특집 04

클라우드 컴퓨팅을 위한 분산 데이터 관리 시스템 및 데이터 서비스 기술

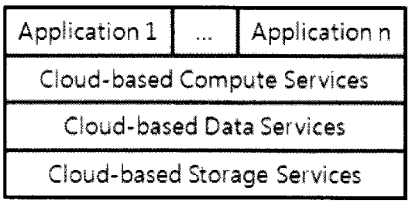
목 차

문상철 · 김형준
(NHN(주))

1. 서 론

클라우드 컴퓨팅은 관점에 따라 다양하게 정의할 수 있는데, 인프라스트럭쳐의 관점에서 바라본 클라우드 컴퓨팅은 높은 확장성(Scalability)을 갖는 추상화된 컴퓨팅 인프라스트럭쳐의 집합을 의미한다. 그리고, 클라우드 컴퓨팅 서비스는 이러한 확장성을 서비스로써 제공하는 것을 목표로 한다. 이러한 클라우드 컴퓨팅 서비스상에서 개발된 애플리케이션은 계산 수행 성능의 확장이나 저장소의 확장을 애플리케이션 내부에서 처리할 필요 없이, 클라우드 내부의 자원을 더 사용함으로써 쉽게 애플리케이션의 규모를 확장할 수 있다.

클라우드 컴퓨팅 서비스가 이러한 확장성을 제공하기 위해서는 서비스를 구성하는 각 계층 역시 쉽게 확장 가능하도록 구성해야 한다. 클라우드 컴퓨팅 서비스의 인프라스트럭쳐는 (그림 1)에서와 같이 크게 스토리지 계층, 데이터 서비스 계층, 컴퓨팅 서비스 계층 등으로 구분할 수 있으며, 컴퓨팅 서비스 계층에서는 가상화 기술, 스토리지 계층에서는 분산 파일시스템 기술을 이용한다. 본 기고에서는 데이터 서비스 계층을 구성하기 위한 기술인 분산 데이터 관리 시스템 기술들의 동향과 클라우드 컴퓨팅 기술로써 확장성과 데이터 관리 시스템의 중요한 목표인 고가용성을 위한 설계에 대해 알아본다.



(그림 1) 클라우드 컴퓨팅 인프라 구성

수십 년간 데이터 저장 기술로써 확고한 지위를 유지하고 있는 관계형 데이터 관리 시스템은 ACID(Atomic, Consistency, Isolation, Durability)[1] 속성에 집중하여, 데이터를 안전하게 저장하고 정합성을 보장하는데 주 목적을 두고 있다. 그러나, 최근 인터넷과 하드웨어의 발전으로 인해 검색 서비스, 데이터웨어하우징, 과학 계산과 같은 다양한 서비스들이 출현하게 되었고, 이러한 서비스들은 강한 수준의 정합성

이나 견고성보다는 손 쉬운 확장성, 시스템 확장 과정이나 장애 상황에서도 서비스를 유지할 수 있는 고 가용성, 그리고 낮은 비용을 요구한다. 이러한 속성을 BASE(Basically Available, Soft-state, Eventually consistent)[2][3]로 표현하며, BASE 속성에 맞는 새로운 데이터 관리 시스템이 출현하게 되었다[4][5]. 결국, 확장성과 고 가용성을 제공해야 하는 클라우드 컴퓨팅의 데이터 서비스 계층은 새로운 데이터 관리 시스템인 분산 데이터 관리 시스템을 사용한다.

본 문서에서는 최근 발표되고 있는 분산 데이터 관리 시스템의 데이터 분산 및 확장 방법과 정합성(Consistency) 혹은 가용성(Availability)을 어떻게 보장하고 있는지에 대해 살펴보고 클라우드 데이터 서비스에서는 이런 속성들이 어떻게 서비스되고 있는지 살펴본다. 2장에서는 대규모의 확장성과 고 가용성이 맞춰 개발된 Bigtable[4], Dynamo[5] 등과 같은 분산 데이터 관리 시스템에 대해 살펴보며, 3장에서는 이를 이용한 SimpleDB[6], SQL Data Service[7], AppEngine Datastore[8] 등과 같은 클라우드 컴퓨팅 환경의 데이터 서비스에 대해 살펴본다.

2. 분산 데이터 관리 시스템

분산 데이터 관리 시스템은 대규모의 구조화된 데이터를 분산 환경에 저장하고 저장된 데이터의 질의를 서비스하는 시스템으로 최근 클라우드 컴퓨팅이 소개되면서 클라우드 데이터 서비스를 위한 기술로 활용되면서 다양한 연구와 개발이 진행되고 있다. 분산 환경에서 시스템을 구성할 경우 다음과 같은 세가지 속성을 고려한다[9].

- 정합성(Consistency): 모든 클라이언트는 항상 동일한 데이터를 보장 받는 속성
- 가용성(Availability): 분산 시스템의 가용성은 네트워크 단절 상황에서도 장애가 발생하지 않은 노드는 모든 요청에 대해 정해진 시간 내에 응답을 해야 하는 속성.
- Partition Tolerance: 네트워크가 단절된 상태에서도 시스템의 속성(정합성 또는 가용성)을 유지해야 속성. 시스템이 Partition Tolerance 한 속성을 갖기 위해서는 노드 사이에 전송되는 메시지가 전달되지 않는 상황에서도 시스템에 제공하고자 하는 정합성 또는 가용성을 지원해야 한다.

이러한 속성을 약어로 CAP 속성이라고 하며 세가지 속성을 모두 만족시키는 분산 시스템을 구성하는 것은 어렵기 때문에 대부분의 분산 시스템은 두 가지 속성만을 지원한다[10]. 관계형 데이터 관리 시스템은 정합성과 가용성에 초점이 맞춰져 있는 반면, 분산 데이터 관리 시스템들은 가용성 또는 정합성과 Partition-Tolerance를 특성을 주로 제공한다.

2.1 Bigtable

Bigtable[4]은 구글에서 개발한 분산 데이터 관리 시스템으로 수백 내지 수 천대의 값싼 하드웨어 장비를 이용하여 페타(Peta) 바이트 이상의 구조화된 데이터(semi structured data)를 저장할 수 있다. Bigtable은 범용성, 확장성, 고성능, 고 가용성의 목표를 가지고 만들어진 시스템이다.

Bigtable의 데이터 모델은 Spare distributed persistent multidimensional sorted map이다. 모든 데이터는 row key, column key, timestamp로 정렬되어 있으며 value에는 byte 배열을 저장한다. 데이터 모델을 구성하는 주요 엘리멘트는 Row, Column Family, Timestamp 등이 있다. 하나의 테이블에 저장된 데이터는 row key로 유일하게 식별되며 read/write 연산은 row 단위로 atomic 하게 처리된다. Bigtable은 하나의 테이블을 row key 범위를 이용하여 파티셔닝하며 파티셔닝 된 단위를 Tablet이라고 부른다. 하나의 Tablet은 특정 노드에 의해 서비스된다. 파티셔닝 범위, 서비스 노드 등과 같은 파티셔닝에 대

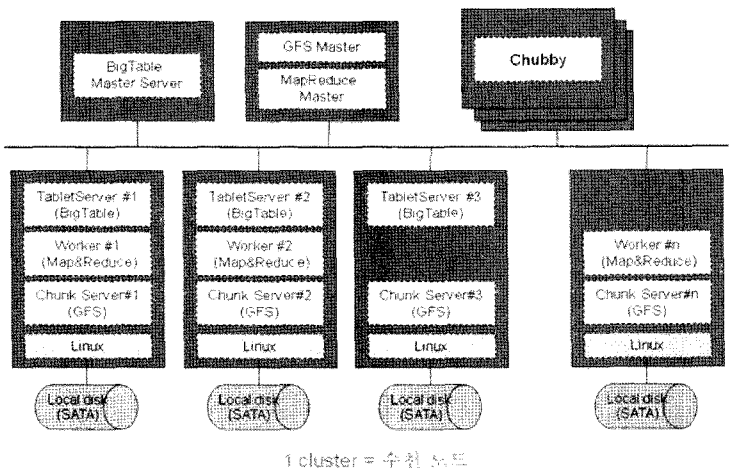
한 정보는 하나의 ROOT Tablet과 다수의 METADATA Tablet에 저장한다. ROOT Tablet의 하나의 row에는 하나의 METADATA Tablet에 대한 정보(Tablet명, max row key, 서버 정보)를 저장한다. METADATA Tablet의 하나의 row에는 사용자 테이블의 하나의 Tablet에 대한 정보를 저장한다. 특정 row key를 서비스하는 사용자 테이블의 Tablet과 Tablet Server를 찾기 위해 Chubby → ROOT Tablet → METADATA Tablet 순으로 찾는다.

Bigtable은 하나의 마스터 서버와 다수의 Tablet 서버로 구성되며, 마스터 서버는 메타 정보와 같이 클러스터 관리에 필요한 정보를 가지고 있기 않기 때문에 마스터 장애 시에도 데이터 서비스는 영향을 받지 않는다. 마스터 서버는 Tablet을 할당하거나, Tablet 서버가 클러스터에 추가/제거되는 것을 감지하고, Tablet 서버의 부하분산과 GFS(Google File System)[11]에 저장된 파일에 대한 가비지 컬렉션을 수행한다. Tablet 서버는 Tablet을 관리하고 클라이언트로부터 데이터 read/write 요청을 받아 처리한다. 하나의 Tablet 서버는 10~1000개의 Tablet을 서비스하고 하나의 Tablet의 크기는 100~200MB 이다.

Bigtable은 GFS, MapReduce[12], Chubby [13] 등과 같은 구글 내부의 여러 분산 플랫폼을 이용한다. Bigtable은 GFS를 데이터 파일 또는 커밋 로그 저장용으로 사용한다. GFS에서 하나의 파일은 3개의 복제 본을 가지고 있기 때문에 추가적인 백업이 필요 없으며, 수천 노드 이상으로 확장할 수 있는 확장성과 복제 본 간의 정합성을 제공한다.

GFS는 파일의 Random write 기능을 제공하지 않기 때문에 이미 저장된 파일을 수정하는 것이 불가능하다. 파일시스템의 이런 제약 때문에 Bigtable은 In-Memory, On-Disk 데이터 관리 시스템의 속성을 모두 가지고 있다. Bigtable의 write 연산은 데이터 파일을 직접 수정하지 않고 메모리에만 write 내용을 기록한다. 메모리가 일정 크기에 도달하면 메모리의 내용을 파일시스템으로 저장한다. Write 연산을 메모리에만 저장하기 때문에 Tablet 서버에 장애가 발생할 경우 데이터 복구를 위해 모든 write 연산 처리 시 GFS에 커밋 로그를 저장한 후 메모리에 저장 한다.

Bigtable에 저장된 데이터에 대해 대규모의 분석 작업이 필요한 경우 MapReduce 플랫폼을 이용하며, 분산 처리되는 단위는 하나의 Tablet이



(그림 2) Bigtable 클러스터 구성

된다. Chubby는 분산 락 서비스를 제공하는 시스템으로 Bigtable에서는 ROOT 메타 정보를 서비스하는 Tablet Server의 위치 정보, 테이블의 스키마 정보를 저장하거나, 여러 마스터 서버가 동시에 실행되어 있을 때 유효한 마스터 서버를 선출하거나, Tablet Server 장애 상황 발생을 감지하는 등에 사용한다. 하나의 Chubby 셀은 5개의 노드로 구성되며 노드간에 모든 정보는 동기화되어 마스터 정보에 대한 SPOF(Single Point Of Failure) 지점을 없애 주는 역할을 수행한다.

Bigtable은 GFS를 이용함으로써 데이터의 정합성은 보장하지만 네트워크 단절 상황이나 일부 노드 장애 상황에서는 고 가용성을 지원하지 않는다. 장애가 발생한 Tablet 서버에서 서비스되던 Tablet이 다른 Tablet 서버로 할당 되기 전에는 특정 row key의 영역은 서비스가 되지 않기 때문이다.

Bigtable은 Web indexing, Google Earth, Google Finance, Google Analytics, Personalized Search 등 이미 구글의 많은 서비스에 적용되어 있다. 최근에는 Bigtable에 새로운 기능을 추가하여 기존의 Bigtable 클러스터 보다 더 큰 클러스터를 구성하는 Megastore[14]를 구축하였다. Bigtable은 논문으로만 발표되었으며 소스는 공개되지 않았다. Apache foundation나 인터넷 서비스 업체들이 Bigtable의 개념을 도입한 시스템을 발표하고 있으며 이들 대부분은 공개 소프트웨어이다.

Neptune[15]은 국내 포털 업체인 NHN에서 개발한 분산 데이터 관리 시스템으로 데이터모델, 아키텍처, 기능 등에서 Bigtable의 개념을 많이 도입하였다. 다양한 분산 파일시스템을 지원하며 Chubby 역할을 수행하는 Pleiades[18]라고 하는 분산 락 서비스를 내장하고 있다. 커밋 로그 처리를 위해 커밋 로그 전용 파일시스템을 제공하여 HDFS와 같이 분산 파일시스템에서 커밋 로그 관련 기능을 제공하지 않는 경우에도 내구성(durability)을 지원한다.

HBase[16]는 Bigtable과 동일한 기능과 확장성을 가지는 분산 데이터 관리 시스템을 목적으로 하는 Apache foundation의 오픈 소스 프로젝트이다. GFS와 MapReduce를 구현하는 Apache 프로젝트인 Hadoop[17] 프로젝트의 서브프로젝트이다. HBase는 Bigtable의 아키텍처를 그대로 사용하고 있지만 Chubby와 같은 분산 락 서비스와 연동되어 있지 않으며 하나의 Master 서버가 클러스터 정보를 관리한다. 따라서 Master 서버에 장애가 발생할 경우 전체 클러스터가 장애 상황에 놓이게 되어, Master 서버가 SPOF(Single Point Of Failure)가 된다. HBase는 데이터 파일 저장소로 HDFS(Hadoop Distributed File System)를 사용한다. HDFS에 저장된 파일은 불변(immutable) 파일로 한번 생성된 후 저장된 파일에 대해서는 추가하거나 변경할 수 없다. 파일시스템의 이런 제약 때문에 HBase의 커밋 로그는 변경 연산 수행 즉시 바로 파일시스템에 저장되는 것이 아니라 주기적으로 파일시스템에 저장된다. 따라서 커밋 로그가 저장되기 이전에 노드에 장애가 발생한 경우 데이터 유실이 발생할 수 있다. HBase는 CAP 속성에서는 Bigtable과 유사하지만 SPOF가 존재하기 때문에 가용성 측면에서 Bigtable과 비교하여 좋지 않으며, 데이터 유실의 가능성도 있어 내구성(durability)도 완벽하게 지원하지 않는다.

Hypertable[19]은 Bigtable의 개념을 도입한 분산 데이터관리시스템으로 Zvents라는 검색 엔진 회사에서 개발하여 소스를 공개하였다. 최근 중국 검색 서비스 회사인 Baidu에서 개발과 적용에 적극적으로 참여하고 있다. HDFS, KFS[20] 등과 같은 여러 종류의 분산파일시스템을 지원한다. Hyperspace라고 하는 분산 락 서비스를 제공하지만 하나의 서버에서만 수행되기 때문에 장애가 발생할 경우 전체 클러스터에 영향을 준다. Hypertable도 커밋 로그를

DFS에 저장하기 때문에HBase와 동일한 문제를 가지고 있다.

2.2 Dynamo

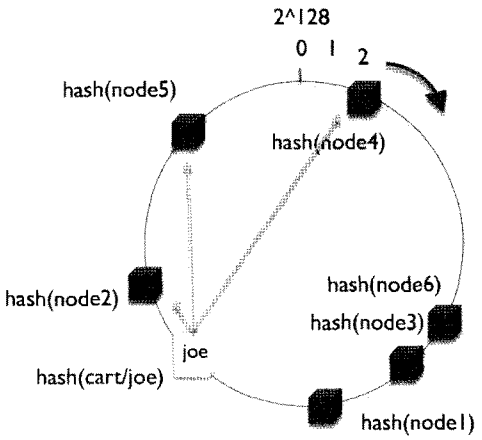
Dynamo[5]는 아마존에서 개발한 key, value 기반의 분산 데이터 관리 시스템이다. 아마존 내에는 여러 종류의 스토리지 솔루션이 존재하며 Dynamo는 쇼핑 카드, 사용자 정보, 제품 카탈로그 등과 같은 key, value 기반의 고 가용성이 필요한 업무에 주로 사용한다.

Dynamo는 고 가용성을 위해 데이터를 여러 노드에 복제하고 write 연산에서는 복제 본 간의 정합성을 보장하지 않고 read 연산 처리시 복제 본 간의 버전 충돌을 처리한다. 이런 특징 이외에 Dynamo는 다음과 같은 설계 원칙을 가지고 있다.

- Incremental scalability: 한번에 하나의 스토리지 노드를 추가하며 노드 추가 시 시스템에서 제공하는 기능 또는 시스템 자체에 최소한의 영향만 미치도록 한다
- Symmetry: 단순한 시스템 구성과 관리의 편의를 위해 클러스터의 모든 노드는 동일한 역할을 수행한다.
- Decentralization: 중앙 집중적으로 관리되는 데이터나 이를 관리하는 서버는 존재하지 않는다.
- Heterogeneity: 사양이 다른 서버들로 클러스터를 구성할 수 있다.

Dynamo는 데이터를 분산 배치하기 위해 P2P(Peer-to-Peer) 분산 기법[21]을 이용한다. 초기의 P2P 기술에서는 비 구조화된(unstructured) 네트워크 구성을 사용함으로써 네트워크 오버헤드 또는 중앙집중 관리 방식에서의 장애 상황 등과 같은 단점이 있었다. 최근에 사용하는 P2P 기법은 structured P2P라고 하여 데이터의 hash(k)와 클러스터에 참여하는 노드의 hash(node id)의 값을 동일한 주소공간으로 매핑하여 관리하는 DHT(Distributed Hash Table)[21] 알고리즘을 사용한다.

Dynamo는 DHT 기법을 이용하여 데이터를 파티셔닝 한다. DHT를 이용함으로써 노드의 추가/삭제 시 key에 대해서 일관된 매핑 결과를 보장 받을 수 있으며, 매핑 정보도 부분적으로만 변경되기 때문에 클러스터에 부하를 주지 않는다.



(그림 3) Dynamo 클러스터 구성

Dynamo는 데이터의 신뢰성을 위해 데이터를 여러 노드에 복제한다. 데이터의 복제는 클러스터에서 key에 매핑되는 노드와 매핑된 노드 다음에 위치하는 노드에서 복제 본 수 - 1개의 노드에 저장한다. 데이터 복제는 낙관적(optimistic) 정책을 사용한다. 낙관적 복제 정책은 복제 본의 저장이 완료되지 않더라도 write 연산을 정상적으로 처리하고 백그라운드 작업을 통해 복제 본을 유지시키는 정책이다. 이렇게 함으로써 특정 노드에 장애 또는 네트워크 단절 상황이 발생해도 write 연산을 처리할 수 있는 가용성을 보장한다.

낙관적 복제 정책을 사용할 경우 복제 본 간의 데이터 일관성에 문제가 발생한다. Dynamo는 복제 본 간의 일관성을 맞추기 위해 두 가지 전략을 가지고 있다. 첫 번째 방법은 Dynamo 시스

템 내부적으로 처리하는 방법으로 마지막에 저장된 데이터를 최종 버전으로 저장한다. 두 번째는 애플리케이션에서 처리하는 방법이다. 애플리케이션에서 get(key) 연산을 이용하여 데이터를 조회하면 버전이 여러 개 있는 경우 여러 개의 버전과 이들에 대한 정보를 가지고 있는 컨텍스트를 전달한다. 애플리케이션은 내부 정책에 따라 저장할 데이터를 생성한 후 put(key, context, object) 연산을 호출하여 데이터를 저장한다. 데이터 복제는 여러 데이터 센터에 걸쳐 수행되며 특정 데이터 센터에 장애가 발생하여도 데이터 서비스가 가능하다.

Dynamo의 기본 요구사항은 write 연산에서 대한 고 가용성이지만 N, R, W라는 세가지 파라미터를 이용하여 성능, 가용성, 내구성(durability)의 수준을 설정함으로써 다른 요구사항을 갖는 서비스에서도 사용이 가능하다. N 값은 최종적인 복제 본의 개수를, R, W 값은 read, write 연산에서 성공적으로 처리되어야 하는 복제 본의 개수를 나타낸다. N의 값이 크면 내구성이 증가한다. W 값이 1이면 하나의 노드만 존재하는 경우에도 write를 보장하기 때문에 성능과 가용성은 증가하지만 복제 본 사이의 정합성이 떨어진다. 기본 구성에서 N, R, W의 값은 (3, 2, 2)이다.

Dynamo의 CAP 속성은 하나의 노드가 유효할 때까지 데이터 서비스가 가능하기 때문에 고 가용성을 지원하며 서비스 중단 없이 데이터 파티셔닝을 지속적으로 수행할 수 있다. 하지만 Eventual Consistency 정책으로 인해 정합성은 완벽하게 지원하지 않는다.

Dynamo의 소스는 공개되지 않았지만 Scalaris [22], Project Voldemort[23], Ringo[24] 등과 같은 Dynamo의 개념을 구현한 오픈 소스 프로젝트가 있다. Scalaris는 독일의 Zuse Institute Berlin (ZIB)에서 개발한 웹 2.0 서비스를 위한 대용량 분산 데이터 관리 시스템으로 공개소프트웨어이다. Scalaris는 네트워크 장애나 노드 장애 상황 등에서도 강력한 데이터 일관성을 보장한다. Dynamo 등과 같은 P2P 기반의 분산 데이터 관리 시스템은 대부분 eventual consistency만 제공하는 것에 비해 Scalaris는 하나의 트렌젝션 내에서 데이터의 일관성과 복제 본 간의 동기화를 보장한다. 또한 하나의 트렌젝션 내에서 복수의 데이터에 대한 트렌젝션 처리 기능을 제공한다. Scalaris는 물리적인 저장소에 저장하는 기능을 제공하지 않고 메모리만 사용한다.

Project Voldemort는 key, value를 저장하는 분산 데이터 저장소로 Dynamo의 개념을 구현한 오픈 소스 프로젝트이다. 데이터는 자동으로 파티셔닝 되어 분산된 노드에 배치되고 복제된다. 파티셔닝은 Dynamo와 같이 consistent hashing을 이용한다. 복제 본 간의 일관성은 write 성능을 위해 모든 복제 본에 값이 저장된 후에 트렌젝션을 종료하지 않고 하나의 노드에 버전 정보와 같이 저장 한 다음 데이터를 읽을 때 버전 정보를 이용하여 버전이 맞지 않는 경우를 찾아 처리하는 기법을 사용한다.

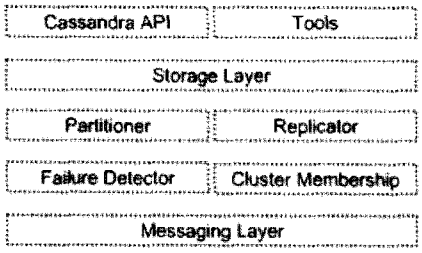
Ringo는 consistent hashing을 이용한 분산, 복제 key-value 저장소이다. Ringo는 주로 4KB 미만의 작은 데이터나 100MB 미만의 중간 크기의 데이터를 실시간 저장하는 목적에 사용한다. Ringo에 저장된 데이터는 immutable 데이터로 한번 저장된 데이터는 수정, 삭제할 수 없다. Ringo에 저장된 데이터는 low latencies (<10ms)로 데이터를 조회할 수 있으며 bulk data access 기능도 제공한다. Write 연산에 대해서 복제 본 간의 정합성은 보장하지 않는다. 데이터가 정해진 크기보다 큰 경우에는 외부 파일로 저장하고 Ringo DB에는 위치 정보만 저장한다.

2.3 Cassandra

Cassandra[25]는 P2P 네트워크 환경에서 구조화된 데이터 저장소를 제공하는 시스템으로

Facebook에서 개발하여 공개하였으며 현재 Apache incubation 프로젝트로 등록되어 있다.

Cassandra는 Bigtable의 데이터 모델, In-Memory/On-Disk 처리 기법과 Dynamo의 consistent hashing 기법을 혼합하여 구성한 시스템이다. Bigtable은 데이터의 파티셔닝 정보를 META 테이블을 이용하여 별도로 관리하지만 Cassandra는 consistent hashing 기법을 이용하고 있기 때문에 META 정보를 가지고 있지 않다. Cassandra의 데이터 모델의 Bigtable과 비슷하지만 Simple, Super 두 가지 타입의 Column Family를 제공한다.



(그림 4) Cassandra 시스템 구성

클라이언트는 클러스터 내 임의의 서버로 read/write 연산 요청하면 Partitioner에 의해 해당 데이터를 서비스하는 노드를 결정한다. Partitioner는 hash를 이용하여 노드를 찾는다. 커밋 로그와 데이터 복제는 eventual consistency 전략으로 여러 노드에 복제한다.

Cassandra의 CAP 속성은 Dynamo와 같이 고가용성과 Partition Tolerance 속성은 제공하지만 데이터 정합성은 보장하지 않는다.

2.4 CouchDB

CouchDB[26]는 Apache foundation의 오픈소스로 문서기반(Document oriented) 분산 데이터 관리 시스템이다. CouchDB가 저장하는 단위는 문서(Document)이며 문서 내에는 메타 데이터를 위해 다수의 필드를 가질 수 있으며 필드는 유일하게 식별할 수 있는 필드 명과 필드 값으로 구성된다. 트렌잭션의 단위는 문서 단위가 되며 문서 내의 임의의 필드만 수정/삭제하는 기능은 제공하지 않는다.

CouchDB의 가장 큰 특징은 read/write에 대해 잠금 처리를 하지 않기 때문에 뛰어난 성능을 보장할 수 있다는 것이다. 잠금 처리를 하지 않기 때문에 동시에 여러 클라이언트가 동일한 문서에 대해 write/read 연산을 수행할 경우 버전 충돌이 발생한다. 동일한 document에 대해 다른 버전이 존재할 경우 CouchDB는 버전 충돌(conflict)를 자동으로 감지하여 사용자에게 제공한다. CouchDB의 read 연산은 각각의 클라이언트는 최종 버전의 스냅샷을 가지고 있는 형태의 Multi-Version Concurrency Control (MVCC)[27] 모델을 사용한다. CouchDB는 Erlang으로 구현되어 있어 ErlangVM 상에서 수행되며 문서를 저장하는 Storage Engine, 데이터를 복제하는 Replicator, View를 처리하는 View Engine으로 구성된다.

CouchDB는 데이터의 복제 기능만 제공하며 분산된 노드로 데이터를 분산 배치하는 기능은 제공하지 않는다. CAP 속성 중 가용성은 제공하지만 정합성 속성은 지원하지 않는다.

지금까지 분산 데이터 관리 시스템에 대해 살펴 보았다. 본 문서에서 살펴 본 분산 데이터 관리 시스템은 데이터의 파티셔닝 및 배치에 대한 정보를 중앙에서 관리하고 분산 파일시스템을 사용하는 Bigtable 방식과 consistent hashing 기법을 이용하는 Dynamo와 같은 방식으로 나눌 수 있다. Bigtable 기반의 분산 데이터 관리 시스템은 주로 분산 파일 시스템을 이용하기 때문에 분산 파일 시스템에서 제공하는 수준의 정합성을 제공하고 있다. 또한 배치 정보를 중앙 집중 관리하고 있어 저장된 데이터의 분석이 필요한 경우에는 MapReduce와 같은 분산/병렬 컴퓨팅 플랫폼과 쉽게 연동할 수 있는 장점이 있다. 반

면 consistent hashing을 이용하는 Dynamo와 같은 시스템은 고 가용성을 필요로 하는 비교적 단순한 데이터에 대한 실시간 저장 및 조회가 필요한 시스템에 주로 사용한다.

3. 클라우드 데이터 서비스

클라우드 데이터 서비스는 클라우드 컴퓨팅 환경 내에서 구조화된 데이터에 대한 저장, 질의, 인덱스 등과 같은 데이터 관리 시스템의 기능을 제공하는 서비스를 말한다. 클라우드 데이터 서비스라는 용어는 업계 표준으로 정착된 용어가 아니며 DaaS(Data as a Service), 클라우드 데이터베이스(Database), 클라우드 데이터 액세스(Data access) 등 다양한 용어가 존재한다.

클라우드 데이터 서비스는 2007년 말 아마존에서 SimpleDB 베타 버전을 출시한 것을 시작으로 2008년 마이크로소프트, 구글 등에서도 이와 유사한 서비스를 출시하였다. 클라우드 데이터 서비스는 다음과 같은 공통된 특징을 가지고 있으며 2장에서 살펴본 분산 데이터 관리 시스템과 비슷한 특징을 가지고 있다.

- 확장성
  클라우드 환경에서의 데이터 서비스는 데이터 저장공간 및 데이터 센터 등의 지역적인 제약 조건 없이 확장 가능한 서비스를 제공한다.
- 고 가용성
  클라우드 내에 저장된 데이터는 일부 서버의 장애 또는 데이터 센터 전체 장애 상황에서도 데이터 서비스를 제공받을 수 있다.
- 약한 데이터 정합성
  데이터 정합성은 서비스 별로 차이가 있지만 관계형 데이터 관리 시스템에서 제공하는 수준의 일관성은 제공하지 않는다. 데이터 일관성에 대한 문제는 응용 애플리케이션에서 처리해야 하는 부담이 있다.
- 단순한 데이터 모델
  대부분의 클라우드 데이터 서비스는 관계형 데이터 모델은 지원하지 않으며 key/value 기반의 단순한 형태의 데이터 모델만 지원한다. 또한 테이블의 스키마와 데이터 타입에 대한 엄격한 정의가 없다.
- 요금 정책
  클라우드 데이터 서비스는 트래픽, 저장공간 등의 사용량에 따라 서비스 사용료를 부과한다. 이 경우에도 미리 용량을 확보할 필요 없이 사용한 만큼만 비용을 지불한다.
- 보안
  전통적인 데이터 관리 시스템은 웹 서버나 애플리케이션 서버, 관리자 등 일부 한정된 서버 또는 사용자로부터의 접근만을 허용하여 보안에서 많은 것을 고려할 필요가 없다. 반면, 클라우드 데이터 서비스는 모든 서비스 사용자가 동일한 인터페이스를 통해 접근하고, HTTP 프로토콜 기반으로 데이터가 서비스되기 때문에 보안이 중요한 요소가 된다.

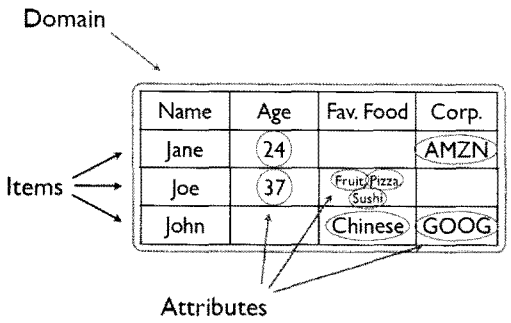
이번 장에서는 최근 서비스되거나 베타 서비스되고 있는 클라우드 데이터 서비스의 사례로 Amazon SimpleDB, Microsoft SSDS, Google AppEngine Datastore를 소개한다.

3.1 SimpleDB

SimpleDB[6]는 아마존에서 출시한 클라우드 데이터 서비스로 데이터의 인덱스와 질의와 같은 핵심 데이터베이스 기능을 웹 서비스를 통해 제공하는 서비스이다.

SimpleDB의 데이터 모델은 Domain, Item, Attribute이다. Domain은 Item의 집합으로 관계형 데이터 관리 시스템의 테이블과 유사한 개념이지만 Item 객체를 저장하는 ArrayList로 정의할 수 있다. 하나의 Domain내에 저장된 Item 객체는 다른 Item과 관계를 갖지 않는다. 따라서 도메인 간에 Foreign key도 존재하지 않는다. SimpleDB내의 각각의 질의는 하나의 도메인에서만 수행된다. Item은 Attribute의 집합이다. 관

계형 데이터 관리 시스템의 row와 비슷하지만 Item 이름을 가지고 있으며 동일한 Domain에 있는 서로 다른 Item은 동일한 Attribute 개수를 가질 필요는 없다. Attribute는 Attribute 이름과 n개의 텍스트 값을 가진다. 다음 (그림 5)은 SimpleDB에 저장된 하나의 Domain에 대한 예이다.



(그림 5) SimpleDB Domain 예

SimpleDB에 저장된 모든 데이터는 자동으로 인덱스가 생성되며 가용성을 위해 Domain 단위로 복제한다. 복제 본 간의 데이터 일관성은 eventual consistency 정책을 취하고 있다. 데이터 저장 명령을 호출하고 성공했다는 메시지를 받은 경우 데이터는 일정 시간이 지난 후에는 반드시 복제 본에 저장되지만 저장 요청 즉시 저장된 데이터를 조회할 경우 데이터가 조회되지 않을 수 있다. 모든 복제 본까지 저장되는 시간은 수초 정도 소요될 수 있다.

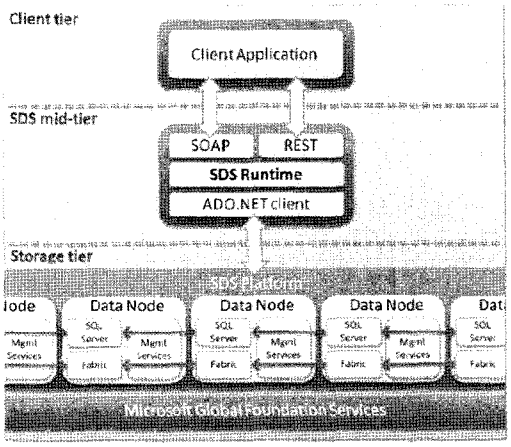
데이터의 저장뿐만 아니라 애플리케이션에서 SimpleDB로 질의나 연산을 요청하는 경우 CPU와 메모리 등과 같은 클라우드 내의 자원을 사용하기 때문에 자원에 대한 사용료를 지불해야 한다. 따라서 SimpleDB는 모든 요청에 대해 BoxUsage 라고 하는 사용료의 기준이 되는 시스템 리소스를 얼마나 사용하는지에 대한 정보를 제공한다. SimpleDB는 SOAP이나 REST 프로토콜을 이용하여 데이터를 조회하거나 저장한다.

SimpleDB 서비스는 CAP 속성 중 데이터의 일관성은 제공하지 않으며 고 가용성을 제공한다.

3.2 SQL Data Service

마이크로소프트는 2008년 하반기에 Azure [32] 서비스 플랫폼이라고 하는 클라우드 서비스 플랫폼을 출시하였다. Azure 서비스 플랫폼은 Windows Azure, Microsoft .Net Services, Microsoft SQL Services, Live Services 등으로 구성되어 있으며 Microsoft SQL Service는 Azure 플랫폼 내에서 클라우드 데이터 서비스를 제공한다. SQL Service내에는 다양한 데이터 서비스가 제공될 예정이지만 현재에는 SQL Data Service[7]만 제공한다.

SQL Data Service는 기반 기술로 SQL Server를 사용하고 있지만 확장성, 가용성 등을 위해 SQL Server에서 제공하는 JOIN등과 같은 관계형 데이터 관리 시스템의 핵심 기능은 제공하지 않는다. 다음 (그림 6)은 SQL Data Service를 구성하는 기술 스택이다.



(그림 6) SQL Data Service 기술 스택

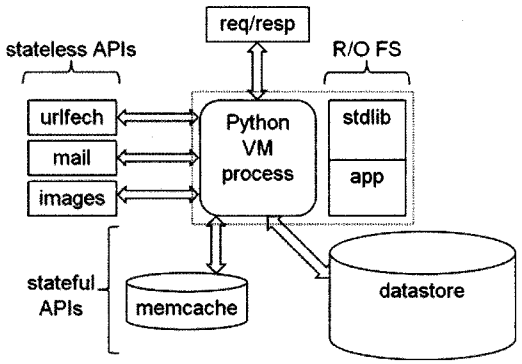
SQL Data Service의 데이터 모델은 Authority, Container, Entity(ACE 모델)로 구성되어 있다. SQL Data Service는 여러 데이터 센터를 통해 서

비스 되며, 각 데이터 센터는 여러 개의 Authority 정보를 저장하고, 각 Authority는 유일한 DNS 이름을 가진다. Authority는 n개의 Container를 가지고, Container는 하나의 데이터 센터 내에서 복제된다. Container는 부하 분산과 가용성의 단위가 되며 Container에 장애가 발생하면 자동으로 새로운 복제 본을 만든다. 하나의 Container는 다수의 Entity를 가지며 각 Entity는 다수의 Property를 가진다. Property는 name, type, value로 구성된다.

SQL Data Service는 모든 복제 본에 저장되어야만 트랜잭션을 성공시키는 transactional consistency 정책을 통해 정합성을 제공한다. SQL Data Service 서비스 백서에는 고 가용성도 지원한다고 되어 있지만 구체적으로 수준은 나와 있지 않다.

3.3 Google AppEngine Datastore

구글은 독립적인 클라우드 데이터 서비스는 제공하지 않지만 웹 애플리케이션을 클라우드 환경에서 수행하게 하는 서비스인 AppEngine [8] 서비스 내에 Datastore라는 이름으로 데이터 서비스를 제공한다. AppEngine Datastore는 앞에서 설명한 다른 데이터 서비스와 달리 Python 기반의 API를 이용해야 하며 반드시 AppEngine내에서만 수행되어야 하는 제약이 있다.



(그림 7) Google AppEngine 구성

AppEngine Datastore의 저장 단위는 Entity이며 Entity는 Kind, Key와 다수의 Property를 가진다. Key는 자동으로 생성하거나 사용자가 입력한 문자열이 되며 foreign key로 연결될 수 있다. Type은 Entity의 종류를 나타내며 관계형 데이터 관리 시스템의 테이블 명이나 객체지향 개념에서 class에 해당한다. 하나의 Entity는 여러 개의 Property 가질 수 있으며 각각의 Property는 name과 value를 가진다. 동일한 Type을 가지는 Entity가 동일한 Property를 가지고 있을 필요는 없다.

Property의 value에 사용 가능한 데이터 타입은 String, Bool, Byte, Int, Long, DateTime 등과 같은 일반적인 데이터 타입, Value내에 다시 n개의 값을 저장할 수 있는 List 타입, Email, URL Link 등과 같은 특별한 정보를 저장할 수 있는 타입을 제공한다. 타입 중에는 다른 Entity를 가리킬 수 타입이 있으며 이런 기능을 이용하여 one-to-many 또는 many-to-many 관계를 구성할 수 있다.

하나의 트랜잭션 내에서 여러 개의 연산을 수행할 수 있으며 처리 도중 특정 연산에 실패할 경우 트랜잭션 내에서 수행된 모든 연산에 대해 rollback 할 수 있는 기능을 제공한다. 테이블의 인덱스는 프로그램 내부에서 사용하는 질의를 이용하여 Datastore가 자동으로 생성하거나 사용자 정의 인덱스를 구성할 수 있다.

AppEngine Datastore는 내부적으로 Bigtable을 활용하여 구현하고 있기 때문에 Bigtable의 CAP 속성을 따른다.

다음 표는 지금까지 설명한 분산 데이터 관리 시스템과 클라우드 데이터 서비스에 대한 비교표이다.

<표 1> 분산 데이터 관리 시스템 및 클라우드 데이터 서비스 특징 비교

항목	Bigtable	Dynamo	Cassandra	CouchDB	SimpleDB	SQL Data Service	AppEngine
Consistency	O	X	X	X	X	O	O
Availability	X	O	O	O	O	X	X
Partition-Tolerance	O	O	O	O	O	O	O
Partition 관리	META	hashing	hashing	X	N/A	N/A	N/A
Replication	O	O	O	O	O	O	O
Data Model	Bigtable	blob	Bigtable	Document	Domain Item Attribute Value	Container Entity Property	Entity Property
Index	Row key Column key	X	X	DocID	All values	All properties in entity	All Properties User defined composited index
Language	C++	Java	Java	Erlang	N/A	N/A	N/A
Persistence	GFS	BDB MySQL	Disk	BDB	Unknown	MS SQL Server	Bigtable
Client Protocol	C++ API	HTTP	Thrift	REST	SOAP REST	SOAP REST	Python API
Open Source	X	X	O	O	service	service	service
License	X	X	Apache	Apache	N/A	N/A	N/A
Company	Google	Amazon	Facebook	Apache	Amazon	Microsoft	Google

4. 결 론

지금까지 최근 발표된 분산 데이터 관리 시스템과 클라우드 데이터 서비스에 대해 살펴보았다. 분산 데이터 관리 시스템은 관계형 데이터 모델보다 단순한 모델을 제공하며, 대용량의 데이터를 분산된 노드에 배치시키고 데이터 복제를 통해 고 가용성과 확장성에 초점을 맞추고 있다. 이러한 특징은 대규모의 데이터를 다루는 인터넷 서비스 회사들이 그들의 서비스 요구사항에 맞는 데이터 관리 시스템을 개발하는 과정에서 도출되었다. 따라서, 이러한 분산 데이터 관리 시스템을 사용하거나 참조할 때는 이들 기술의 배경과 서비스의 성격에 대한 이해가 필요하며, 각 기술이 제시하는 데이터 모델, 정합성, 고 가용성, 성능, 확장성 등의 다양한 특성을 고려해야 한다.

SimpleDB, SQL Data Service, AppEngine Datastore 등과 같은 클라우드 데이터 서비스는 현재까지는 대부분 베타 서비스 등의 형태로만 제공되어 있다. 따라서, 데이터 서비스를 사용하여 구축된 시스템들은 아직 단순한 기능만 제공하거나 인터넷 서비스 업체에서 제공하는 오픈 API를 이용하여 만든 Mashup 형태의 서비스만 제공하는 수준이다. 하지만, 클라우드 데이터 서비스의 미래는 밝다. 그 이유는 날이 갈수록 데이터 처리량은 많아지면서 데이터 저장 규모도 커지는 가운데, 이러한 시스템을 직접 구축하는 일은 어렵고 비용이 많이 드는 상황에서 클라우드 데이터 서비스는 손 쉬운 개발 방법을 제공하고, 데이터 저장과 처리 능력의 확장 및 데이터 가용성에 대한 고민을 덜어주기 때문이다. 클라

우드 데이터 서비스가 안정적으로 정착되기 위해서는 개발 과정에서 서비스에 접속하지 않고서도 개발자 고유의 환경에서 개발 및 테스트가 가능하도록 지원을 해야 하며, 서로 다른 데이터 서비스들간의 표준적인 인터페이스나 질의 언어를 수립하는 일이 필요하다.

참고문헌

[1] J. Gray. "The Transaction Concept: Virtues and Limitations." In Proceedings of VLDB. Cannes, France, September 1981, 144-154.

[2] A. Fox, S. Gribble, Y. Chawathe, E. Brewer, P. Gauthier, "Cluster-based scalable network services", In Proceedings of the sixteenth ACM symposium on Operating systems principles, p.78-91, 1997.

[3] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, P. Helland, "The End of an Architectural Era (It's Time for a Complete Rewrite)." In Proceedings of VLDB, 2007.

[4] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, R. Gruber, "Bigtable: A Distributed Storage System for Structured Data.",InProceedingsofthe7thOSDI,2006.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels, "Dynamo Amazon's Highly Available Key-value Store", In Proceedings of SOSP'07, 2007.

[6] SimpleDB, http://aws.amazon.com/simpledb/

[7] "Microsoft SQL Data Services White Paper", Microsoft, 2008.

[8] Google AppEngine, http://code.google.com/appengine/

[9] E. Brewer. "Towards robust distributed systems. (Invited Talk) Principles of Distributed Computing", Portland, Oregon, 2000.

[10] S. Gilbert , N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", ACM SIGACT News, v.33 n.2, 2002.

[11] S. Ghemawat, H. Gobioff, S.-T. Leung "The Google file system." In Proceedings of the 19th ACM SOSP, pp. 29-43, 2003.

[12] J. Dean, S. Ghemawat, "MapReduce: Simplified data processing on large clusters." In Proceedings of the 6th OSDI, pp. 137-150, 2004.

[13] M. Burrows, "The Chubby lock service for loosely coupled distributed systems." In Proceedings of the 7th OSDI, 2006.

[14] W. Hsieh, J. Madhavan, R. Pike. "Data management projects at Google." In Proceedings of ACM SIGMOD, pages 725-726, 2006.

[15] Neptune, http://dev.naver.com/projects/neptune

[16] HBase, http://www.hbase.org

[17] Hadoop, http://hadoop.apache.org

[18] Pleiades, http://dev.naver.com/projects/neptune

[19] Hypertable, http://www.hypertable.org

[20] D. Silva, L. Soares, O. Krieger. "KFS: Exploring flexilibity in filesystem design." In Proceedings of the Brazilian Workshop in Operating Systems, Salvador, Brazil, August 2004.

[21] 김병오, 이일우, 박호진, "분산 해시 테이블 기반 P2P 기술 동향" 전자통신동향분석 제21권 제6호, pp. 179-189, 2006.

[22] T. Schutt, F. Schintke, A. Reinefeld. "Scalaris: Reliable transactional P2P key/value store." In ACM SIGPLAN Erlang Workshop, 2008.

[23] Project Voldemort, http://project-voldemort.com

[24] Ringo, http://github.com/tuulos/ringo/tree/master

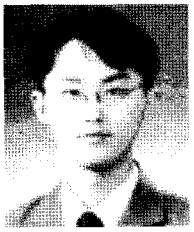
[25] Cassandra, http://code.google.com/p/the-cassandra-project

[26] CouchDB, http://couchdb.apache.org

[27] P. Bernstein, N. Goodman, "Multiversion concurrency control-theory and algorithms", ACM Transactions on Database Systems (TODS), v.8 n.4, pp.465-483, 1983.

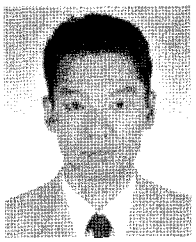
[28] Azure, http://www.microsoft.com/azure/

저자약력



문 상 철

2000년 2월 연세대학교 전산학과 (이학사)
2007년 8월 한국정보통신대학교 공학부 (공학석사)
2007년 8월~현재 NHN CTO 선행기술개발랩
관심분야 : 분산 파일 시스템 및 분산 데이터 관리 시스템
이 메 일 : sangchul@nhncorp.com



김 형 준

1995년 2월 성균관대학교 정보공학과 졸업
2006년 7월~현재 NHN CTO 선행기술개발랩
관심분야 : 분산 파일 시스템 및 분산 데이터 관리 시스템
이 메 일 : jindolk@nhncorp.com