

클라우드 컴퓨팅을 위한 분산 파일 시스템 기술 동향

A Trend to Distributed File Systems for Cloud Computing

클라우드 컴퓨팅 특집

민영수 (Y.S. Min)	저장시스템연구팀 Post-Doc.
진기성 (K.S. Jin)	저장시스템연구팀 선임연구원
김홍연 (H.Y. Kim)	저장시스템연구팀 책임연구원
김영균 (Y.K. Kim)	저장시스템연구팀 팀장

목 차

-
- I. 서론
 - II. 요구 사항
 - III. 분산 파일 시스템 기술
 - IV. 결론

최근 클라우드 컴퓨팅 시장에 진출했거나 진출을 선언한 글로벌 IT 기업들을 살펴보면 이미 보유하고 있는 기반 기술들을 활용하거나 상호 협력을 통해 다양한 클라우드 서비스들을 제공함으로써 급격하게 성장하고 있는 클라우드 컴퓨팅 시장에서 자신들의 영역을 지속적으로 확장해 나가고 있다. 분산 파일 시스템은 데이터의 저장과 관리뿐만 아니라 상위 계층 서비스가 요구하는 충분한 성능과 안정성을 보장해주기 위한 클라우드 컴퓨팅의 핵심 기술 중의 하나이다. 본 고에서는 클라우드 컴퓨팅을 위해 분산 파일 시스템이 갖추어야 할 사항들과 클라우드 컴퓨팅에서 활용 가능한 분산 파일 시스템들을 소개하고 현재 클라우드 컴퓨팅 시장에서 활용되고 있는 분산 파일 시스템의 동향을 살펴보고자 한다.

I. 서론

최근 클라우드 컴퓨팅 시장에 진출했거나 진출을 선언한 Google, IBM, Microsoft, Oracle 등과 같은 글로벌 IT 기업들은 그 동안 쌓아온 기술력을 바탕으로 클라우드 컴퓨팅을 제공하는 데 필요한 IT 인프라를 지속적으로 확충해 나가고 있으며 제반 기술들을 개발하고 향상시키기 위해 막대한 투자를 쏟아붓고 있다. 또한 공개 소프트웨어의 도입이나 기업 간의 전략적인 협력을 통해 부족한 부분들을 상호 보완하면서 다양한 클라우드 서비스들을 제공함으로써 급격하게 성장하고 있는 클라우드 컴퓨팅 시장에서 자신들의 영역을 확장하기 위해 끊임없이 노력하고 있다.

최근 클라우드 컴퓨팅 시장에 뛰어든 기업들에게 있어서 막대한 양의 데이터를 저장하고 관리하기 위해 지불해야 하는 비용을 최대한 줄이는 것이 가장 큰 이슈로 부각되고 있으며, 점점 다양해지고 있는 사용자들의 요구를 충족시킬 수 있는 클라우드 서비스들을 찾아내고 편리하게 사용할 수 있는 환경을 제공하여 보다 많은 사용자들을 확보하기 위한 경쟁이 치열해지고 있다.

클라우드 컴퓨팅을 위해 IT 인프라를 저렴하게 구축하고 효율적으로 활용할 수 있는 기술들에 대한 연구가 활발히 진행되고 있으며, 막대한 양의 데이터를 수많은 서버들에 분산 저장하고 관리하는 분산 파일 시스템 기술은 그러한 기술 중의 하나이다. 클라우드 컴퓨팅에서 요구하는 분산 파일 시스템은 단순히 데이터의 저장과 관리만을 하는 것이 아니라 하드웨어의 장애에 유연하게 대처하여 서비스가 중단되지 않도록 하고 적절한 병렬 처리를 통해 서비

스가 요구하는 성능도 만족시킬 수 있어야 한다.

현재 수많은 분산 파일 시스템들이 존재하지만 최근까지 클라우드 컴퓨팅에 활용되고 있는 분산 파일 시스템은 그리 많지 않다. 대표적으로 Google의 Google 파일 시스템[2]과 Apache의 Hadoop 분산 파일 시스템[3]이 클라우드 컴퓨팅에서 가장 잘 알려진 분산 파일 시스템이다.

본 고에서는 클라우드 컴퓨팅을 위해 분산 파일 시스템이 갖추어야 할 사항들과 클라우드 컴퓨팅에서 활용 가능한 분산 파일 시스템들을 살펴보고, ETRI에서 개발중인 GLORY 분산 파일 시스템을 소개한다. 또한 현재 클라우드 컴퓨팅 시장에서의 분산 파일 시스템 동향을 살펴보고자 한다.

II. 요구 사항

클라우드 컴퓨팅을 위해 분산 파일 시스템이 갖추어야 할 사항들은 실로 다양하다. 이러한 사항들을 도출하기 위해 클라우드 컴퓨팅의 대표적인 특징들을 분산 파일 시스템 측면에서 살펴보면 다음과 같이 몇 가지로 크게 요약해 볼 수 있다.

첫째, 클라우드 컴퓨팅을 위한 분산 파일 시스템들이 다루는 데이터와 서버의 규모는 기존의 분산 파일 시스템들과는 비교가 되지 않을 정도로 거대하다는 점이다. 대부분의 핵심적인 사항들은 여기에서 발생하며 비용적인 측면에서의 효율성, 지속적으로 증가하는 데이터의 수용, 빈번하게 발생하는 고장에 대한 대처, 관리의 편리성과 같은 사항들이 해당된다.

둘째, 점점 다양해지고 있는 사용자들의 요구를 충족시키기 위해 제공되는 클라우드 서비스들이 분산 파일 시스템에게 만족할 만한 데이터의 입출력과 처리 성능을 요구한다는 점이다. 여기에는 대용량 파일에 대한 신속한 입출력 성능, 네트워크 위상 구조 인식을 통한 데이터 최적 배치, 효과적인 캐시의 사용, 순간적으로 집중되는 부하의 유연한 대처와 같은 사항들이 해당된다.

셋째, 클라우드 컴퓨팅을 활용하고자 하는 기업들뿐만 아니라 일반 사용자들은 데이터에 대한 보안

● 용어 해설 ●

클라우드 컴퓨팅: 인터넷을 기반으로 하여 IT 자원들을 서비스 형태로 제공하는 컴퓨팅이다[1].

분산 파일 시스템: 클라이언트가 서버 상에 저장된 데이터를 마치 자신에게 저장되어 있는 것처럼 접근하고 처리할 수 있는 클라이언트/서버 기반의 파일 시스템이다.

문제를 가장 불안해하고 있으며, 확실하고 안전한 보안 체계를 요구한다는 점이다. 여기에는 데이터에 대한 암호화, 데이터 영역에 대한 사용자간 엄격한 접근 제어, 사용자 데이터에 대한 관리자의 접근 제한 같은 사항들이 해당된다.

넷째, 사용자가 저장한 데이터에 오류가 발생하지 않도록 방지하고, 저장 공간을 최적으로 사용하기 위한 방법들을 요구한다는 점이다. 여기에는 데이터 오류 감지 및 복구, 데이터 중복 제거와 같은 사항들이 해당된다.

1. 저비용

클라우드 컴퓨팅에서 다루어지는 데이터는 꾸준히 증가하고 있으며 이러한 데이터를 저장하고 관리하기 위해 막대한 IT 인프라가 요구된다. 만일 고가의 서버와 고속의 네트워크를 활용하여 이러한 인프라를 구축하고 유지할 수 있다면 두말할 나위 없이 좋겠지만 비용을 고려했을 때 현실적으로 쉽지 않은 것이 사실이다. 따라서 클라우드 컴퓨팅 시장에 뛰어들어 기업에게 있어서 이러한 인프라를 구축하고 유지하는 데 드는 비용을 최소화하는 것은 가장 중요한 경쟁력 중의 하나이다. 이를 위해 저렴한 서버들과 네트워크를 활용하여 비용을 대폭적으로 줄이면서도 비슷한 성능을 낼 수 있는 분산 파일 시스템 기술들을 개발하고 적용해야 한다.

2. 확장성

클라우드 컴퓨팅 기업들은 꾸준히 증가하는 데이터의 저장과 관리를 위해 초기 구축된 IT 인프라를 지속적으로 확장해야 한다. 이러한 확장에 있어서 분산 파일 시스템은 논리적으로 공간 확장을 무한히 할 수 있어야 하며 공간을 확장하거나 축소할 때 서비스의 중단이 발생하지 않도록 해야 한다. 간단히 말하면, 서비스의 중단 없이 공간의 관리가 가능해야 한다는 것이다. 이를 위해 분산 파일 시스템은 모든 서버들의 공간을 사용자들에게 단일 공간으로 보이도록 가상화를 수행하며, 이러한 가상화를 통해

서비스의 중단 없이 자유롭게 공간을 관리하고 사용자들에게는 단일 저장소로 사용할 수 있도록 한다.

3. 안정성

앞서 언급하였던 저비용 측면에서 저가의 장비들을 대규모로 활용하여 시스템을 구축함으로써 고장이 빈번하게 발생할 수 밖에 없다. 물론 고가의 장비들은 저가의 장비들에 비해 고장이 좀 더 적게 발생하는 것은 하지만 고장이 발생한다는 사실 자체는 변하지 않는다. 중요한 점은 이러한 고장으로 인해 시스템 전체 또는 서비스를 중단시키는 상황이 발생하지 않아야 한다는 것이다. 분산 파일 시스템은 모니터링을 통하여 고장이 발생한 상황을 인지하고 적절하게 대처함으로써 서비스가 중단되는 상황을 방지해야 한다. 또한 고장이 발생한 서버나 디스크로 인해 데이터가 유실되는 것을 방지하기 위해 데이터의 중복 저장과 같은 안정적인 데이터 보관 방법이 갖춰져야 한다.

4. 고성능

기존의 PC 환경에서 사용하던 수많은 응용들이 클라우드 서비스를 통해 클라우드 컴퓨팅 속으로 들어오고 있다. 이러한 응용들의 접근 패턴은 분산 파일 시스템의 설계에 있어서 성능과 밀접한 관련을 갖는다. 예를 들면 사진, 동영상과 같은 대용량의 데이터들을 다루는 클라우드 서비스들이 최근 많은 관심을 끌고 있는데, 분산 파일 시스템의 입장에서 바라볼 때 이러한 응용들은 대용량의 데이터를 순차적으로 입출력하는 패턴을 가지며, 대부분 저장된 데이터를 읽어가는 연산을 수행한다. 이러한 패턴에 최적화된 분산 파일 시스템은 이러한 패턴에 있어서는 최상의 성능을 나타내겠지만, 저용량의 임의적인 입출력 패턴의 경우에는 현저하게 성능이 저하될 수도 있다. 결과적으로 분산 파일 시스템이 모든 접근 패턴에 최고의 성능을 보장할 수 있으면 좋겠지만 사실상 그것은 상당히 어려우며 분산 파일 시스템 상위에서 접근 패턴 자체를 분산 파일 시스템이 지

원하는 최적의 접근 패턴으로 변형하여 최상의 성능을 얻어내거나 적절한 다른 분산 파일 시스템을 활용하는 것이 보다 현실적인 대안이 될 수 있다.

분산 파일 시스템은 서버, 스위치, 랙 등의 네트워크 위상 구조를 인식하고 이를 이용하여 최적으로 데이터를 배치함으로써 클라이언트의 요청을 빠르게 처리할 수 있어야 한다. 또한 메모리 캐시를 효율적으로 활용하여 디스크 입출력을 최소화함으로써 성능을 향상시킬 수 있다.

앞서 예로 들었던 클라우드 서비스에서 수많은 사용자들이 관심을 가지고 짧은 기간 안에 접근하게 되는 hot 데이터라 불리는 특정 데이터는 과도한 접근으로 인해 서비스가 불가능한 상태에 빠질 수도 있는데, 이러한 것을 방지하기 위한 방법이 갖춰져야 한다.

5. 간편성

클라우드 컴퓨팅을 활용하는 기업들과 일반 사용자들은 클라우드 서비스를 통해 제공되는 기능들만을 사용하므로 분산 파일 시스템에 대해 알 필요가 전혀 없지만 분산 파일 시스템을 활용하여 클라우드 서비스를 제공하고자 하는 기업의 관리자는 현재 분산 파일 시스템에 의해 관리되고 있는 모든 사항들을 모니터링하고 분석해야 하며 분산 파일 시스템의 상세한 사항을 이해하고 있어야 한다. 소규모의 시스템의 경우는 관리자가 명령 라인을 통해 분산 파일 시스템을 관리하고 여러 상황들을 확인할 수 있지만, 대규모의 클라우드 컴퓨팅 환경에서는 이러한 작업이 불편하고 어려울 수밖에 없다. 따라서 클라우드 컴퓨팅을 위한 분산 파일 시스템은 관리자에게 웹 또는 GUI 형태의 관리 도구를 통해 분산 파일 시스템을 관리하는 모든 서버나 네트워크의 상황을 일목요연하게 보여주고 손쉽게 관리할 수 있는 기능을 제공해야 한다.

6. 보안성

클라우드 서비스를 이용하는 사용자들이 가장 우

려하는 사항이 바로 보안성이다. 우려를 하고 있는 사항은 크게 두 가지로 볼 수 있는데, 첫째는 관리자나 해당 기업의 관계자에 의한 정보 유출이다. 이 유형의 경우는 클라우드 서비스를 이용하는 사용자들의 데이터가 사용자의 동의 없이 상업적으로 이용되거나 유출되는 것이다. 또한 사용자의 동의 없이 삭제된 데이터가 보관되는 것도 포함된다. 둘째는 취약한 보안 허점으로 악의적인 해킹에 의해 사용자의 데이터가 손실되거나 유출되는 것이다. 이러한 사용자들의 우려를 불식시키기 위해서는 철저한 데이터 관리와 더불어 제도적인 사항이 뒷받침되어야 하며, 전문적인 보안관련 기업들과의 협력을 통해 지속적으로 보안을 강화해야 한다. 분산 파일 시스템의 입장에서는 철저한 사용자 인증과 접근 허용 권한 관리가 필수적이다.

7. 무결성

데이터를 저장하다 보면 하드웨어적인 오류나 알 수 없는 원인에 의해 의도된 대로 데이터가 저장되지 않고 오류가 발생하는 경우가 종종 생긴다. 분산 파일 시스템은 이러한 상황으로부터 사용자의 데이터를 보호하기 위해 오류를 검출하고 복구할 수 있어야 한다. 클라우드 컴퓨팅에서 다루어지는 데이터의 양은 너무나도 방대하여 기존의 파일 시스템 검사 방법으로는 모든 데이터를 검사한다는 것이 사실상 불가능하다. 따라서 분산 파일 시스템의 전체 데이터를 빠르고 효과적으로 검사할 수 있는 방법이 갖춰져야 한다.

Ⅲ. 분산 파일 시스템 기술

현재까지 무수히 많은 분산 파일 시스템들이 존재하고 있으며 여전히 진화를 거듭하고 있다. 최근 클라우드 컴퓨팅이 부각되면서 기존의 분산 파일 시스템 기술들이 다양한 형태로 활용되고 있으며, 적절한 변형을 통해 클라우드 컴퓨팅으로의 접근을 꾀하고 있다. 이 장에서 살펴볼 클라우드 컴퓨팅에 활

용되고 있거나 활용이 가능한 분산 파일 시스템들은 앞서 살펴보았던 요구 사항들을 반영하고 있다.

1. Google 파일 시스템

Google은 클라우드 서비스를 제공하기 위해 자체 개발한 분산 파일 시스템인 Google 파일 시스템(이하 GFS)을 사용하고 있다. GFS는 (그림 1)에서처럼 여러 클라이언트에 의해 접근되는 단일 마스터와 여러 청크(chunk) 서버들로 구성되며, 저가의 리눅스 서버상에서 사용자 수준 프로세스로 구동된다. 마스터는 모든 파일 시스템 메타데이터를 관리하고 시스템 전반적인 동작을 제어하는 역할을 수행하며, 청크 서버들은 실제 데이터를 보관하고 클라이언트의 입출력 요청을 처리하는 역할을 수행한다. 이처럼 GFS는 메타데이터와 데이터의 흐름을 분리한 비대칭형 구조를 통해 확장성과 고성능을 보장하고 빈번하게 발생하는 장애에 적절히 대처함으로써 안정성을 확보한다.

GFS에서 파일들은 청크라 불리는 64 메가바이트 크기의 단위로 나뉘어 관리되며 각 청크들은 여러 청크 서버에 분산되어 저장된다. 또한 각 청크에 대한 다수의 사본을 여러 청크 서버에 분산하여 저장함으로써 장애로 인한 데이터의 손실을 방지한다. GFS는 이렇게 청크라 불리는 큰 단위로 데이터를 관리함으로써 마스터가 관리해야 하는 메타데이터의 수를 줄이고 메타데이터를 교환하기 위해 필요로 하는 네트워크 부하를 감소시킨다.

가. 클라이언트

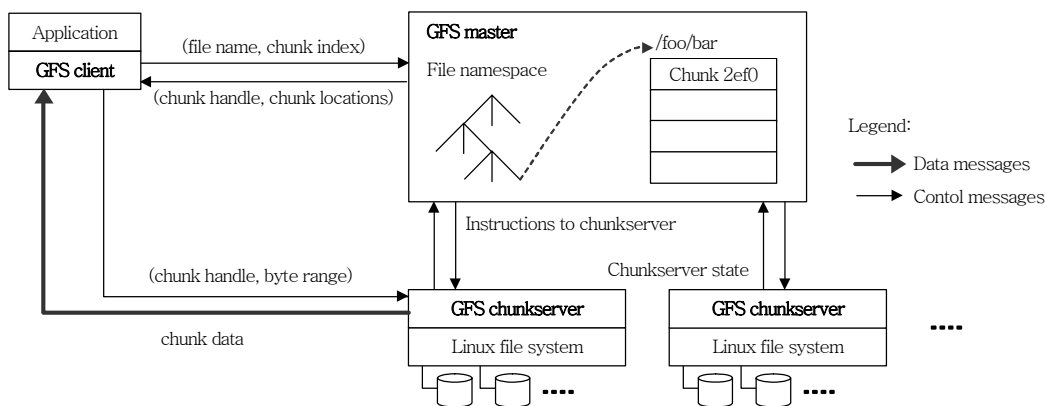
GFS는 대용량 파일의 입출력 연산에 맞도록 개발된 분산 파일 시스템으로 GFS 클라이언트는 파일 시스템 인터페이스와 유사한 자체적인 인터페이스를 제공하며 GFS 응용들은 이러한 인터페이스를 기반으로 개발된다.

GFS 응용들의 요청을 처리하기 위한 GFS 클라이언트의 역할은 크게 두 가지로 볼 수 있는데, 첫째는 마스터와 통신하여 메타데이터 연산을 수행하는 것이고, 둘째는 청크 서버들과 직접 통신하여 데이터 입출력을 수행하는 것이다. 만일 GFS 응용이 어떤 파일의 일부 데이터를 얻기 위해 GFS 클라이언트에게 읽기 요청을 하였다면 GFS 클라이언트는 다음과 같은 순서로 동작을 수행한다.

- GFS 응용으로부터 전달받은 파일의 이름과 청크의 번호를 마스터에게 전달한다.
- 마스터로부터 청크의 ID에 해당하는 핸들과 청크가 위치한 청크 서버들의 위치를 받는다.
- 청크가 위치한 청크 서버에게 청크 핸들과 청크 내에서 얻고자 하는 데이터 범위를 전달한다.
- 청크 서버로부터 전달되는 데이터를 받는다.

나. 마스터

GFS에서 가장 핵심적인 역할을 수행하는 마스터는 네임스페이스, 접근 제어 정보, 파일과 청크들간



(그림 1) GFS의 구조

의 사상 정보, 청크들의 위치 정보와 같은 파일 시스템 메타데이터를 관리하고 네임스페이스 및 접근 제어에 대한 연산, 청크와 사본의 관리, 청크 서버의 관리와 상태 모니터링 및 장애 처리, 사용되지 않는 저장 공간의 회수 등 시스템 전반적인 동작들을 제어한다.

마스터는 메타데이터 연산을 보다 빠르게 수행하기 위해 초기 기동이나 장애 발생으로 인한 재 기동 시에 청크 서버들로부터 청크에 대한 정보를 받아서 메모리 상에 메타데이터를 구축하고 로그를 활용하여 구축 이후에 발생하는 메타데이터 변경 사항을 디스크에 저장한다. 이러한 로그는 마스터의 장애에 대비하여 여러 마스터 후보 서버들로 복제되며 마스터에 장애가 발생하면 즉시 마스터 후보 서버들 중 하나가 새로운 마스터로 동작을 수행한다. 또한 읽기 전용의 새도(shadow) 마스터라 불리는 서버들을 두어 마스터의 장애 시에 읽기 연산에 대한 가용성을 높인다.

마스터는 주기적으로 수집되는 청크 서버들의 heartbeat 정보를 통하여 청크 서버들의 상태를 파악하고 해당 청크 서버들이 수행해야 할 작업들을 알려준다. 마스터는 어떤 청크 서버의 장애가 감지되면 그 청크 서버가 포함하고 있는 청크들에 대해 다른 청크 서버에 있는 사본을 활용하여 또 다른 청크 서버로 사본을 복제하는 방법을 통해 안정성을 확보한다.

만일 어떤 청크의 모든 사본들이 동일한 네트워크 스위치에 연결된 서버들에 저장되어 있을 때 그 네트워크 스위치에 고장이 발생하면, 그 청크를 서비스해 줄 수 없는 심각한 상황이 발생한다. GFS 마스터는 청크의 사본들이 위치할 청크 서버들을 선택할 때 네트워크 위상 구조를 활용하여 서로 다른 스위치에 연결된 서버들을 선택하여 이러한 상황이 발생되지 않도록 한다.

마스터는 저장 공간의 활용도를 높이고 청크 서버들간의 부하를 적절하게 분산시키기 위해 주기적으로 청크 사본들의 분산 배치 상황을 검사하여 청크 사본들에 대한 청크 서버들간의 재분배를 수행한

다. 또한 저장 공간 확장을 위해 새로 추가된 청크 서버로 과도하게 물리는 부하를 방지하고 점진적으로 저장 공간 활용도가 균형을 이룰 수 있도록 유도하기 위해 이러한 재분배를 수행한다.

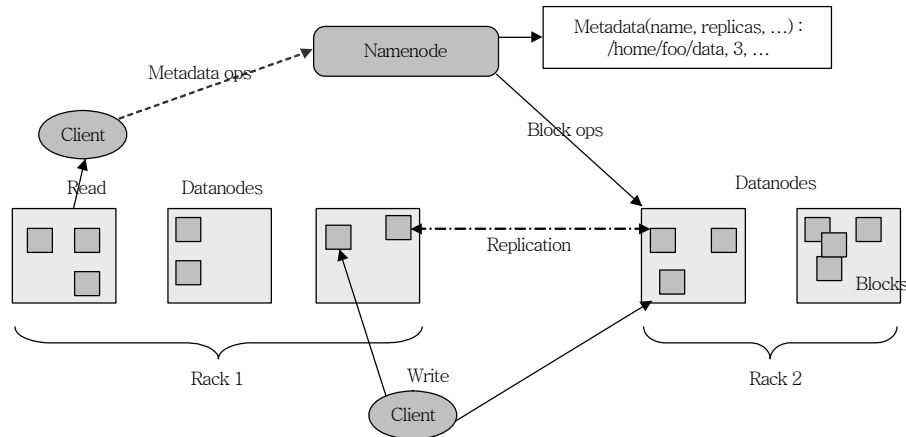
다. 청크 서버

청크 서버는 청크 데이터의 관리와 클라이언트가 요청하는 입출력을 처리하며 주기적으로 마스터에게 자신의 상태와 자신이 관리하고 있는 청크들에 대한 정보를 heartbeat을 통해 전달한다. 청크 서버는 청크들을 ext3와 같은 로컬 파일 시스템의 파일로 저장하며 각 청크에 대한 무결성을 보장하기 위해 별도의 파일로 저장되는 checksum을 활용한다.

장애가 발생했던 청크 서버는 재 기동 시에 마스터에게 자신이 재 기동 되었음을 알려면서 자신이 관리하고 있는 청크들에 대한 정보를 보낸다. 마스터는 재 기동된 청크 서버가 관리하고 있는 청크들의 정보와 자신이 관리하고 있는 청크들의 정보를 비교하여 일치하지 않는 청크들을 삭제하도록 재 기동된 청크 서버에게 알려주며, 이러한 과정을 통해 재 기동된 청크 서버는 최신의 상태를 유지하게 된다.

2. Hadoop 분산 파일 시스템

Hadoop 분산 파일 시스템(이하 HDFS)은 신뢰성과 확장성을 갖춘 분산 컴퓨팅을 위한 오픈 소스 소프트웨어 개발 프로젝트인 Hadoop에서 분산 컴퓨팅 프레임워크를 지원하기 위해 개발된 분산 파일 시스템이다. HDFS는 현재 Amazon, IBM, Yahoo 등과 같은 글로벌 IT 기업들의 클라우드 컴퓨팅 플랫폼의 기반이 되는 분산 파일 시스템으로 가장 널리 활용이 되고 있다[4]. HDFS는 GFS를 본보기로 삼아 개발된 분산 파일 시스템으로 플랫폼간의 이식성을 보장하기 위해 자바를 사용하여 구현되었으며 공개 소프트웨어이다. HDFS는 (그림 2)에서처럼 이름만 다를 뿐 GFS와 동일한 구조와 기능을 제공한다. 네임노드는 GFS의 마스터와 동일하며 데이터 노드는 GFS의 청크 서버와 동일하다. 또한 HDFS



(그림 2) Hadoop 분산 파일 시스템의 구조

에서 블록이라 불리는 파일 관리 단위는 GFS의 청크와 동일하다.

가. 클라이언트

HDFS 클라이언트는 HDFS 파일을 생성할 때 우선 자신의 로컬 저장 영역에 한 블록 크기의 임시 파일을 생성하고 그 파일에 먼저 데이터를 기록한다. HDFS 클라이언트에서 HDFS 파일이 생성되는 과정을 살펴보면 다음과 같다.

- 한 블록 크기의 임시 파일이 데이터로 모두 채워지거나 더 이상 기록될 데이터가 없으면 클라이언트는 네임노드에게 메타데이터에 반영해 줄 것을 요청하고 사본이 위치할 데이터노드들의 목록을 얻어온다.
- 얻어온 데이터노드들의 목록에서 자신과 가장 근접한 첫번째 데이터노드로 임시 파일과 나머지 데이터노드들의 리스트를 전송한다.
- 첫번째 데이터노드는 4 킬로바이트 단위로 데이터를 받아 자신의 로컬 파일 시스템에 파일로 저장하면서 곧바로 다음 데이터노드로 데이터를 전송한다.
- 이러한 파이프라인 형태로 임시 파일의 모든 데이터를 목록에 포함된 데이터노드들로 전송하고 모든 데이터노드들이 정상적으로 그 데이터를 저장했는지 확인한다.

- 클라이언트는 모든 데이터노드에 데이터가 정상적으로 저장된 것이 확인되면 네임노드에게 생성 완료료를 알린다.

나. 네임노드

간결하게 설계된 메타데이터 구조를 기반으로 모든 메타데이터를 메모리 상에 유지하는 네임노드는 메타데이터를 자신의 로컬 파일 시스템을 사용하여 두 개의 파일로 저장한다. 첫번째 파일은 네임스페이스, 접근 제어 정보, 파일과 블록들간의 사상 정보, 블록들의 위치 정보와 같은 파일 시스템 메타데이터를 저장하는 FsImage라 불리는 파일이며, 두번째 파일은 FsImage 파일이 기록된 시점 이후에 발생하는 모든 변경 사항을 기록하는 EditLog라 불리는 트랜잭션 로그 파일이다. 네임노드는 기동 시에 메타데이터를 메모리 상에 구축하기 위해 FsImage와 EditLog 파일을 사용하며, 구축이 완료된 후에는 현재 구축된 메타데이터를 FsImage에 저장하고 EditLog 파일에 체크포인트를 기록한다.

사본을 어떤 데이터노드에 배치할 것인가와 클라이언트에게 어떤 데이터노드로부터 데이터를 읽도록 추천할 것인가를 결정하는 정책은 안정성과 성능을 높이는 데 매우 밀접한 관련을 가진다. HDFS는 랙 인지(rack awareness)라 불리는 네트워크 위상 인지를 활용하여 기본적으로 3개의 사본을 유지하

는 상황에서 동일한 랙의 서로 다른 서버에 한 개씩, 다른 랙에 있는 서버에 나머지 한 개를 배치하며, 이러한 배치 정책과 더불어 읽기 요청을 한 클라이언트에게 가장 가까운 데이터노드를 추천한다.

네임노드는 주기적으로 모든 데이터노드들로부터 자신의 상태를 나타내는 heartbeat과 자신이 관리하고 있는 블록들의 목록인 blockreport를 받는다. 네임노드는 heartbeat을 통해 장애가 발생한 데이터노드를 감지하고 그 데이터노드가 포함하고 있는 블록들에 대해 다른 데이터노드에 있는 사본을 활용하여 또 다른 데이터노드로 사본을 복제하는 방법을 통해 안정성을 확보한다. 또한 blockreport와 메타데이터로 유지하고 있는 블록들의 정보를 비교하여 일치하지 않는 블록들을 삭제하도록 데이터노드에게 알려준다.

네임노드는 파일이나 디렉토리에 대한 삭제 요청에 대해 곧바로 삭제하지 않고 임시 디렉토리로 옮겨서 삭제된 것처럼 보이도록 하고 설정된 일정 시간이 지나면 실제로 삭제를 수행한다. 이러한 지연 삭제는 실제로 삭제되지 않은 데이터의 복구를 빠르게 할 수 있으며, 곧바로 데이터를 삭제할 때 발생하는 부하를 줄일 수 있는 장점을 지닌다.

다. 데이터노드

HDFS 파일들은 블록의 단위로 서로 다른 여러 데이터노드들에 분산되어 저장되며, 해당 데이터노드들은 로컬 파일 시스템의 파일로 블록들을 저장하고 관리한다. 데이터노드는 디렉토리 당 최적의 파일 수를 고려하여 서브 디렉토리들을 구성하고 블록에 해당되는 파일들을 적절한 위치에 저장한다. 또한 데이터의 무결성을 보장하기 위해 블록에 해당되는 파일을 저장할 때 그 파일에 대한 체크섬(checksum)을 별도의 숨겨진 파일로 저장한다. 클라이언트가 어떤 블록에 대한 읽기를 요청하면 데이터노드는 블록에 해당되는 파일과 숨겨진 체크섬 파일을 함께 전달한다. 이때 클라이언트는 전달된 체크섬을 활용하여 오류를 검사하고 오류가 있다면 동일한 사본을 가진 다른 데이터노드에게 읽기를 다시 요청한다.

3. Amazon S3 파일 시스템

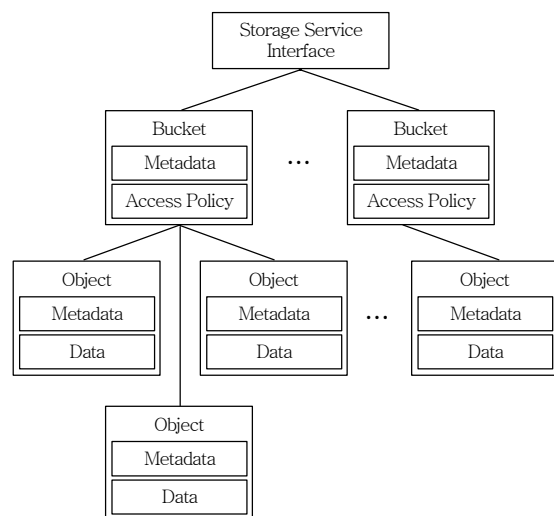
Amazon S3 파일 시스템은 Amazon 웹 서비스에 의해 제공되는 온라인 스토리지 웹 서비스인 Amazon S3를 위한 분산 파일 시스템이다. 현재 Amazon S3 파일 시스템의 구조는 Amazon의 특허를 통해 살펴볼 수 있다[5].

S3 파일 시스템은 5 기가바이트 크기의 객체를 저장할 수 있으며 각 객체 당 최대 2 킬로바이트 크기의 메타데이터를 수반한다. 각 객체는 버킷(bucket)들로 구성되며 각 버킷과 객체는 ACL를 통해 접근이 관리된다.

가. 스토리지 모델

(그림 3)은 웹 서비스를 통해 사용자에게 데이터 스토리지를 제공하기 위한 스토리지 모델을 나타낸 것이다. 스토리지 모델은 사용자의 데이터를 저장하는 객체들과 객체들의 집합인 버킷들로 구성된다. 사용자는 스토리지 서비스를 위해 제공되는 스토리지 서비스 인터페이스를 사용하여 버킷에 접근할 수 있다. 스토리지 서비스 인터페이스는 우리가 일반적으로 사용하는 웹 주소 표현 방식을 그대로 사용한다.

버킷은 메타데이터와 접근 정책으로 이루어진다.



(그림 3) 스토리지 모델

버킷에 포함된 메타데이터는 버킷의 생성 시간, 생성자의 ID, 버킷이 객체들을 포함하고 있는지에 대한 정보, 버킷에 포함된 객체들의 총 크기, 버킷에 대한 사용자의 접근 기록, 버킷과 관련된 청구 기록 등이다. 객체는 메타데이터와 데이터로 이루어진다. 객체에 포함된 메타데이터는 객체의 생성 시간, 객체의 크기, 객체에 저장된 데이터의 타입, 객체의 사용과 이력 정보, 접근 정책 등이다. 각 객체들은 스토리지 서비스 시스템에서 키(key)와 로케이터(locator)에 의해 구별된다. 키는 버킷 단위로 유일한 값을 가지는 반면 로케이터는 모든 버킷에 대해 유일한 값을 갖는다.

나. 구성요소

(그림 4)에서처럼 S3 파일 시스템의 스토리지 클라이언트는 웹 서비스 플랫폼을 통해 서비스를 제공 받는다. 웹 서비스 플랫폼은 다수의 스토리지 서비스 조정자(coordinator)를 통해 서비스를 제공하며 S3 파일 시스템은 이러한 조정자와 노드 선택자(node picker), 복제자(replicator), 키맵 인스턴스(keymap instance), 비트스토어 노드(bitstore node), 복제자 키맵(replicator keymap), DFDD로 구성된다.

키맵 인스턴스는 객체와 로케이터에 대한 사상 정보로 구성되며, 비트스토어 노드는 객체를 저장하

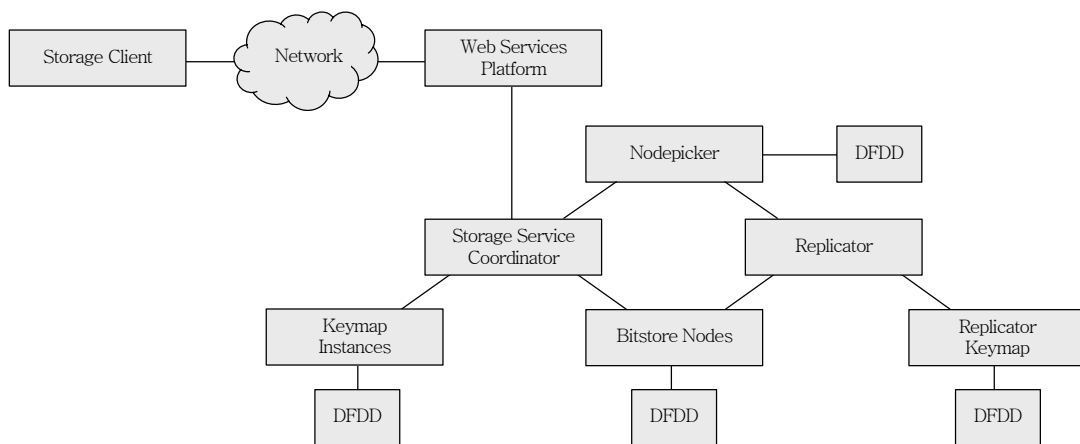
는 역할을 수행한다.

노드 선택자는 조정자와 복제자가 요구하는 객체가 저장될 적절한 비트스토어 노드를 선택하는 역할을 수행하며 다양한 선택 정책들을 포함한다.

복제자는 객체의 복제를 담당한다. 복제자 키맵은 객체와 사본에 대한 사상 정보로 구성된다. DFDD는 조정자, 복제자와 같은 DFDD 클라이언트들에게 비트스토어 노드들과 키맵 인스턴스들의 활동 상태를 제공하기 위해 상태 정보를 검출한다.

조정자는 클라이언트의 웹 서비스 요청에 대해 다양한 버킷 관련 연산들을 처리하며 웹 서비스 플랫폼과 스토리지 서비스 시스템의 다른 구성요소들 사이의 활동을 조정한다. 조정자의 가장 중요한 임무는 객체 데이터와 객체들에 대한 메타데이터의 접근을 처리하는 것이다. 조정자는 객체를 읽을 경우에 키맵 인스턴스에 접근하여 그 객체의 사본이 저장된 비트스토어 노드를 가리키는 로케이터들을 검색하며 객체를 생성하거나 조작하는 경우에 여러 비트스토어 노드에 객체들의 사본들을 저장하고 생성되거나 수정된 사본들의 로케이터를 반영하기 위해 키맵 인스턴스를 갱신한다. 조정자는 생성이나 수정된 객체의 요구되는 사본 수를 모두 생성하지는 않으며 향후 복제자에 의해 비동기적으로 복제가 수행되면서 요구되는 수만큼의 사본이 생성된다.

키맵 인스턴스는 스토리지 서비스 시스템에 저장된 모든 객체들에 대해 키맵 계층 구조를 관리하고



(그림 4) S3 파일 시스템의 구조

색인하기 위해 사용되는 여러 데이터뿐만 아니라 키와 로케이터로 이루어진 키맵 엔트리(entry)들을 포함한 키맵 데이터를 유지한다. 키맵 인스턴스들은 서로 키맵 정보를 교환하며 각 키맵 인스턴스는 서로 통신하는 다수의 호스트들로 구성된다.

키맵 인스턴스 내에서 키맵 데이터는 여러 호스트들에 분산되어 저장된다. 키맵 데이터는 파티션(partition)들로 분할되고 파티션들은 브릭(brick)들로 분할된다. 최종적으로 브릭들은 키맵 엔트리를 포함하는 블록들로 분할된다. 키맵 인스턴스에 포함된 각 호스트는 파티션 색인을 포함하며 파티션 색인은 브릭들을 색인한다. 키맵 인스턴스들에서 파티션들의 복제는 브릭 수준에서 수행된다. 키맵 인스턴스는 키맵 접근 관리, 동기화 방법 또는 프로토콜을 담당하는 키맵 조정자를 포함한다.

4. 병렬 네트워크 파일 시스템

병렬 네트워크 파일 시스템(이하 pNFS)은 클라이언트들이 저장 장치들을 직접적이고 병렬적으로 접근하는 것을 허용하는 NFSv4.1 표준의 한 부분이다[6],[7]. EMC, IBM, NetApp, Panasas와 같은 기업들은 표준에 따라 pNFS를 개발하여 테스트를 진행하고 있다[8]. pNFS의 구조는 (그림 5)에서처럼 데이터 경로로부터 메타데이터 서버를 분리하여 기존의 NFS 서버들이 가지고 있는 확장성과 성능의 문제를 제거했다.

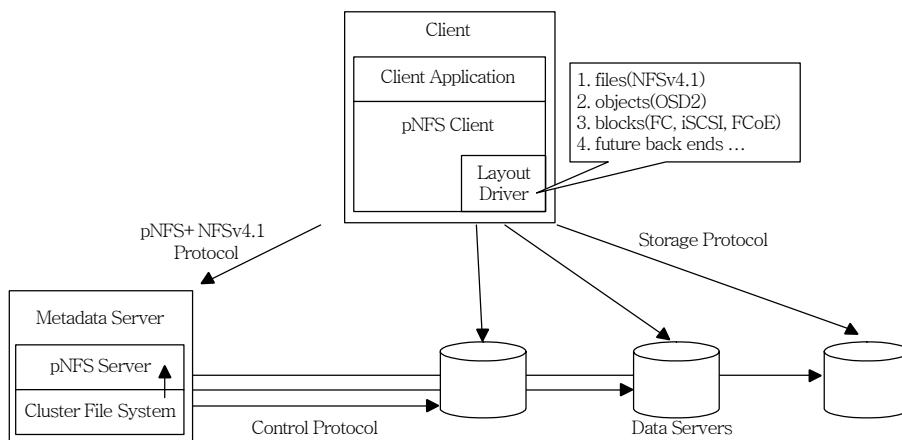
능의 문제를 제거했다.

메타데이터 서버는 네임스페이스, ACL, 속성들과 같은 기본적인 정보뿐만 아니라 다양한 형태의 저장 위치 정보들을 관리한다.

클라이언트는 pNFS 연산들 뿐만 아니라 데이터 서버들과 입출력을 수행하기 위한 최소 하나의 스토리지 프로토콜을 제공한다. 클라이언트는 메타데이터 서버로부터 파일과 데이터 서버들에 대한 사상 정보인 레이아웃(layout)을 얻어온 후 그 레이아웃을 사용하여 데이터 서버들과 직접 입출력을 수행한다. 만일 파일에 대한 변경을 수행한 경우에는 메타데이터 서버에 레이아웃을 전달하여 반영한다.

파일 데이터를 저장하는 데이터 서버는 객체 저장 장치, SAN을 통해 접근되는 블록 장치 등이 될 수도 있다. 스토리지 프로토콜은 클라이언트가 직접 데이터 서버에 접근하여 입출력을 수행하는 데 사용되는 방법이다. 이것은 데이터 서버의 형태에 따라 다양하게 존재한다. 제어 프로토콜(control protocol)은 메타데이터 서버와 데이터 서버를 관리하는 외부 파일 시스템에 의해 사용되며, NFSv4.1 표준의 영역에 포함되지 않는다.

pNFS는 특별한 분산 파일 시스템이라기 보다는 기존의 NFS에서 부족했던 확장성과 성능의 문제를 해결하기 위해 고안한 표준 프로토콜이라고 볼 수 있다.



(그림 5) pNFS의 구조

5. CloudStore

2007년에 Kosmix에 의해 오픈 소스로 공개된 CloudStore는 GFS를 C++로 구현한 것으로 자바로 구현되는 HDFS와 유사하다[9]. GFS와 동일하게 CloudStore는 로그 데이터 저장, map/reduce 데이터와 같은 웹 응용들을 위한 고성능 분산 파일 시스템이다. CloudStore는 리눅스 상에 마운트(mount) 될 수 있도록 FUSE 모듈을 사용하고 GFS는 자체적인 인터페이스를 제공한다는 점에서 약간의 차이가 있다. CloudStore는 현재 지속적으로 개발중이며 HDFS와 통합을 계획하고 있다.

6. GLORY 파일 시스템

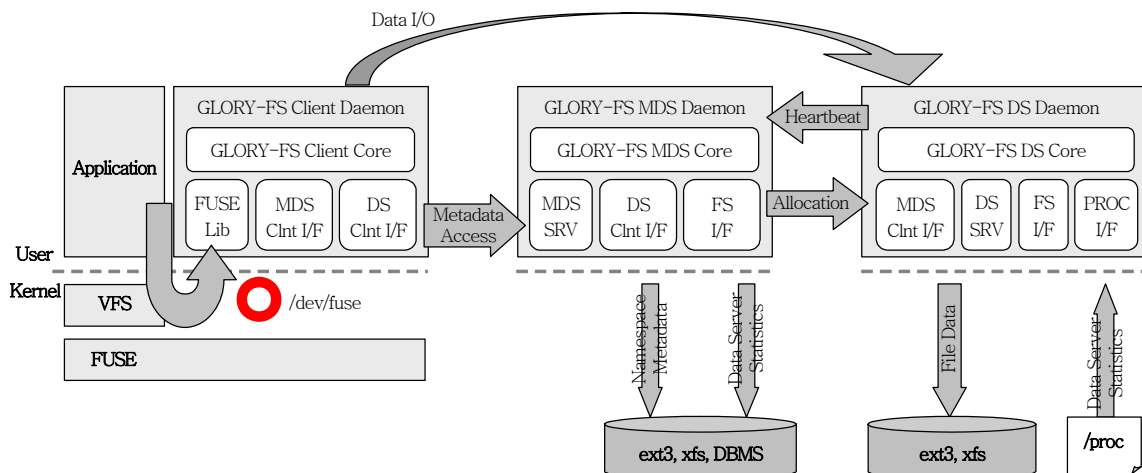
ETRI에서 개발중인 GLORY 파일 시스템(이하 GLORY-FS)은 수천에서 수만 대의 저비용 서버들을 이용하여 저장 공간 구축에 드는 비용을 최소화하면서도 장애에 대한 효율적인 통제 능력과 높은 입출력 처리 성능을 갖춘 대규모 인터넷 서비스를 위한 분산 파일 시스템이다. GLORY-FS는 다음과 같은 특징들을 갖는다.

- 데이터를 저장하는 수많은 서버들의 저장 공간을 하나의 저장 공간인 것처럼 가상화하여 클라이언트에게 제공한다. 이러한 가상화를 통하여

클라이언트에게 엑사바이트 이상의 저장 공간을 제공할 수 있다.

- 서비스의 중단 없이 자유롭게 저장 공간을 관리할 수 있으며, 서버나 디스크를 온라인으로 추가함으로써 간단하게 저장 공간을 확장할 수 있다. 또한 이러한 확장을 통해 부가적인 입출력 성능 향상도 기대할 수 있다.
- 장애가 발생한 서버나 디스크로 인해 서비스가 지연되거나 중단되는 것을 방지하기 위한 장애 감지 및 복구 기능이 제공된다. 자체 치유 기능을 통해 관리자의 개입을 최소화 하였다.
- POSIX에 준하는 파일 시스템 API를 제공함으로써 기존의 응용들을 수정 없이 그대로 사용할 수 있다.
- 서버, 스위치, 랙 등의 네트워크 위상 구조를 인지하고 이를 이용하여 데이터를 최적으로 배치하는 기능을 제공한다.
- 접근이 빈번한 서버나 파일을 감지하여 데이터 재분배나 데이터 사본 증가를 통한 hot spot 회피 기능을 제공한다.

GLORY-FS는 여러 클라이언트에 의해 접근되는 메타데이터 서버(이하 MDS)들의 클러스터와 여러 데이터 서버(이하 DS)들로 구성되며, 저가의 리눅스 서버 상에서 사용자 수준 프로세스로 구동된



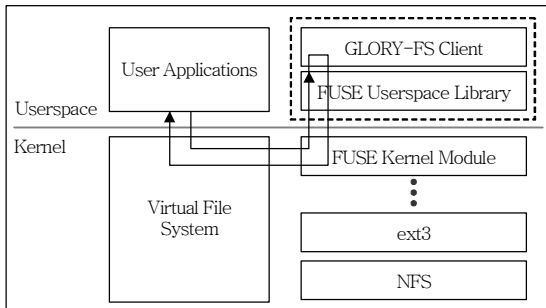
(그림 6) GLORY-FS의 구조

다. GLORY-FS도 (그림 6)에서처럼 GFS나 HDFS와 마찬가지로 비대칭형 구조를 통해 확장성과 고성능을 보장하고 장애에 대한 안정성을 확보한다.

GLORY-FS도 파일을 GFS나 HDFS처럼 일정한 크기의 단위로 쪼개어 여러 데이터 서버들에 분산 저장하며 다수의 사본을 유지함으로써 장애로 인한 데이터의 손실을 방지한다.

가. 클라이언트

GLORY-FS 클라이언트는 (그림 7)에서처럼 FUSE[10]를 기반으로 사용자 수준에서 동작한다. 사용자는 임의의 디렉토리에 GLORY-FS를 마운트하여 로컬 파일 시스템처럼 사용하면 된다. GFS나 HDFS가 자체적인 인터페이스를 통해서만 사용할 수 있는 것에 반해 GLORY-FS 클라이언트는 POSIX에 준하는 파일 시스템 API를 제공함으로써 기존의 응용들을 그대로 수용할 수 있다는 장점을 가진다. GLORY-FS 클라이언트가 동작하는 방식은 GFS와 매우 유사하며 HDFS처럼 임시 파일을 생성하지는 않는다.



(그림 7) FUSE 기반 GLORY-FS 클라이언트

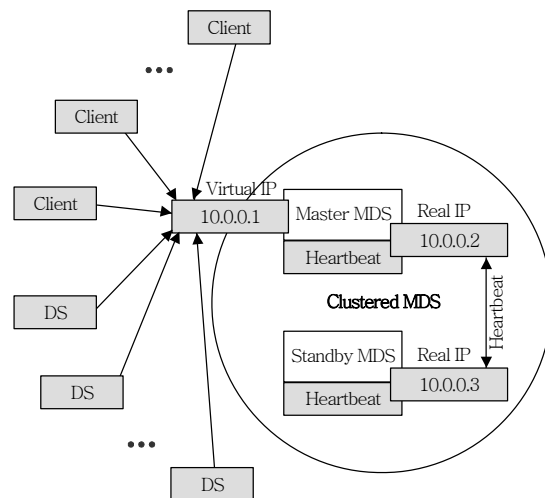
나. 메타데이터 서버

GLORY-FS의 MDS도 GFS와 HDFS의 마스터, 네임노드처럼 GLORY-FS에서 가장 핵심적인 역할을 수행하며 시스템 전반적인 동작들을 제어한다. GFS와 HDFS가 모든 메타데이터를 메모리 상에 유지하고 관리하는 반면 GLORY-FS의 MDS는 메타데이터의 저장과 관리를 위해 DBMS를 활용하며 디

스크 기반 DBMS를 활용할 경우 빈번하게 활용되는 일부 메타데이터만을 메모리 상에 유지한다. DBMS를 활용함으로써 트랜잭션 관리, 안정되고 빠른 접근 처리, 복구를 통한 장애 대처와 같은 여러 가지 이점들을 얻을 수 있으며, 적절한 DB 모델링을 통해 구축된 메타데이터는 외부 응용 프로그램을 통해 손쉽게 재사용될 수 있다.

GLORY-FS의 MDS는 모든 DS들로부터 주기적으로 heartbeat을 받아 DS들의 상태를 파악하고 장애가 감지된 DS에 대해 사본 복제 방법을 통해 안정성을 확보한다. 또한 GLORY-FS에서 관리되는 모든 디스크들은 고유 ID를 가지고 있어서 다른 서버로 디스크를 옮기더라도 MDS가 디스크의 위치 변동을 인식하므로 별도의 처리 과정이 필요 없다.

GLORY-FS는 DBMS의 복제 기능을 활용한 여러 형태의 MDS 클러스터를 제공하며 (그림 8)에서처럼 가장 간단한 active-standby 형태의 MDS 클러스터를 기본적으로 제공한다. GLORY-FS는 이러한 MDS 클러스터를 통해 MDS의 장애에 대처하고 시스템의 안정성을 높인다. Active-standby 형태의 MDS 클러스터에서 active 상태의 마스터(master) MDS는 모든 메타데이터와 관련된 연산을 처리하며, standby 상태의 슬레이브(slave) MDS에게 DBMS의 복제 기능을 통해 지속적으로 메타데이



(그림 8) Active-standby 형태의 MDS 클러스터

터의 동기화를 수행한다. 슬레이브 MDS는 마스터 MDS의 상태를 항상 주시하면서 마스터 MDS에 장애가 발생했을 때 active 상태로 전환하여 마스터 MDS가 재 기동될 때까지 읽기 모드로 동작한다. 클라이언트와 DS는 마스터 MDS의 가상 IP 주소를 통하여 정보들을 교환하며, 마스터 MDS와 슬레이브 MDS는 실제 IP 주소를 통해 정보를 교환한다.

다. 데이터 서버

DS는 데이터의 무결성 보장과 안전한 보관, 데이터 저장 공간의 효율적 관리, 최적의 입출력 성능 보장을 최우선으로 한다. GFS, HDFS와 동일하게 청크는 DS의 로컬 파일 시스템 파일로 저장된다.

DS는 사본 생성을 위해 GFS와 HDFS의 파이프라인 방식과 다르게 지연 복제 방식을 사용한다. 파이프라인 방식에서는 모든 사본들이 안전하게 저장된 후에 클라이언트가 파일 생성을 완료하는 형태인 반면 지연 복제 방식은 첫 사본이 안전하게 저장되면 클라이언트는 파일 생성을 완료하게 되며 별도의 복제 프로세스가 사본들에 대한 복제를 담당하는 형태이다.

DS에서는 청크를 로컬 파일 시스템 파일로 저장할 때 pre-version과 post-version 정보를 함께 저장한다. 두 개의 버전은 초기에 0으로 설정되고 청크에 변경 사항이 발생할 때마다 1씩 증가한다. DS는 모든 변경 사항이 발생할 때마다 pre-version 증가, 변경 사항 갱신, post-version 증가의 3단계를 수행한다. 이러한 버전 정보를 활용하여 파일의 기록 도중에 오류가 발생했는지를 간단하게 판단해 볼 수 있다.

IV. 결론

본 고에서는 최근 급격하게 성장하고 있는 클라우드 컴퓨팅 시장에서 데이터의 저장과 관리뿐만 아니라 상위 계층 서비스가 요구하는 충분한 성능과 안정성을 보장해 주기 위해 분산 파일 시스템이 갖추

어야 할 사항들과 클라우드 컴퓨팅에서 활용 가능하거나 활용되고 있는 대표적인 분산 파일 시스템들을 살펴보았다. 대부분의 클라우드 컴퓨팅에 활용되는 분산 파일 시스템들이 앞에서 살펴보았던 요구 사항들을 만족시키기 위해 거의 유사한 구조와 비슷한 기능들을 갖추고 있는 것을 볼 수 있었다.

클라우드 컴퓨팅에서 분산 파일 시스템은 비용적인 측면에서의 효율성, 지속적으로 증가하는 데이터의 수용, 빈번하게 발생하는 고장에 대한 대처, 관리의 편리성, 신속한 입출력 성능, 데이터 최적 배치, 효과적인 캐시 사용, 부하 집중에 대한 유연한 대처, 데이터에 대한 보안 등과 같이 갖추어야 할 사항들이 무수히 많다. 이러한 사항들이 잘 갖춰진 분산 파일 시스템만이 클라우드 컴퓨팅 시장에서 살아남을 수 있을 것이다.

2007년부터 ETRI에서 개발하고 있는 GLORY 파일 시스템은 클라우드 컴퓨팅에서 분산 파일 시스템이 갖추어야 할 사항들을 잘 갖추고 있으며, 현재 국내 여러 인터넷 서비스 기업들로부터 인정을 받아 실제 서비스를 지원하는 분산 파일 시스템으로 운용되고 있다. 향후 GLORY 파일 시스템은 클라우드 컴퓨팅을 위한 분산 파일 시스템으로 충분히 활용할 수 있는 가능성을 가지고 있다.

약어 정리

ACL	Access Control List
DBMS	Database Management System
DFDD	Discovery and Failure Detection Daemon
DS	Data Server
FUSE	Filesystem in Userspace
GFS	Google File System
GLORY	Global Resource Management System for Future Internet Service
GUI	Graphic User Interface
HDFS	Hadoop Distributed File System
MDS	Metadata Server
NFS	Network File System
pNFS	Parallel Network File System
POSIX	Portable Operating System Interface

S3 Simple Storage Service
SAN Storage Area Network
VFS Virtual File System

참 고 문 헌

- [1] Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing
- [2] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," *In Proc. of ACM Symp. on Operating Systems Principles*, 2003, pp.20-40.
- [3] HDFS, <http://hadoop.apache.org/core/docs>
- [4] Who uses Hadoop, <http://wiki.apache.org/hadoop/PoweredBy>
- [5] "Distributed Storage System with Web Services Client Interface," US Patent No. 2007 0156842A1, 2007.
- [6] <http://www.pnfs.com>
- [7] <http://en.wikipedia.org/wiki/PNFS>
- [8] pNFS BOF, http://www.pnfs.com/docs/sc08_pnfs_bof_slides.pdf
- [9] <http://sourceforge.net/projects/kosmosfs>
- [10] FUSE, <http://fuse.sourceforge.net>