한국컴퓨터정보학회지
제18권 제1호, 2010. 6.

클라우드 컴퓨팅을 위한 클러스터 파일 시스템의 데이터 복제 기법

• 김세희(한양대학교 컴퓨터공학과)
• 김준상(해군사관학교 전산과학과)
• 윤석현(청강문화산업대학 컴퓨터정보과)

Ⅰ. 서론

클라우드 컴퓨팅이란 개인용 컴퓨터나 기업 서버에 개별적으로 저장하던 데이터 및 소프트웨어들을 클러스터로 구축하여 필요할 때 PC나 휴대폰과 같은 각종 단말기를 이용해 원격으로 원하는 작업을 수행할 수 있는 환경을 의미한다[1].

이러한 특징을 가지는 클라우드 컴퓨팅은 2006년 Google의 엔지니어가 처음으로 제안하였으며 유연한 동적 IT 인프라와 QoS가 보장되는 컴퓨팅 환경, 그리고 구성 가능한 소프트웨어 서비스를 제공하기 때문에 많은 관심을 받고 있다. 국외에서는 이미 Amazon, Google, IBM, MS 등의 기업들과 학계에 의해 많은 연구와 개발이 진행되었으며, 국내에서도 한국클라우드컴퓨팅협의회(CCKI)가 출범하고 삼성, LG 등 관련 업계를 중심으로 클라우드 컴퓨팅에 대한 관심이 높다.

클라우드 컴퓨팅의 구조는 대략 그림 1과 같이 클라우드 클러스터와 제공되는 서비스, 그리고 클라이언트의 단말로 구분된다. 서비스 제공자는 서비스에 필요한 데이터를 관리, 저장하는 클라우드 클러스터를 구축하고 사용자들에게 서비스 카탈로그를 제공한다. 사용자들은 네트워크 연결이 가능한 단말기만 있으면 장소와 시간에 제한 받지 않고 이러한 서비스들을 저렴한 비용으로 사용할 수 있다. 또한 사용자는 해당 작업을 처리하는 클라우드 클러스터 내부의 구조나 동작에 대하여 전혀 알 필요가 없이 단순히 작업을 요청만 하면 된다.

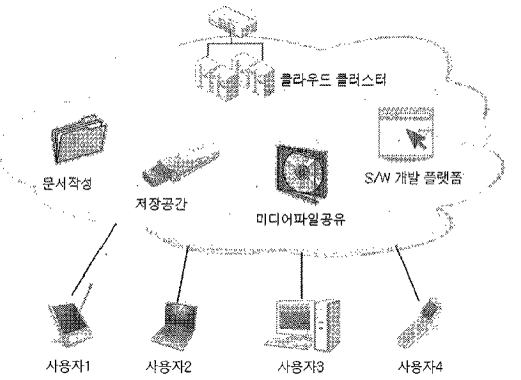

그림 1. 클라우드 컴퓨팅 개념도

이러한 클라우드 컴퓨팅의 클러스터 파일 시스템은 장애 발생 시에도 서비스 중단 및 데이터 손실이 없이 사용자에게 지속적으로 서비스가 이루어질 수 있어야 하며 효율적으로 대용량의 데이터를 관리하고 제공할 수 있어야한다. 따라서 본 논문에서는 현재 클라우드 컴퓨팅을 위해 설계된 대표적인 클러스터 파일 시스템들을 살펴보고 이러한 시스템에서 신뢰성과 가용성을 보장하기 위해 사용하는 데이터 복제 기법들에 대해서 살펴본다.

본 논문의 2장에서는 클라우드 컴퓨팅을 위한 대표적인 클러스터 파일 시스템인 GFS, HDFS와 국내의 OwFS의 설계 특징, 구조 그리고 장애 대책 등을 자세히 소개한다. 3장에서는 이러한 클러스터 파일 시스템에서 서버 장애 발생

시 빠른 복구와 데이터 손실을 막기 위해 사용하고 있는 전체-파일 복제 기법의 장단점을 살펴본다. 또한 이러한 전체-파일 복제 기법의 대안으로 최근 연구되고 있는 Erasure code를 이용한 복제 기법에 대해 소개한다.

Ⅱ. 클러스터 파일 시스템 기술 동향

클라우드 컴퓨팅을 위한 클러스터 파일 시스템은 데이터 입출력 성능을 최대화하기 위해 데이터를 효율적으로 저장 및 관리해야 한다. 또한 복제 정책을 사용하여 데이터 손실을 막고 빈번한 서버 장애에 신속하고 유연하게 대처해야 한다. 본 장에서는 클라우드 컴퓨팅을 위해 개발된 대표적인 국내외 클러스터 파일 시스템들을 소개한다.

1. Google File System(GFS)

Google File System(GFS)은 Google의 대규모 클라우드 서비스를 제공하기 위해 개발된 클러스터 파일 시스템이다 [2]. GFS는 수많은 저비용 서버를 사용하여 클러스터를 구축하였다. 따라서 장애에 대한 내고장성(fault tolerance)을 가지며 동시에 수많은 클라이언트 요청을 처리할 수 있도록 설계되었다.

1.1. 설계 목표

Google은 자사의 응용 작업부하와 기술적인 환경에 특화되도록 GFS를 개발하기 위해 다음과 같은 사항들을 고려하여 설계 및 구현하였다.

- 저비용 서버를 사용하여 시스템 고장이 빈번하게 발생한다.
- 시스템에 저장되는 대부분의 파일들의 용량이 기본 100MB에서 수 GB 정도이므로 대용량 파일들을 효과적으로 관리할 수 있어야 한다.
- 읽기 연산은 기본적으로 많은 데이터를 순차적으로 읽거나 적은 데이터를 랜덤하게 읽는 연산으로 구성된다.
- 또한 많은 순차 데이터를 파일에 추가하는 작업이 많다. 한번 쓰고 나면 파일에 대한 수정은 거의 드물다.
- 시스템은 반드시 동시에 같은 파일에 데이터를 추가하려는 다수의 클라이언트에 대해서도 원자성을 보장하도록 효율적으로 구현되어야 한다.
- 적은 대기 시간보다는 높은 대역폭을 유지하는 것이 더 중요하다.

1.2. 구조

GFS 클러스터는 그림 2와 같이 단일 마스터와 여러 chunk서버들로 이루어지며 다수의 클라이언트들이 접근한다. 파일은 고정된 크기의 chunk들로 나눠지며 각 chunk들은 생성 시 마스터가 할당하는 chunk handle로 구분된다. chunk의 크기는 기본 64MB이며 설정을 통해 변경 할 수 있다. 큰 크기의 chunk를 사용하여 마스터와 클라이언트 간 통신비용을 줄이고 마스터에 저장되는 메타데이터의 크기도 줄였다.

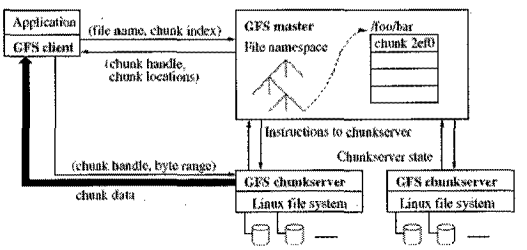

그림 2. GFS 구조

chunk서버는 chunk들을 자신의 로컬디스크에 저장하는데 이때 안정성을 위해 다수의 chunk서버에 복제본 수만큼 복제하여 저장한다. 기본적으로 GFS에서는 3개의 복제본을 저장하고 복제본 수는 사용자가 설정 가능하다. 마스터는 파일 시스템의 네임스페이스, 접근 관리 정보, 파일의 chunk에 대한 매핑 정보 및 chunk의 위치 정보 등에 대한 모든 메타데이터를 관리한다. 이러한 메타데이터는 마스터의 메모리에 저장되기 때문에 연산이 빠르며 주기적으로 마스터가 전체 상태를 점검하는데 효율적이다. 또한 주기적으로 chunk서버와 HeartBeat 메시지로 통신하면서 지시를 내리거나 chunk서버의 상태 정보를 수집한다. 클라이언트는 마스터로부터 메타데이터 정보를 제공받고 이 정보를 이용해서 데이터 관련된 연산은 chunk서버와 직접 통신한다.

1.3. 복제 정책

GFS 클러스터는 약 수백 개의 chunk서버들을 가지며 chunk서버들이 모여 다시 하나의 랙(Rack)을 구성한다. 이러한 chunk서버들은 동일 랙이나 다른 랙의 많은 클라이언트들로부터 요청을 받는데, 이때 다른 랙일 경우 하나 이상의 네트워크 스위치를 통해 통신이 이루어지므로 네트워크 대역폭이 동일 랙 상에서 이루어지는 통신보다 작아지게 된다. 따라서 GFS에서는 랙을 고려한 복제본 배치 정책을 사용하여 네트워크 대역폭 활용률을 높였다. 또한 복제본을 동일 랙 뿐 만 아니라 다른 랙에도 배치시킴으로써 데이터 안정성과 가용성을 향상시켰다.

chunk의 복제본은 생성(creation), 재복제(re-replication), 재분산(rebalancing) 단계에서 생성된다. 생성단계에서 마스터는 chunk를 생성하고 배치할 곳을 결정한다. 이때 디스크 공간 활용이 평균이하인 chunk서버를 선택하므로 전체 chunk서버의 디스크 공간 활용률을 균일화할 수 있다. 또한 위에서 언급한 것과 같이 동일 랙 뿐 만 아니라 다른 랙에도 chunk를 배치한다. 마스터는 또 가용한 chunk 수가 정해진 복제본 수보다 적을 경우 재복제한다. 마지막으로 마스터는 부하분산 및 디스크 공간 활용률을 높이기 위해 복제본들을 주기적으로 재분산한다.

1.4. 장애 대책

GFS 클러스터는 저비용 서버로 구성되므로 장애가 빈번히 발생할 수 있다. 이러한 서버 장애는 전체적인 시스템을 마비시킬 수 있으며 데이터 손실을 야기할 수도 있다. 따라서 GFS는 이러한 장애에 대한 내고장성과 높은 가용성을 보장하기 위해 빠른 복구와 복제 대책들을 구현하였다.

먼저 마스터와 chunk서버들은 그들의 상태를 회복하고 수 초 내에 재시작할 수 있도록 설계되었다. chunk들은 여러 다른 랙에 있는 여러 chunk서버에 복제되어 저장된다. 마스터의 연산 로그와 체크포인트 등의 상태정보는 안정성을 위해 다수의 서버에 복제된다.

2. Hadoop Distributed File System(HDFS)

HDFS는 저비용 서버에서 실행 할 수 있도록 설계된 분산 파일 시스템으로 내고장성을 가지며 응용 프로그램 데이터에 대해서 높은 처리량을 제공한다[3]. HDFS는 Apache의 웹 검색 엔지 프로젝트인 Nutch의 하부 구조로써 만들어졌다가 현재는 Apache lucene 프로젝트의 하부 구조인 오픈 소스 프로젝트 Hadoop의 하부 구조이다. HDFS는 Java로 개발되었으며 GFS와 유사한 특징을 갖는다. 클라이언트-네임노드간의 통신은 TCP/IP 프로토콜을 사용한다. 클라이언트는 네임노드에서 설정한 TCP 포트를 통하여 네임노드와 통신한다.

1.1. 설계 목표

- HDFS의 클러스터의 데이터를 저장하는 서버는 수백 또는 수천으로 구성되므로 각 서버들은 불분명한 실패 확률을 가진다. 따라서 빠른 서버 장애 탐지와 자동적인 복구가 이루어져야한다.
- HDFS에서 수행되는 응용 프로그램들은 스트리밍 데이터 접근을 요구한다. 따라서 데이터 접근 시에 낮은 지연보다는 높은 처리량에 초점을 두고 설계되었다.
- HDFS 응용 프로그램은 파일에 대해 한번 쓰고 여러 번 읽는 접근 모델을 가진다. 따라서 데이터 일관성 이슈를 단순화하고 높은 처리량을 제공한다.

1.2. 구조

HDFS 클러스터는 그림 3과 같이 마스터 서버인 단일 네임노드(Namenode)와 수많은 데이터노드(Datanode)들로 구성된다. 이러한 네임노드와 데이터노드들은 주로 GNU/Linux를 운영체제로 한다.

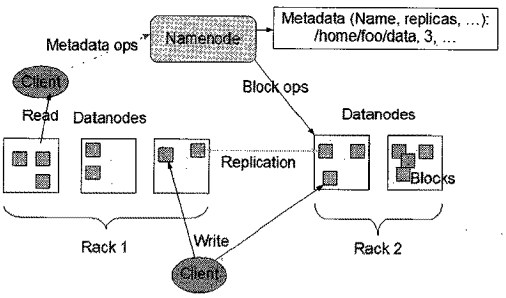

그림 3. HDFS 구조

네임노드는 파일과 디렉토리의 opening, closing, renameing과 같은 파일 시스템 네임스페이스 연산을 수행하고 데이터노드의 블록과 파일의 매핑을 판단한다. 또한 클러스터의 모든 메타데이터에 대한 관리나 제공을 담당한다. 하지만 데이터는 절대 네임노드를 통하지 않도록 설계되어 네임노드의 부담을 줄였다. 각각의 데이터노드는 파일 시스템의 클라이언트들의 읽기 및 쓰기 요청을 제공하며 네임노드의 지시에 따라 블록 생성, 삭제 그리고 복제를 수행한다.

1.3. 복제 정책

HDFS는 대용량 파일을 저장할 수 있도록 설계되었다. HDFS의 파일은 바꿔 쓰기가 안되고(write-once) 사용자 파일을 저장할 때 블록 단위로 나누어서 저장한다. 이때 마지막 블록을 제외한 모든 블록은 같은 크기를 가진다. 기본적으로 블록 크기는 64MB이며 사용자가 설정할 수 있다. 파일의 블록들은 내고장성을 위해 복제되는데 이때 복제본 수는 기본 3개이며 사용자가 설정할 수 있다. 네임노드는 블록의 복제에 관해 모든 결정을 하는데 이때 HeartBeat 메시지와 Blockreport를 클러스터 내의 각 데이터노드로부터 주기적으로 받는다.

HDFS는 데이터 신뢰성과 가용성 및 네트워크 대역폭을 증대시키기 위해 Rack-aware 복제 배치 정책을 사용한다. 일반적으로 HDFS 클러스터는 많은 랙을 거쳐서 형성되며 이때 다른 랙에 있는 두 노드 사이의 통신은 네트워크 스위치를 통해야 한다. 대부분 같은 랙의 노드 간 네트워크 대역폭이 다른 랙의 노드 간 네트워크 대역폭보다 훨씬 크다. 단순히 대역폭 측면에서 본다면 복제본을 모두 동일 랙에 위치시키는 것이 좋다. 하지만 이런 경우 해당 랙의 모든 노드가 실패할 경우 데이터 손실을 초래할 수 있다. 따라서 이러한 문제를 예방하기 위해 HDFS의 복제 배치 정책은 복제본 수가 3개일 때 2개의 복제본은 동일 랙의 서로 다른 두 노드에 위치시키고 1개는 다른 랙에 위치시킨다. 그리고 전체 대역폭 소비와 읽기 지연을 최소화하기 위해 HDFS는 클라이언트로부터 가까운 노드에 위치한 복제본부터 읽기를 수행하도록 설계하였다.

1.4 장애 대책

HDFS는 노드 실패가 발생하더라도 데이터 손실이 없도록 설계되었다. 먼저 각 데이터노드는 네임노드로 Heartbeat 메시지를 주기적으로 보낸다. 네임노드는 이러한 Heartbeat 메시지를 통해서 실패한 데이터노드를 판단하고 실패한 데이터노드에게는 IO요청을 할당하지 않는다. 그리고 데이터노드 실패로 인해 설정된 복제본 수보다 감소된 복제본들을 추적하고 필요 시 복제를 수행한다.

HDFS는 데이터 일관성을 보장하기 위해서 checksum을 사용한다. 클라이언트가 HDFS 파일을 생성할 때 파일의 각 블록의 checksum을 계산하고 이 checksum을 같은 HDFS 네임스페이스 안의 숨겨진 파일로 저장한다. 만약 checksum이 잘못 되었으면 해당 블록의 복제본을 저장하고 있는 다른 데이터노드로부터 읽어서 파일을 복구한다.

HDFS에서는 단일 네임노드를 가지므로 네임노드가 실패할 경우 수동으로 네임노드를 재시작 해야 한다. 아직 자동 재시작이나 네임노드 백업과 관련된 사항은 구현되지 않았다.

3. OwFS(Owner-based File System)

OwFS는 2006년부터 성균관대학교 김진수 교수 연구팀과 NHN이 개발해 온 고성능, 신뢰성, 확장성 및 대용량 처리와 관리 용이성을 갖춘 클러스터 파일 시스템이다. 순수 독자적인 기술로 개발된 OwFS는 2007년 하반기부터 실제 NHN 서비스에 적용되었다[4].

1.1. 설계 목표

초기 OwFS는 메일, 카페, 블로그 등의 네이버 커뮤니티 서비스 적용을 목표로 설계되었다. 이러한 서비스들은 데이터 및 데이터의 메타데이터로 구성되어 있는 특징을 가진다.

- OwFS는 수백 TB에서 수십 PB에 이르는 대용량 저장 공간 지원을 목표로 설계되었다. 또한 저장 장치 구축 비용을 낮추기 위해서 GFS와 같이 다수의 저비용 서버를 활용할 수 있도록 설계되었다.
- OwFS는 저비용 서버를 활용하므로 각 서버의 장애 발생 확률이 높다. 이러한 장애는 커뮤니티 서비스를 사용하는 사용자 데이터 손실을 초래할 수 있다. 따라서 높은 데이터 가용성을 지원하기 위해 OwFS는 GFS와

같이 데이터 복제를 사용한다.

- OwFS는 저장 장치의 관리 편의성을 매우 강조하여 서비스를 중단하지 않고도 저장 장치의 추가, 제거 및 업그레이드가 가능하도록 설계되었다. 또한 관리자의 개입 없이 자동적으로 저장 장치 부하 분산을 실시하는 기능도 제공된다.
- OwFS는 메일 원문, 카페나 블로그의 첨부 파일, 이미지, 음악 및 동영상 파일 등과 같은 불변 파일들을 저장한다. 이때 동영상을 제외한 대부분의 파일들은 크기가 상대적으로 매우 작기 때문에 GFS와는 달리 많은 수의 크기가 작은 파일들을 효율적으로 관리해야한다.

1.2. 구조

OwFS의 구조는 그림 4와 같이 GFS와 유사하나 하나의 메타데이터 서버(MDS)와 다수의 데이터 서버(DS)들로 구성된다.

MDS는 파일의 저장 공간과 관련된 메타데이터를 관리하고 DS 서버의 상태를 감시하여 장애 상황에 대응한다. DS는 파일의 저장을 담당하며 파일에 대한 연산을 수행한다. MDS와 DS는 모두 리눅스 운영체제에서 구현되었다. OwFS 클라이언트 라이브러리는 C와 Java 버전으로 제공되며, 리눅스와 윈도우 환경을 모두 지원한다.

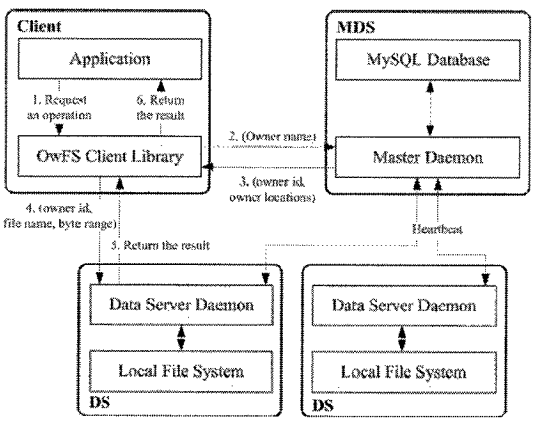

그림 4. OwFS 구조

OwFS가 GFS와 구별되는 가장 큰 차이점 중 하나는 파일 저장 공간에서 서로 관련 있는 파일들을 동일한 owner로 묶어서 관리한다는 점이다. 어떤 파일을 어떤 owner에 저장할 것인지의 여부는 전적으로 OwFS를 사용하는 응용 프로그램에 의해 결정된다.

OwFS에서 owner라는 개념을 사용하는 주된 이유는 OwFS가 다른 파일 시스템들과는 달리 매우 많은 수의 파일들을 다룰 수 있도록 설계되었기 때문이다. 만일 개개의 파일마다 해당 파일의 메타데이터를 MDS에 저장한다면 MDS의 부담이 너무 커지게 되고, MDS가 성능 향상의 장애 요인이 될 수 있다. 반면 OwFS는 응용의 필요에 따라 적절한 수의 owner를 생성하고, 해당 owner에 대한 복제본 위치 정보만을 MDS에 유지함으로써 파일 수의 증가에 따른 MDS의 부담을 경감시킨다.

1.3. 장애 대책 및 복제 정책

OwFS는 각 owner에 대해 기본 3개의 복제본을 유지하며, 모두 동일한 3대의 DS에 저장된다. OwFS에서는 장애 유형을 일시적 장애와 영구적 장애 두 가지로 분류하고 서로 다른 대책을 사용한다. 일시적 장애란 관리자에 의한 시스템 재시작과 같이 일시적으로 DS가 파일 연산 요청에 응답하지 못하는 경우를 의미한다. 이 경우에는 장애 발생 직전의 데이터 접근이 가능하므로, 복제본 복구를 통하여 업데이트된 파일의 일관성을 유지시킨다. 반면 영구적 장애는 파일 시스템 오류 등과 같이 해당 DS가 서비스를 지속할 수 없는 심각한 상황을 의미하며, 이 경우에는 장애 발생 직전의 데이터를 이용할 수 없는 경우가 대부분이므로 다른 DS에 새로운 복제본을 생성한다.

OwFS 초기 버전은 하나의 owner가 생성될 때, 해당 owner의 복제본을 디스크 사용률과 워크로드가 낮은 3개의 DS 서버에 할당하였다. 이 경우 DS가 3대 이상 장애 발생 시 일부 owner에 대해서는 데이터 가용성을 보장할 수 없다. 이러한 문제점을 보완하기 위해 장애 발생 시 접근할 수 없는 DS 서버들을 그룹별로 나누어 DS 그룹 당 1개의 owner 복제본을 가지도록 복제본 배치 정책을 변경하였다. DS 서버 장애가 동시에 DS 그룹 3개 이상에 걸치지 않는다면, DS 서버 장애 수와 관계없이 적어도 1개의 정상 복제본이 존재하므로 데이터 가용성이 향상된다.

DS 그룹별 복제본 배치 정책은 장애에 대한 데이터가용성을 높이지만, DS 서버의 영구적 장애로 인한 복제본 이동

의 성능은 다른 DS 그룹 내의 복제본을 읽어 와야 하기 때문에 저하된다.

Ⅲ. 클러스터 파일 시스템의 데이터 복제 기법

앞에서 GFS와 HDFS 그리고 OwFS에 대해서 살펴보았다. 이와 같은 클러스터 파일 시스템에서는 다수의 저비용 서버들을 사용하므로 서버 장애가 빈번하게 발생할 수 있다는 공통점을 가진다. 그리고 이러한 시스템 장애 시 데이터 손실을 방지하고 높은 가용성을 보장하기 위해서 데이터 복제 기법을 사용한다. 이 장에서는 데이터 복제 기법들에 대해서 알아본다.

1. 전체-파일 복제(Whole-file Replicaion)

전체-파일 복제 기법은 앞에서 살펴본 GFS와 HDFS 등의 클러스터 파일 시스템에서 가장 많이 쓰이는 복제 기법이다. 이 기법은 원본 파일에 대해서 정해진 수만큼의 복제본을 추가로 저장하므로 구현이 간단하다. 이때 복제본 배치 정책에 따라서 시스템에 분산 저장하므로 다중 데이터 읽기 성능을 높일 수 있다. 또한 기본적으로 3개의 복제본을 저장하며 복제본 수가 커질수록 높은 가용성을 보장한다. 하지만 가용성을 위해 데이터를 중복 저장하므로 더 많은 저장 공간 오버헤드를 요구한다는 단점이 있다[5]. 따라서 이러한 저장 공간 오버헤드는 시스템 구축비용의 증가로 이어진다.

2. Erasure code 복제 기법

Erasure code는 binary erasure channel에서의 오류 정정을 위해 개발되었으며 P2P와 같은 WAN 기반의 분산 파일 시스템에서는 복제 기법으로도 많이 사용되고 있다[5].

Erasure code 복제 기법은 원본 파일을 동일한 크기의 n개 데이터 블록으로 나누고 이를 가지고 인코딩을 통해 m개 코딩 블록을 생성하고 (n+m)개의 블록을 저장한다. 이렇게 저장된 (n+m)개 블록 중 어떤 n개 블록만 다운로드하면 원본 파일을 복구할 수 있게 된다. 이때 저장 공간 오버헤드는 (n+m)/n이며 항상 n개의 데이터 블록만 다운로드한다. 따라서 Erasure code 복제 기법은 전체-파일 복제 기법보다 더 적은 저장 공간 오버헤드를 가지면서 동시에 비슷하거나 더 나은 수준의 가용성을 보장하는 이점이 있다.

이러한 Erasure code 복제 기법에는 RS code와 LDPC code가 있다.

1.1. RS(Reed-Solomon) code

RS code는 Erasure code의 하나로 클라이언트는 (n+m)개 블록 중에서 어떤 블록이든지 n개의 블록만을 다운로드하면 원본 파일을 복구할 수 있게 된다. 하지만 n과 m이 커질수록 인코딩 및 디코딩 비용이 급격히 증가한다는 단점이 있다[6].

1.2. LDPC(Low-Density Parity-Check) code

LDPC code는 Erasure code의 하나로 원본 파일을 n개의 데이터 블록으로 나누고 이를 이용해서 인코딩을 통해 m개의 코딩 블록을 생성한다[7][8]. (n, m) LDPC 코드는 그림 5와 같이 n개의 왼쪽 노드와 m개의 오른쪽 노드를 가지는 Tanner 그래프로도 정의할 수 있다. 이때 각 m개의 오른쪽 노드는 각각에 연결된 왼쪽 노드들의 XOR 연산으로 생성한다. LDPC code는 인코딩과 디코딩 시에 상대적으로 간단한 XOR 연산을 사용하므로 RS code보다 계산 비용이 적으며 속도가 빠르다는 장점을 가진다. 하지만 원본 파일 복구를 위해서는 평균 n개 이상의 블록을 다운로드 받아야 한다.

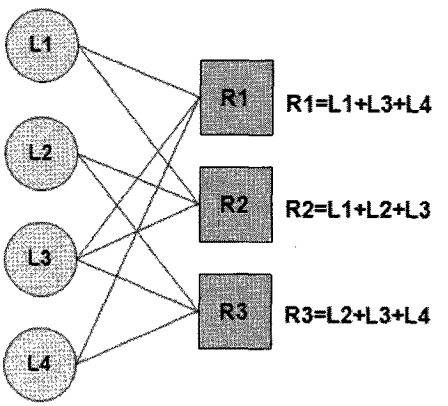

그림 5. (4,3) LDPC code의 Tanner 그래프

LDPC code 복제 기법의 성능 평가 척도로는 overhead factor(f)가 쓰인다. f는 원본 파일 복구를 위해 다운로드 해야 하는 평균 블록 수(overhead o)와 데이터 블록 수(n)의 비율 즉, f=o/n으로 계산된다. LDPC code는 원본 파일 복구를 위해 평균 n개 이상의 블록을 다운로드 받아야 하므로 f가 항상 1보다 크다. 따라서 특정 n과 m에 대해서 f가 1에 가까울수록 성능이 좋다고 볼 수 있다. 예를 들어 그래프에서 왼쪽의 n개 데이터 블록들이 모두 접근 가능한 경우에는 n개 데이터 블록들을 모두 다운로드하고 별다른 디코딩 과정 없이 단순히 순서대로 연결만 하면 원본 파일을 복구 할 수 있다. 이때 n개만 다운로드하므로 f는 1이다. 하지만 n개 중 1개 이상의 데이터 블록이 접근 불가능하면 나머지 접근 가능한 모든 데이터 블록을 다운로드받고 추가로 오른쪽 코딩 블록들을 다운로드 받아야 원본 파일을 복구할 수 있다. 이때 f는 1보다 커지게 된다.

LDPC code는 점진적으로 n이 커질수록 f가 1에 가까워지는 특성을 가진다. 하지만 실제 클러스터 파일 시스템에서는 원본 파일을 아주 작은 블록들로 계속 나눌 수 없기 때문에 n과 m을 무한대로 크게 할 수 없다. 따라서 n과 m이 작을 때(n, m < 10,000) 최적의 f를 찾는 것이 중요한 이슈 중 하나이다.

Ⅳ. 결론

본 논문에서는 클라우드 컴퓨팅을 위해 설계, 구현된 대표적인 클러스터 파일 시스템인 GFS, HDFS와 OwFS들의 구조와 특징을 살펴보았다. 클라우드 컴퓨팅을 위해 클러스터 파일 시스템에서는 다음과 같은 사항이 요구된다. 먼저 대용량 데이터를 효율적으로 관리하고 제공해야하며 높은 데이터 입출력 성능을 제공해야 한다. 이를 지원하기 위해 세 가지 클러스터 파일 시스템에서는 모두 다수의 저비용 서버들을 사용하여 데이터 복제본을 여러 서버에 분산 저장한다. 하지만 저비용 서버들은 장애가 빈번하게 발생하기 때문에 클라우드 컴퓨팅의 중요한 요구 중 하나인 신뢰성을 떨어뜨릴 수 있다. 따라서 신뢰성을 향상시키기 위해 데이터 복제 정책과 장애 대책을 사용하여 서버 장애 발생 시에도 시스템을 빠르게 복구하고 데이터 손실도 방지하였다.

클라우드 컴퓨팅에서는 대용량의 파일들을 다루기 때문에 데이터 입출력 성능 및 신뢰성을 위해 데이터 복제본을 많이 생성하면 할수록 그만큼 클러스터의 저장 공간 오버헤드가 증가한다. 따라서 전체적인 클러스터 구축비용이 증가하게 되는 문제가 발생한다. 이를 위해 본 논문에서는 최근 연구되고 있는 Erasure code 복제 기법에 대해서 살펴보았다. Erasure code 복제 기법은 전체-파일 복제 기법보다 적은 저장 공간 오버헤드를 가지면서 동시에 비슷한 수준의 가용성을 제공한다. 하지만 아직까지 이러한 연구가 활발히 이루어지지 않기 때문에 해결해야 될 이슈들이 많이 남아있다.

참고문헌

[1] 이종숙, 박형우, "국내외 클라우드 컴퓨팅 동향 및 전망," 정보처리학회지, 제 16권, 제 2호, 17-30쪽, 2009년 3월.

[2] S. Ghemawat, H. Gobioff, S.T. Leung, "The Google file system", ACM SIGOPS Operating Systems Review, Vol. 37, No. 5, pp. 29-43, Dec. 2003.

[3] Hadoop, *http://hadoop.apache.org,*

[4] 김진수, 김태웅, "OwFS: 대규모 인터넷 서비스를 위한 분산 파일 시스템", 한국정보과학회지, 제 27권, 제 5호, 77-85쪽, 2009년 5월.

[5] W.K. Lin, D.M. Chiu, Y.B. Lee, "Erasure code replication revisited", IEEE, In proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P 2004), pp. 90-97, Aug. 2004.

[6] J. S. Plank and M. G. Thomason, "A practical analysis of low-density parity-check erasure codes for wide-area storage applications", IEEE, In International Conference on Dependable Systems and Networks, pp. 115-124, Jun. 2004.

[7] B. Gaidioz, B. Koblitz, N. Santos, "Exploring high performance distributed file storage using LDPC codes", Parallel Computing, Vol. 33, No. 4-5, pp. 264-274, Feb. 2007.

[8] X. Li, C. Xie, Q. Wei, and Q. Cao, "A Reliable Scheme for Cluster Storage System", IEEE, In proceedings of 3th International Conference on Semantics, Knowledge and Grid, pp. 394-397, Oct. 2007.

저 자 소 개

김 세 희

2009: 한양대학교
전자컴퓨터공학부 학사.
현 재: 한양대학교
컴퓨터공학과 석사과정.
관심분야: 클라우드컴퓨팅,
분산파일시스템.

윤 석 현

인하대학교 전자공학과 (공학사).
연세대학교 전자공학과 (공학석사).
국민대학교 전자공학과 (공학박사).
1981년~: 동양공업전문대학 교수.
1996년~현재: 청강문화산업대학
컴퓨터정보과
교수.
관심분야: 응용소프트웨어,
멀티미디어 서비스,
클라우드컴퓨팅.

김 준 상

2003: 한양대학교
전자컴퓨터공학부 학사.
2005: 한양대학교
컴퓨터공학과 석사.
2008: 한양대학교
컴퓨터공학과 박사 수료.
현 재: 해군사관학교
전산과학과 전임강사.
관심분야: Grid 데이터베이스,
클라우드컴퓨팅.