장시스템연구팀 팀장

목 차

최근에 블로그, UCC, IPTV 등 사용자 중심의 인터넷 서비스와 언제 어디서나 웹을 통해 서비스를 받을 수 있는 유비쿼터스 컴퓨팅 환경으로의 변화는 대용량 저장 공간과 빠르고 안정된 서비스를 제공할 수 있는 대규모 클러스터 시스템 플랫폼을 필요로 한다. 그리고 이러한 대규모 클러스터 시스템 플랫폼을 효율적으로 관리하고 이용할 수 있는 클러스터 파일 시스템은 필수적이다. 본 고에서는 최근에 연구·개발되고 있는 클러스터 파일 시스템들을 소개하고 기술 동향에 대해 살펴본다.

Ⅰ. 서론

최근에 웹 2.0의 등장으로 날로 증가하고 있는 블로그, UCC, IPTV 등과 같은 사용자 중심의 인터넷 서비스와 모든 디지털 기기를 이용하여 언제 어디서나 웹을 통해 서비스를 받을 수 있는 유비쿼터스 컴퓨팅 환경은 대규모의 클러스터 시스템 플랫폼을 필요로 한다.

대규모 클러스터 시스템 플랫폼은 네트워크 상에 분산된 대량의 서버들을 클러스터로 구성함으로써 대용량의 저장 공간과 빠른 입출력 성능을 제공할 수 있어야 한다. 또한 시스템 확장이 용이하며, 서버 고장과 같은 시스템 장애가 발생하더라도 계속해서 안전하게 서비스를 제공할 수 있는 신뢰성과 가용성을 보장하여야 한다.

따라서 대규모 클러스터 시스템 플랫폼의 근간이 되는 파일 시스템은 위의 요구 사항을 최대한 만족할 수 있어야 한다. 하지만 NFS와 같은 기존의 단순한 클라이언트-서버 수준의 분산 파일 시스템으로는 시스템 성능과 확장에 한계가 있다.

비대칭형(asymmetric) 클러스터 파일 시스템은 성능과 확장 그리고 가용성 면에서 적합한 분산 파일 시스템 구조로 최근에 연구와 개발이 활발히 진행되고 있다. 비대칭형 클러스터 파일 시스템에서는 파일 메타데이터를 관리하는 전용 서버를 별도로 둠으로써 메타데이터를 접근하는 경로와 데이터를 접근하는 경로를 분리한다. 그리고 이를 통하여 파일 입출력 성능을 높이면서 독립적인 확장과 안전한 파일 서비스를 제공하고자 한다. 하지만 메타데이터 서버에 부하가 집중될 수 있으며 single-of-failure 지점이 될 수 있는 문제점도 내포하고 있다.

객체 기반 클러스터 파일 시스템[1]은 비대칭형 구조를 갖는 대표적인 파일 시스템으로 다른 파일 시스템들과는 달리 파일 데이터를 논리적인 객체 단위로 저장하고 관리한다. 따라서 블록 기반 인터페이스 대신에 객체 처리를 위한 객체 기반 인터페이스를 사용한다.

본 고에서는 최근에 연구·개발되고 있는 클러스터 파일 시스템의 사례 시스템들로 Ceph, GlusterFS, Google 파일 시스템, Hadoop 분산 파일 시스템, Lustre, Panasas, PVFS2, OASIS를 소개하고 이러한 시스템들의 기술적인 동향에 대해 살펴본다.

Ⅱ. 클러스터 파일 시스템

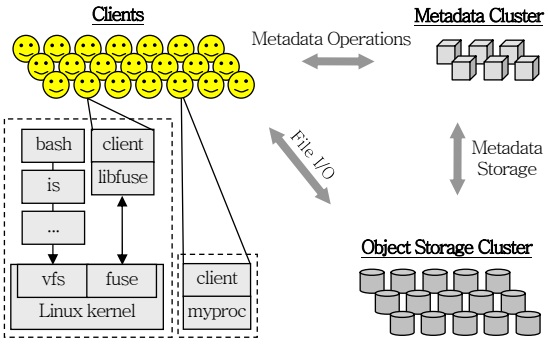
1. Ceph

Ceph[2],[3]는 University of California, Santa Cruz에서 개발하고 있는 프로토타입 수준의 객체 기반 클러스터 파일 시스템이다.

Ceph는 (그림 1)에서와 같이 크게 클라이언트와 메타데이터를 관리하는 메타데이터 서버 클러스터, 그리고 데이터를 객체 수준으로 저장하고 관리하는 객체 저장 서버 클러스터로 구성된다.

Ceph 클라이언트는 FUSE를 이용하여 마운트한 파일 시스템으로 접근할 수 있으며 또한 응용 프로그램에서 직접 링크하여 사용할 수도 있는 파일 시스템 인터페이스를 제공한다. 메타데이터 서버 클러스터는 파일 시스템의 이름 공간뿐만 아니라 보안과 시스템 일관성을 관리한다. 객체 저장 서버 클러스터에는 파일 메타데이터와 데이터가 모두 저장된다. 따라서 객체 저장 서버 클러스터는 클라이언트와 메타데이터 서버 클러스터에게 하나의 논리적인 볼륨으로 보여진다.

Ceph에서는 파일 메타데이터와 데이터의 분리를 강화하기 위하여 대부분의 객체 기반 파일 시스



(그림 1) Ceph 구조

템에서 유지하는 파일의 객체 할당 리스트를 유지하지 않는다. 대신에 전체 파일 시스템에서 유일한 값인 파일의 inode 번호를 이용하여 객체 이름을 정하고 CRUSH[4]라는 방법을 통하여 객체를 객체 저장 서버에 분산하여 저장한다. 이렇게 함으로써 Ceph 구성요소의 어느 위치에서든지 CRUSH를 이용하여 객체의 이름과 위치를 쉽게 계산해서 얻을 수 있도록 한다.

Ceph에서는 메타데이터 서버의 확장성을 고려하여 메타데이터 서버 클러스터링 방법으로 동적 서브트리 분할(dynamic subtree partitioning) 방법[5]을 채용한다. 이를 통해 작업 부하의 패턴에 따라 동적으로 여러 메타데이터 서버에 작업부하를 분산하거나 또는 메타데이터를 재분배함으로써 메타데이터 서버 클러스터에 효율적으로 작업을 분배하도록 한다.

Ceph는 저장공간과 성능의 선형적 증가를 보장하면서 클러스터 확장, 고장 검출 및 회복을 효율적으로 제공하기 위하여 데이터 마이그레이션, 데이터 복제, 고장 검출 및 회복에 대한 작업을 객체 저장 서버 클러스터에서 담당하도록 한다.

객체 저장 서버에서는 대부분의 객체 기반 파일 시스템에서 Ext3와 같은 로컬 파일 시스템을 사용하는 것과는 달리 같은 연구실에서 개발한 EBOFS[6]를 사용한다. EBOFS는 객체 기반 작업 부하의 특성을 반영하여 개발된 사용자 수준의 파일 시스템이다.

2. GlusterFS

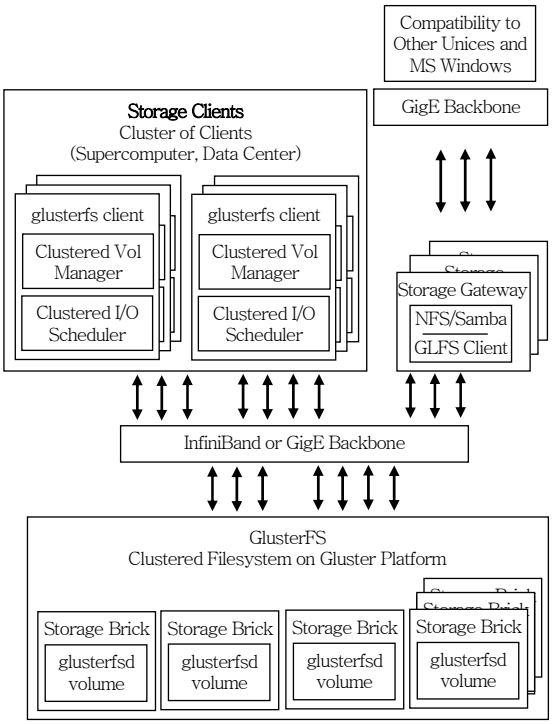
Gluster[7]는 Z RESEARCH Inc.에서 개발되고 있는 클러스터 응용 플랫폼으로 클러스터 파일 시스템인 GlusterFS, 고성능 컴퓨팅 클러스터를 구성할 수 있는 GlusterHPC, 그리고 시스템 프로비저닝과 자동화된 플랫폼 관리를 제공하는 GlusterEP가 포함되어 있다.

GlusterFS는 대용량 저장 공간과 고성능 컴퓨팅을 제공하기 위한 클러스터 파일 시스템으로 수 페타 바이트 수준까지 확장 가능한 저장 공간과 각 서버의 대역폭을 효율적으로 결집할 수 있는 다양한 형태의 서버 클러스터링 방법들을 제공한다. 또한 서버들은 TCP/IP 이외에도 Infiniband RDMA와 같은 고속 네트워크로 연결된다.

GlusterFS는 (그림 2)에서와 같이 서버, 클라이언트, 그리고 전송 모듈과 스케줄러 모듈 등으로 구성된다.

서버는 저장 볼륨을 구성하고 클라이언트로부터의 파일 입출력 요청을 처리한다. 서버는 사용자 수준 데몬 프로세스 형태로 동작한다. 각 서버에서는 로컬 파일 시스템을 이용하여 파일의 메타데이터와 데이터를 저장한다.

클라이언트는 FUSE를 이용하여 구현되었다. 클라이언트에서는 각 서버에서 제공하는 저장 볼륨을 여러 형태로 구성하고 이를 마운트하여 사용할 수 있다. GlusterFS에서 메타데이터는 특정 서버와 저장 볼륨을 지정하거나 또는 클러스터를 구성하는 모든 서버에 분산되어 저장할 수 있다. 전자의 경우에는 이름 공간을 지정된 서버의 저장 볼륨에 기록하



(그림 2) GlusterFS 구조

고 실제 데이터는 다른 볼륨들에 저장한다. 하지만 메타데이터를 저장하는 볼륨에는 이름 공간만 저장되지 실제 데이터가 저장된 볼륨에 대한 정보는 기록되지 않는다. 따라서 실제 데이터가 저장된 볼륨을 찾기 위해서 모든 서버의 저장 볼륨들을 접근해야 한다. 후자의 경우에는 클라이언트에서 구성한 모든 서버들의 저장 볼륨들이 동일한 디렉토리 구조를 가지며 데이터는 동일한 경로 아래에 파일로 저장된다.

GlusterFS는 클라이언트와 서버 간에 선택적인 클러스터 구성과 다양한 기능이 translator 형태로 구현되어 있다. Translator는 공유 라이브러리 형태로 서버 또는 클라이언트의 실행 시에 적재되어 동작한다.

예를 들어, translator에는 서버를 RAID-0, RAID-1, linear 등의 형태로 구성하는 클러스터링 translator, 서버에서 read-ahead, write-behind, 멀티쓰레드 기능을 제공하기 위한 성능 향상 translator, 서버와 클라이언트 간의 네트워크 프로토콜을 구현한 translator 등이 있다. 그밖에 필요한 경우에는 사용자가 자신이 원하는 기능을 갖는 translator를 직접 구현하여 추가할 수 있다.

GlusterFS에서는 클라이언트에서 미리 클러스터를 구성할 서버에 대한 정보를 알고 있어야 하며 클러스터 확장, 서버 고장 등으로 인한 서버 클러스터의 변화에 동적으로 대응하기 어렵다. 또한 파일의 메타데이터와 데이터를 저장하는 볼륨이 분리되어 있음에도 불구하고 파일의 메타데이터를 찾더라도 파일이 존재한다는 사실만 확인할 뿐 실제로 파일 데이터가 어느 서버의 저장 볼륨에 저장되어 있는지는 모른다. 따라서 파일 데이터가 저장된 서버의 저장 볼륨을 찾기 위해 모든 서버를 접근해야 한다.

3. Google 파일 시스템

Google 파일 시스템(GFS)[8]은 구글의 대규모 클러스터 서비스 플랫폼의 기반이 되는 파일 시스템으로 개발되었다.

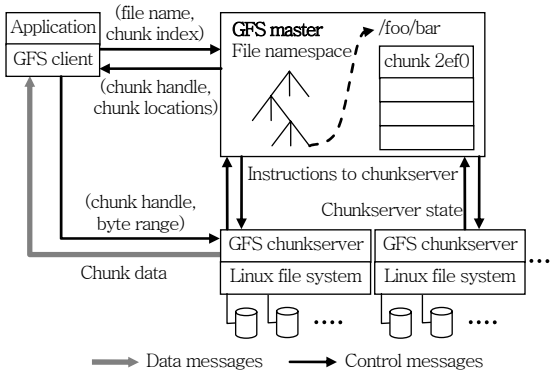
GFS는 다음과 같은 가정을 토대로 설계되었다.

- 저가형 서버로 구성된 환경으로 서버의 고장이 빈번히 발생할 수 있다고 가정한다.
- 대부분의 파일은 대용량 파일을 가정한다. 따라서 대용량 파일을 효과적으로 관리할 수 있는 방법이 요구된다.
- 작업 부하는 주로 연속적으로 많은 데이터를 읽는 연산이거나 또는 임의의 영역에서 적은 데이터를 읽는 연산으로 구성된다.
- 파일에 대한 쓰기 연산은 주로 순차적으로 데이터를 추가하는 연산이며 파일에 대한 갱신은 드물게 이루어진다.
- 여러 클라이언트에서 동시에 동일한 파일에 데이터를 추가하는 환경에서 동기화 오버헤드를 최소화할 수 있는 방법이 요구된다.
- 낮은 응답 지연시간보다 높은 처리율이 보다 중요하다.

GFS는 (그림 3)에서와 같이 클라이언트, 마스터, chunk 서버들로 구성된다.

GFS의 클라이언트는 POSIX 인터페이스를 지원하지 않으며 파일 시스템 인터페이스와 유사한 자체적인 인터페이스를 제공한다. 또한 여러 클라이언트에서 원자적인 데이터 추가(atomic append) 연산을 지원하기 위한 인터페이스를 제공한다.

GFS에서 파일은 고정된 크기의 chunk들로 나누어서 chunk 서버들에 분산되어 저장된다. 그리고 각 chunk에 대한 여러 개의 복제본도 chunk 서버에



(그림 3) Google 파일 시스템 구조

분산되어 저장된다. 따라서 클라이언트는 파일을 접근하기 위하여 먼저 마스터에게 해당 파일의 chunk가 저장된 chunk 서버의 위치와 핸들을 받아 온다. 그리고 나서 직접 chunk 서버로 파일 데이터를 요청한다.

GFS의 마스터는 단일 마스터 구조로 파일 시스템 이름 공간과 파일의 chunk 매핑 정보 그리고 각 chunk가 저장된 chunk 서버들의 위치 정보 등 모든 메타데이터를 메모리 상에서 관리한다. GFS에서는 기본 chunk의 크기를 64MB로 지정함으로써 파일 메타데이터의 크기를 줄이고 또한 기존의 트리 구조가 아닌 해시 테이블 구조 등을 사용함으로써 메모리 상에서 보다 효율적인 메타데이터 처리를 지원하려고 한다. 마스터에서는 주기적으로 하트 비트를 이용하여 chunk 서버에 저장된 chunk들의 상태를 체크하고 상태에 따라 chunk를 재복제하거나 재분산하는 것과 같은 회복 동작을 수행한다.

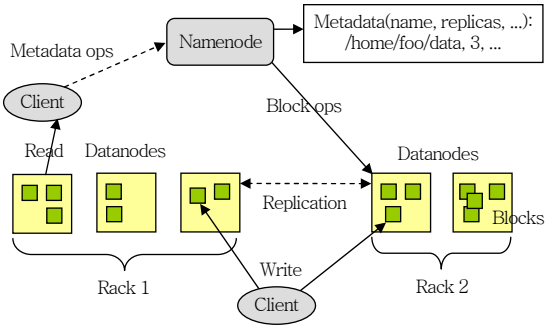
마스터에 대한 장애 처리와 회복을 위해서 파일 시스템 이름 공간과 파일의 chunk 매핑 변경 연산을 로깅하고 마스터의 상태를 여러 섀도 마스터에 복제한다.

Chunk 서버는 로컬 디스크에 chunk를 저장하고 관리하면서 클라이언트로부터의 chunk 입출력 요청을 처리한다. chunk는 마스터에 의해 생성되고 삭제될 수 있으며 유일한 식별자에 의해 구별된다. 마스터는 하나의 chunk 서버를 primary로 지정하여 복제본의 갱신 연산이 일관되게 이루어질 수 있도록 보장한다.

4. Hadoop Distributed File System

Hadoop은 Apache Lucene 프로젝트의 일부분으로 진행되고 있는 프로젝트로 Hadoop 분산 파일 시스템(HDFS)과 MapReduce 구현 등을 포함한다. 그리고 Apache Lucene 프로젝트는 이전에 먼저 시작된 웹 검색 소프트웨어 개발 프로젝트인 Apache Nutch 프로젝트의 일부분으로 진행되었다.

HDFS는 처음에 Apache Nutch 웹 검색 엔진의 기반 시스템으로 Java로 개발되었다. HDFS는 Google 파일 시스템과 유사한 특징을 갖는다.



(그림 4) HDFS 구조

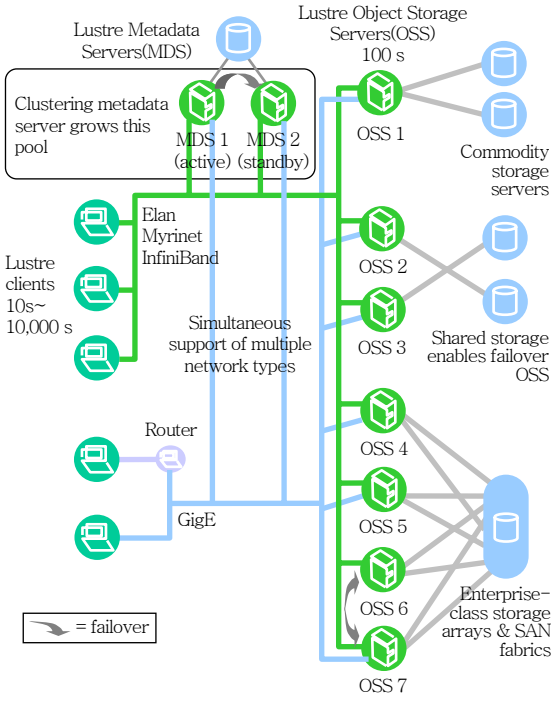
HDFS[9]는 (그림 4)에서와 같이 하나의 namenode와 다수의 datanode들로 구성된다. Namenode는 파일 시스템의 이름 공간을 관리하면서 클라이언트로부터의 파일 접근 요청을 처리한다. HDFS에서 파일 데이터는 블록 단위로 나뉘어서 여러 datanode에 분산되어 저장된다. 그리고 블록들은 가용성을 보장하기 위하여 다시 복제되어 저장된다. 따라서 datanode는 클라이언트로부터의 데이터 입출력 요청을 처리한다. HDFS에서 파일은 한번 쓰여지면 변경되지 않는다고 가정한다. 따라서 HDFS는 데이터에 대한 스트리밍 접근을 요청하며 배치 작업에 적합한 응용을 대상으로 한다.

Namenode는 datanode들로부터 하트 비트(heartbeat)를 주기적으로 받으면서 datanode들의 상태를 체크한다. 또한 하트비트 메시지에 포함된 블록 정보를 가지고 블록의 상태를 체크할 수 있다.

HDFS는 client, namenode, datanode 간의 통신을 위하여 TCP/IP 네트워크 상에서 RPC를 사용한다.

5. Lustre

Lustre[10]는 Cluster File Systems Inc.에서 개발된 객체 기반 클러스터 파일 시스템이다. Lustre는 (그림 5)와 같이 클라이언트 파일 시스템, 메타데이터 서버, 객체 저장 서버들로 구성된다. 그리고 이들은 고속 네트워크로 연결된다. Lustre에서는 계층화된 모듈 구조로 TCP/IP, Infiniband, Myri-



(그림 5) Lustre 구조

net, Quadrics Elan3, Quadrics Elan4 등과 같은 네트워크를 지원한다.

클라이언트 파일 시스템은 리눅스 VFS 하에서 설치할 수 있는 파일 시스템으로 메타데이터 서버와 객체 저장 서버들과 통신하면서 클라이언트 응용에 파일 시스템 인터페이스를 제공한다. 메타데이터 서버는 파일 시스템의 이름 공간과 파일에 대한 메타데이터를 관리한다. 그리고 객체 저장 서버는 파일 데이터를 저장하고 클라이언트로부터의 객체 입출력 요청을 처리한다. 객체는 객체 저장 서버들에 스트라이핑 되어 분산 저장된다.

Lustre는 Unix 시맨틱을 제공하면서 파일 메타데이터에 대해서는 write-back 캐시를 지원한다. 이를 위해 클라이언트에서 메타데이터 변경에 대한 갱신 레코드를 생성하고 나중에 메타데이터 서버에 전달한다. 그러면 메타데이터 서버는 전달된 갱신 레코드를 재수행하여 변경된 메타데이터를 반영한다. 그리고 메타데이터 서버에서는 메타데이터를 동시에 접근하는 부하에 따라 클라이언트 캐시에서 write-back 캐시를 지원하거나 또는 메타데이터 서버에서 메타데이터 처리를 수행하는 방식을 적용한다. 따라서 메타데이터에 대한 동시 접근이 적을 경우에는 클라이언트 캐시를 이용한 write-back 캐시를 사용하고 메타데이터에 대한 동시 접근이 많을 경우에는 클라이언트 캐시를 사용함으로써 발생할 수 있는 오버헤드를 줄이기 위하여 메타데이터 서버에서 처리하도록 하는 방식을 적용한다.

Lustre는 파일 메타데이터와 파일 데이터에 대한 동시성 제어를 위해 별도의 잠금을 사용한다. 메타데이터를 접근하기 위해서는 메타데이터 서버의 잠금 서버로부터 잠금을 획득해야 하고 파일 데이터를 접근하기 위해서는 해당 데이터가 저장된 객체 저장 서버의 잠금 서버로부터 잠금을 획득하여야 한다. 또한 Lustre에서는 클라이언트와 메타데이터 서버 간의 네트워크 트래픽을 최소화하기 위하여 메타데이터에 대한 잠금 요청 시에 메타데이터를 접근하는 의도를 같이 전달하는 intent 기반 잠금 프로토콜을 사용한다. 따라서 메타데이터 서버는 메타데이터 접근 의도에 따라 해당 동작을 수행하고 잠금을 승인해 주는 처리를 함께 수행함으로써 클라이언트와 메타데이터 서버 간의 네트워크 트래픽을 줄일 수 있다.

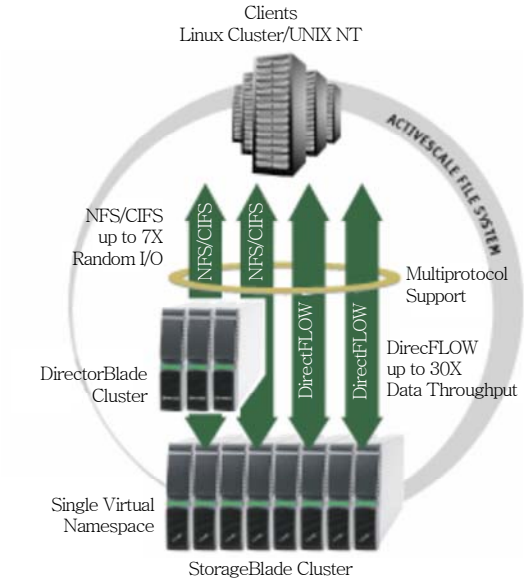
6. Panasas

Panasas ActiveStor Storage Cluster[11]는 리눅스 클러스터 시스템 환경에서 대량의 데이터 처리를 요구하는 응용 분야를 위한 객체 기반 저장 시스템 솔루션이다.

ActiveStor Storage Cluster는 (그림 6)에서와 같이 아래 세 가지 구성요소로 이루어진다.

• Panasas ActiveScale 파일 시스템(PanFS)

PanFS는 파일을 객체로 나누어 여러 개의 StorageBlade에 걸쳐 분산 저장할 수 있는 기능과 클라이언트에서 StorageBlade로부터 직접 파일 객체를 병렬로 입출력 할 수 있는 기능을 제공한다. 이를 위해서 PanFS는 클라이언트와 StorageBlade 간에 DirectFLOW 소프트웨어를 제공한다. 클라이언트



(그림 6) ActiveStor Storage Cluster 구조

에 설치되는 DirectFLOW Client는 클라이언트 커널에 적재되는 파일 시스템으로 리눅스 VFS 하에 구현되었으며 캐시 정책을 수행한다. 또한 StorageBlade로부터의 iSCSI/OSD 명령을 통하여 파일 객체를 직접 입출력 할 수 있는 기능을 수행한다. PanFS는 DirectFLOW 외에 네트워크 파일 시스템 프로토콜인 NFS와 CIFS를 지원한다. 이때 DirectorBlade는 NFS/CIFS 클라이언트에게 가상의 서버로 동작하게 된다.

• DirectorBlade 메타데이터 서버

DirectorBlade는 StorageBlade에 저장된 파일 객체에 대한 메타데이터를 관리하면서 파일 객체를 접근하려는 클라이언트에 대한 인증을 수행한다. 그리고 인증된 클라이언트에게 파일 객체에 대한 메타데이터를 제공하고 클라이언트 캐시의 일관성을 보장한다. DirectorBlade는 시스템의 확장성과 가용성을 위하여 클러스터로 구성될 수 있다.

• StorageBlade 스토리지 서버

StorageBlade는 파일 객체가 저장되는 공간으로 클러스터 형태로 구성되며 StorageBlade 간의 부하 분산과 RAID 재구성을 통해 확장과 회복을 지원한다. 그리고 각각의 StorageBlade에서 수행되는 DirectFLOW StorageBlade는 클라이언트들이 StorageBlade를 직접 접근하여 파일 객체를 입출력 할 수 있는 경로를 제공한다.

PanFS에서는 클라이언트와 StorageBlade 서버에서 파일 데이터에 대한 캐싱을 지원한다. 특히 클라이언트에서는 여러 번 나누어 저장될 데이터를 캐싱하였다가 한 번에 StorageBlade로 저장함으로써 StorageBlade와의 네트워크 트래픽을 줄일 수 있도록 한다. 또한 클라이언트에서는 파일에 대한 메타데이터와 파일 접근 인증 정보인 capability에 대한 캐싱을 제공한다. 그리고 클라이언트 응용에서 파일을 접근하는 패턴에 따라 파일에 대한 prefetch도 지원한다.

클라이언트에 캐싱된 파일 데이터에 대한 일관성은 DirectorBlade에 의해 보장된다. DirectorBlade는 클라이언트에 캐싱된 파일 데이터 또는 속성 정보에 대한 callback을 유지하면서 lease 기반의 캐시 일관성 기법에 의해 한 클라이언트에서 데이터가 변경되면 다른 클라이언트에 캐싱된 데이터를 무효화 하도록 함으로써 캐싱된 데이터의 일관성을 보장한다.

7. PVFS2

PVFS2[12]는 클러스터 환경에서 고속 병렬 입출력을 위한 파일 시스템으로, 기존의 PVFS를 보다 사용하기 편하고 모듈화된 구조로 개선한 것이다. 기본적으로 성능에 초점을 맞추었기 때문에 MPI I/O를 비롯한 다양한 슈퍼 컴퓨팅용 인터페이스를 제공한다. 또한 네트워크 측면에서도 기존의 TCP/IP 이외에도 Myrinet, Infiniband 등의 다양한 방식을 통하여 병렬 작업에 최적화된 클러스터의 구성을 가능하게 한다. PVFS2는 커널 수준에서 동작하는 파일 시스템으로 기본 Unix I/O 인터페이스 역시 제공하기 때문에 기존의 애플리케이션을 수정 없이 사용할 수 있다.

클러스터 환경에서 빈번하게 발생하는 메타데이

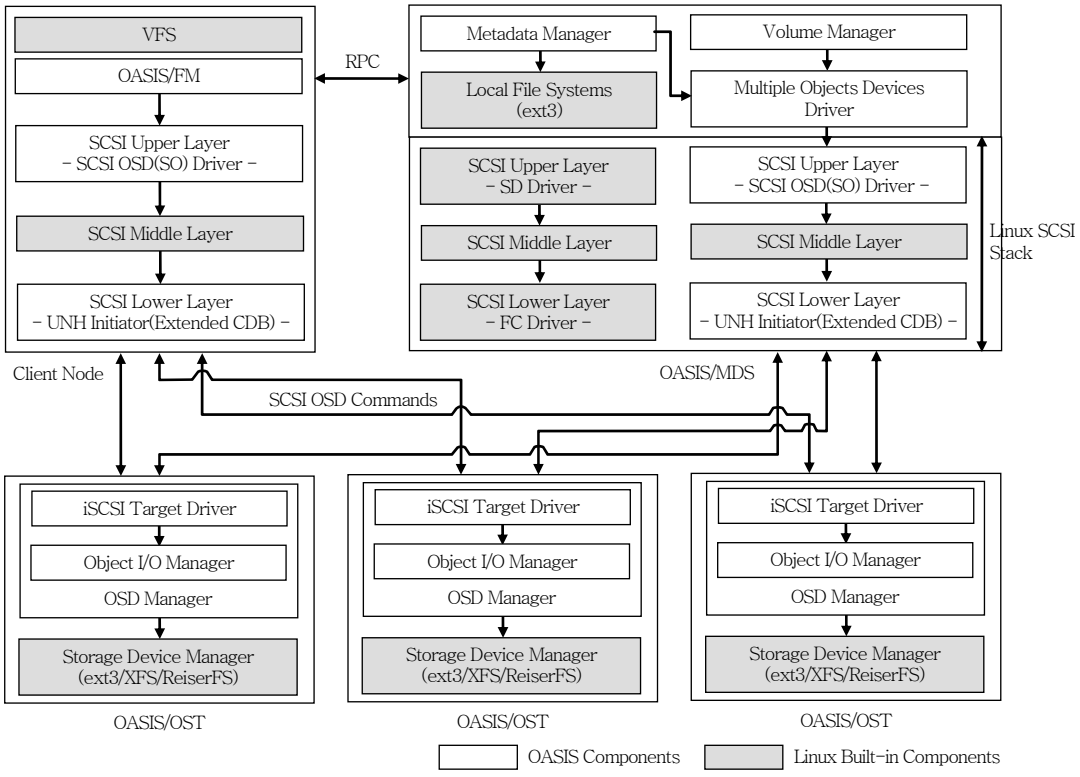
터 서버의 병목현상을 해결하기 위해서, PVFS2는 메타데이터 서버 역시 클러스터 형태로 구현이 가능하다. 이때 각 메타데이터 서버들은 전체 메타데이터 중 겹치지 않는 부분들을 서로 나누어서 그에 해당하는 사용자의 요청을 전담하여 처리하게 된다. 데이터 역시 다수의 데이터 서버에 스트라이핑을 통하여 나누어 저장되기 때문에 각 서버의 저장 공간을 효율적으로 사용할 수 있을 뿐만 아니라, 각 데이터 서버의 작업 부하 역시 고루 분산되게 할 수 있다.

PVFS2는 클러스터의 성능 측면에 초점을 맞추어서 클러스터의 가용성 측면에서는 여러 취약점을 갖고 있다. 메타데이터의 경우 복사본 없이 오직 하나만 존재하기 때문에 메타데이터 클러스터를 구성하는 서버들 중 어느 하나에 문제가 생길 경우 메타데이터의 손실을 피할 수 없다. 데이터 서버의 경우에도 파일 시스템 차원에서 데이터의 가용성을 늘리기 위한 지원은 하지 않고 전적으로 하드웨어 기능에 의존한다.

또한 동작의 설정이 정적으로 시작 전에 이루어지는 단점 때문에 클러스터의 서버 구성이 동적으로 변화하는 환경에는 부적합하다. 메타데이터 클러스터에서 각 서버가 담당할 메타데이터 영역의 설정이 초기에 정적으로 이루어지기 때문에, 메타데이터 서버가 추가될 경우 온라인으로 해당 서버를 이용하는 것은 불가능하고, 오프라인으로 이루어지는 경우에도 메타데이터 및 데이터의 이동, 각 서버 설정의 변경 등 번거로운 작업들을 요구한다. 또한 초기 설정 과정에서 클러스터에 참여하는 전체 서버의 목록을 사전에 알아야 한다는 단점이 있다.

8. OASIS

OASIS[13]는 한국전자통신연구원 저장시스템 연구팀에서 개발한 객체 기반 클러스터 파일 시스템이다. OASIS는 (그림 7)에서와 같이 세 개의 서브시스템으로 구성된다. 이들 서브시스템들은 기가비트



(그림 7) OASIS 구조

이더넷으로 서로 연결된다.

• FM

FM은 클라이언트에 설치할 수 있는 파일 시스템으로 Linux VFS 하에 적재될 수 있는 커널 모듈로 제공된다. 따라서 POSIX API를 지원한다. FM은 파일셋 단위로 마운트하여 사용할 수 있으며 RPC를 통하여 MDS로 파일 메타데이터에 대한 연산을 요청한다. 그리고 OST에 저장된 파일 데이터 객체를 접근하기 위하여 iSCSI 프로토콜을 이용하여 SCSI/OSD 명령을 전달한다. 이를 위해 SCSI/OSD 명령을 처리할 수 있는 장치 드라이버를 제공하며 SCSI/OSD 표준을 준수한다. 또한 FM에서는 다중 OST를 여러 RAID 방식으로 사용할 수 있는 장치 드라이버도 지원한다. 따라서 FM에서는 파일을 객체 단위로 다중 OST에 분산하여 저장할 수 있다. FM은 파일 데이터에 대한 read-ahead 캐시와 write-back 캐시를 제공한다.

• MDS

MDS는 파일 및 디렉토리에 대한 메타데이터를 관리한다. 파일 메타데이터에는 파일 객체가 저장된 OST들의 정보도 포함된다. MDS에서는 OST들을 묶어서 linear, RAID-0, RAID-1, RAID-5 형태로 구성하고 관리하는 기능도 제공한다. MDS에서는 여러 클라이언트에 캐싱되어 있을 수 있는 파일 메타데이터와 데이터에 대한 캐시 일관성을 보장하기 위하여 파일 단위의 callback 기반 잠금 기법을 제공한다. MDS는 커널 모듈로 구현되었으며 리눅스 커널의 VFS 함수를 직접 호출함으로써 하위 로컬 파일 시스템에 독립적이다.

• OST

OST는 파일 데이터를 객체 형태로 저장하고 관리한다. OST에서는 FM과 MDS로부터 iSCSI 프로토콜을 통해 전달된 SCSI/OSD 명령을 처리한다. OST는 커널 모듈로 구현되었으며 리눅스 커널의 VFS 함수를 직접 호출하도록 하여 하위 로컬 파일 시스템에 독립적이다.

Ⅲ. 결론

본 고에서는 클러스터 파일 시스템의 사례 시스템들로 Ceph, GlusterFS, Google 파일 시스템, Hadoop 분산 파일 시스템, Lustre, Panasas, PVFS2, OASIS를 살펴보았다.

위의 사례 시스템들에서 보는 바와 같이 대규모 클러스터 환경을 위한 분산 파일 시스템들은 주로 비대칭형 구조로 클라이언트와 메타데이터 서버 그리고 데이터 서버로 구성되며 아래에 기술된 특성들을 갖는다.

클라이언트에서는 기존 응용에 대한 호환성을 제공하기 위하여 POSIX 표준에 준하는 파일 시스템 인터페이스를 지원한다. 또한 POSIX 인터페이스 외에도 파일 시스템을 사용하는 인터넷 서비스 응용에서 요구하는 인터페이스를 추가로 지원하고 있다. 예를 들어, 원자적 데이터 추가, chunk가 할당된 서버 정보 등을 들 수 있다. 이러한 클라이언트는 대부분 구현상의 용이함 등의 이유로 FUSE 등을 이용하여 사용자 수준으로 구현되고 있다.

인터넷 서비스 응용은 주로 순차(sequential) 읽기 연산이 주를 이루는 작업 부하로 구성된다. 그리고 쓰기 연산의 경우는 갱신 연산 보다 추가 연산이 많은 비중을 차지한다. 따라서 이러한 작업 부하를 효율적으로 처리하기 위하여 메타데이터 또는 데이터를 캐싱하지 않거나 보다 완화된 캐시 일관성 모델을 사용하고 잠금 등을 사용함으로써 발생할 수 있는 오버헤드를 최소화 한다.

메타데이터 서버는 구조상 병목 지점이 될 수 있기 때문에 메타데이터 연산을 보다 빠르게 수행하기 위하여 모든 메타데이터를 메모리 상에 올려 놓고 해시 테이블과 같은 구조 등을 사용한다. 또한 메타데이터와 데이터를 최대한 분리함으로써 데이터를 접근하기 위하여 메타데이터를 접근해야 하는 부하를 줄이려고 한다.

메타데이터 서버에 대한 가용성을 보장하기 위한 방법으로 메타데이터 서버를 클러스터로 구성하는 방법을 주로 적용하고 있다. 하지만 아직까지 확장

가능하며 높은 성능을 제공할 수 있는 메타데이터 서버의 클러스터링 방법이 좀더 연구될 필요가 있다.

데이터 서버에 저장되는 파일 데이터는 입출력 성능을 고려하여 chunk 단위로 스트라이핑 되어 분산 저장된다. 이때 chunk의 크기는 서비스의 특성에 따라 가변적으로 설정할 수 있으며 큰 용량의 chunk를 지원한다.

또한 대규모 클러스터를 구성할 때 발생할 수 있는 빈번한 서버 고장에 대비하여 chunk를 여러 노드에 복제하여 저장한다. 이때 chunk 복제로 인한 성능 저하를 최소로 할 수 있으면서 복제본들 간의 일관성을 보장할 수 기법을 제공한다. 그리고 클러스터의 네트워크 구성 정보 등을 이용하여 복제본을 저장할 데이터 서버를 선정하고 chunk를 할당한다.

클러스터에 데이터 서버를 추가하거나 제거하는 작업은 클라이언트에 독립적으로 이루어진다. 따라서 새로운 데이터 서버가 추가되거나 제거되는 것과는 상관없이 클라이언트에서는 계속 파일 시스템을 접근할 수 있다.

메타데이터 서버에서는 하트 비트 등을 이용하여 주기적으로 데이터 서버의 상태를 체크하고 상태에 따라 데이터를 다른 서버로 마이그레이션하거나 또는 재복제, 재분산하는 작업을 수행한다.

네트워크 측면에서 클라이언트와 메타데이터 서버 그리고 데이터 서버들 간에는 TCP/IP 외에 Infiniband, Myrinet 등 고속의 네트워크들도 지원하고 있다.

● 용 어 해 설 ●

FUSE(Filesystem in Userspace): FUSE는 SourceForge에서 A Virtual Filesystem(AVFS) 프로젝트의 일부분으로 시작되었다가 현재는 독립적인 프로젝트로 진행되고 있는 프로젝트로 커널 모듈, 사용자 라이브러리 그리고 마운트 유틸리티로 구성되어 있다. FUSE는 일반 사용자가 단지 사용자 프로그램으로 자신의 파일 시스템을 구현하고 마운트 할 수 있는 방법을 제공한다.

객체 기반 저장 장치(Object-based Storage Device: OSD): 객체 기반 저장 장치는 물리적인 저장공간 관리와 같은 기능을 저장 장치에서 직접 수행하도록 함으로써 성능과 확장 그리고 플랫폼 독립적인 데이터의 안전한 공유 등을 제공할 수 있도록 한다. 객체 기반 저장 장치에서 데이터는 논리적인 객체 단위로 저장되며 객체에는 데이터 접근 방법, 데이터 속성 정보, 데이터 보안 방법 등을 포함할 수 있다. 객체 기반 저장 장치에 대한 인터페이스는 SNIA에서 정의되었으며, 이후에 ANSI T10에서 SCSI 프로토콜의 추가 명령어로 표준화되었다. 현재는 OSD-2에 대한 표준화가 진행중이다.

약 어 정 리

EBOFS	Extent and B-tree based Object File System
FM	File Manager
FUSE	File System in User Space
HDFS	Hadoop Distributed File System
IPTV	Internet Protocol Television
iSCSI	Internet Small Computer Systems Interface
MDS	Metadata Server
MPI	Message Passing Interface
OASIS	Object-based storage Architecture for Scalability, Intelligence and Security
OSD	Object-based Storage Device
OST	Object-based Storage Target
POSIX	Portable Operating System Interface
PVFS	Parallel Virtual File System
RAID	Redundant Array of Independent Disks
RDMA	Remote Direct Memory Access
UCC	User-Created Content
VFS	Virtual File System

참 고 문 헌

[1] M. Mesnier, G. Ganger, and E. Riedel, “Object-based Storage,” *In IEEE Communications Magazine*, Aug. 2003, pp.84-90.

[2] http://ceph.sourceforge.net

[3] Sage Weil, Scott A. Brandt, Ethan L. Miller, Darrell D.E. Long, and Carlos Maltzahn, “Ceph: A Scalable, High-Performance Distributed File System,” *In Proc. of Conf. on Operating Systems Design and Implementation*, Nov. 2006, pp.307-320.

[4] Sage Weil, Scott A. Brandt, Ethan L. Miller, and Carlos Maltzahn, “CRUSH: Controlled, Scalable, De-

centralized Placement of Replicated Data," *In Proc. of the 2006 ACM/IEEE Conf. on Supercomputing*, Nov. 2006.

[5] Sage Weil, Kristal Pollack, Scott A. Brandt, and Ethan L. Miller, "Dynamic Metadata Management for Petabyte-Scale File Systems," *In Proc. of the ACM/IEEE Conf. on Supercomputing*, Nov. 2004.

[6] Feng Wang, Scott A. Brandt, Ethan L. Miller, and Darrell D.E. Long, "OBFS: A File System for Object-Based Storage Devices," *In Proc. of the 21st IEEE/12th NASA Goddard Conf. on Mass Storage Systems and Technologies*, Apr. 2004, pp.283-300.

[7] http://www.gluster.org

[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," *In Proc. of ACM Symp. on Operating Systems Principles*, 2003, pp.20-43.

[9] http://lucene.apache.org/hadoop

[10] http://wiki.lustre.org

[11] http://www.panasas.com

[12] http://www.pvfs.org

[13] 민영수, 차명훈, 김영철, 진기성, 이상민, 정병권, 김준, "객체기반 저장 장치를 이용한 클러스터 파일 시스템의 구현," 한국차세대컴퓨팅학회 논문지, Vol.2, No.4, 2006, pp.42-52.