

RDBMS, Hadoop, Neptune

작성자: 김형준(babokim@gmail.com, <http://dev.naver.com/projects/neptune>)

작성일: 2009.07.13



이 저작물은 [크리에이티브 커먼즈 코리아 저작자표시-비영리-변경금지 2.0 대한민국 라이선스](http://creativecommons.org/licenses/by-nc-nd/2.0/)에 따라 이용하실 수 있습니다.

최근 No-SQL(<http://www.itworld.com/open-source/70146/no-sql-anti-database-movement-gains-steam>) 또는 Anti-RDBMS(<http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores>)라는 내용으로 RDBMS로 부터 벗어나고자(?) 하는 실험들이 진행되고 있다.

이런 새로운 시도는 저장해야 할 데이터가 계속해서 증가하고 twitter와 같이 대량의 트랜잭션을 실시간 처리해야 하는 서비스가 많아지면서 기존의 관계형 데이터베이스와는 다른 새로운 개념의 데이터 저장소에 대한 요구사항이 늘어났기 때문이다.

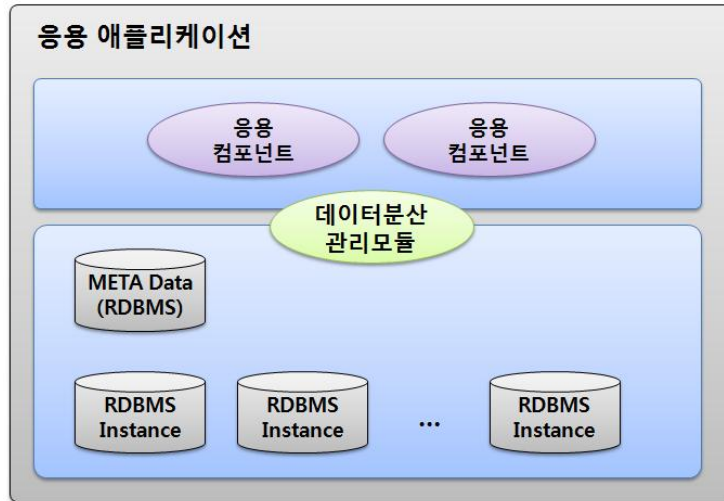
국내에서는 이제 겨우 Hadoop에 대한 기술적 검토만 수행되거나 적용되고 있는 상황이다. 따라서 관계형 데이터베이스(MySQL)와 분산파일시스템(Hadoop FS), 분산데이터저장소(Neptune) 등에 대한 정확한 차이점과 사용 용도에 대해 혼란이 많다. 이번 글에서는 이들 사이의 차이점과 각 사용 사례별로 적합한 저장소를 살펴보고자 한다.

No-SQL이나 Anti-RDBMS에서 주장하는 RDBMS의 문제점 또는 제약 조건으로 제시하는 데이터의 속성은 주로 확장성과 안정성이다.

확장성

RDBMS는 하나의 DBMS 인스턴스 내에서 저장할 수 있는 데이터의 크기는 제한적이다. 오라클과 같은 상용 DBMS의 경우에는 하나의 인스턴스에 수십억 이상의 레코드와 수백 GB 이상의 데이터를 저장할 수는 있지만 이보다 더 큰 데이터의 경우 여러 노드에 분산 배치하거나 Oracle-RAC 등과 같은 분산 기반의 솔루션을 도입해야 한다. MySQL과 같은 오픈소스 DBMS의 경우 하나의 인스턴스에서 많은 수의 레코드를 서비스하기 위해서는 적절하게 튜닝해야 하며 DBMS가 설치된 장비의 디스크 용량 내에서만 서비스 가능하다.

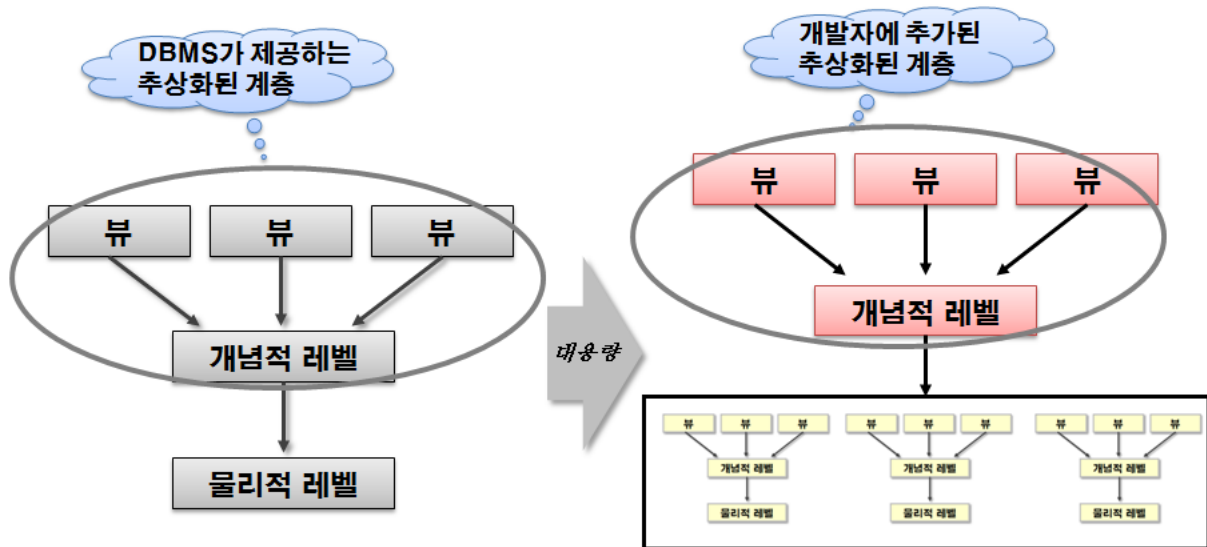
따라서 지금까지는 하나의 장비에서 수용 가능한 디스크 크기(보통 수백 GB) 이상의 데이터 저장에 대한 요구가 있는 경우 여러 대의 RDBMS에 분산시키고 분산된 DBMS의 데이터 파티션에 대한 정보를 관리하는 모듈을 각자 개발하여 사용하였다.



[그림] RDBMS를 이용한 클러스터 구성

이렇게 파티션 될 경우 분리된 파티션 간에는 RDBMS의 최고 강점인 엔티티 간의 관계연산 즉 JOIN 연산을 사용할 수 없다. 이런 문제를 해결해주는 솔루션이 Distributed RDBMS이지만 비용이 비싼 단점이 있다. 어쨌든 최근의 No-SQL에 대한 시도는 “RDBMS에서 JOIN을 사용할 수 없다면 굳이 RDBMS를 사용할 필요가 있는가?” 라는 의문에서 시작했다고 할 수 있다.

이런 구성에서는 데이터의 파티셔닝, 파티셔닝에 따른 query 수행 전략 등의 코드를 모두 개발자가 만들어야 한다. DBMS의 역할 중에 하나는 구조화된 데이터의 접근에 대해 추상화된 계층을 제공하여 개발자가 쉽게 접근 가능하도록 하는 것인데, 위와 같은 구조에서는 이런 역할을 수행하는 계층을 개발자가 구현해야 하는 부담이 있다.



[그림] 데이터관리시스템의 기능-추상화된 데이터관리 계층

또한 이런 정적인 구성에 가장 큰 문제점은 최초 구성된 클러스터 보다 더 많은 데이터 저장 공간이 필요하게 될 경우 새로운 DBMS 인스턴스를 추가하는 것 이외에 전체 클러스터에 대해 파티셔닝 작업을 다시 해야 하며 이런 작업은 보통 시스템을 멈춘 상태에서 작업을 하거나 데이

터 유실이 발생할 수 있는 아주 위험한 작업이 된다.

따라서 데이터의 크기가 하나의 서버에서 처리할 수 있는 용량(보통 수백 GB)보다 더 큰 경우 분산 환경을 고려해야 한다. 가장 간단한 방법은 앞서도 잠깐 언급한 분산 RDBMS를 사용하는 것이다. 분산 RDBMS는 기존의 RDBMS 기능(JOIN 등)을 그대로 가지면서 대용량 데이터를 저장할 수 있는 솔루션이다. 분산 RDBMS의 가장 큰 문제는 비용이다. twitter에서 발생하는 데이터나 웹에서 발생하는 데이터 등을 위해 이렇게 고 비용 저장소가 필요하지는 않다. 물론 텔레콤 회사의 요금 계산의 기초가 되는 콜 정보 등과 같이 중요하면서 대용량을 요구하는 데이터의 경우 고가의 분산 RDBMS에 저장할 수 있을 것이다.

다음으로 생각할 수 있는 스토리지는 Hadoop 등과 같은 오픈소스 기반의 분산 데이터 저장소이다. Hadoop은 데이터를 저장하는 파일시스템(hdfs, Hadoop distributed file system)과 데이터를 분석하는 MapReduce 두 가지 핵심적인 기능을 제공한다. Hadoop에서 가장 오해를 많이 하는 부분이 데이터 저장 부분이다. Hadoop에서 제공하는 데이터 저장소는 단순히 파일 시스템 기능만 제공하기 때문에 파일 내부에 저장된 데이터 중 일부를 검색하거나, 키를 이용하여 데이터를 수정하는 등과 같은 실시간 연산에서는 사용하기 어렵다. Hadoop 파일 시스템이 실시간 처리가 가능하다는 오해를 갖게 된 가장 큰 원인은 Hive라고 하는 Hadoop의 서브 프로젝트 때문이다. Hive는 SQL과 비슷한 문법을 이용하여 Hadoop 내에 테이블을 생성하고 생성된 테이블에 데이터를 저장하거나 조회하는 기능을 제공한다. 하지만 Hive에서 수행되는 모든 SQL(Query)은 MapReduce 작업으로 수행되며 Hadoop은 MapReduce 작업을 실행하는데 최소 수초 이상의 준비 작업을 거친다. 따라서 Hive는 실시간 처리를 위한 SQL이 아니라 배치 처리(MapReduce) 작업을 SQL 문법을 이용해서 쉽게 해주는 도구에 불과하다.

또 다른 데이터 저장소로는 최근 많이 이슈화 되고 있는 Key-value 스토리지가 있다. Key-Value 스토리지는 대용량 데이터에 대해 심플한 데이터 모델과 실시간 처리를 주요 기능으로 제공한다. 따라서 엔티티의 관계 연산(JOIN)은 지원하지 않으며 분산(또는 standalone)된 환경에서 운영된다. Key-Value 스토리지는 구글의 Bigtable 개념을 도입한 스토리지와 아마존의 Dynamo의 개념을 도입한 스토리지로 구분할 수 있다.

아마존의 Dynamo 개념을 도입한 스토리지는 DHT(Distributed Hash Table)라고 하는 해쉬 기반의 키 파티셔닝 전략을 이용하여 데이터를 분산된 노드에서 저장한다 한다. Bigtable 개념을 도입한 스토리지는 특정 Key를 서비스하고 있는 노드에 대한 정보를 별도의 META 정보로 관리하며 이들 Key에 대해서 인덱스 정보까지 가지고 있어 Dynamo 계열 저장소 보다는 다양한 데이터 연산을 제공한다.

Key-Value 저장소는 기본적으로 대용량 처리를 목적으로 설계되었기 때문에 데이터의 확장성은 대체적으로 잘 지원하고 있다. 앞서 언급한 RDBMS를 분산 배치시켜 개발자가 만든 프로그램에 의해 관리되던 부분을 Key-Value 저장소가 대체함으로써 개발자에게 다시 추상화된 데이터 저장소 계층을 제공하는 데이터관리시스템의 본연의 기능을 수행하고 있다.

안정성

RDBMS는 다양한 방법을 이용하여 저장된 데이터의 안정성은 확보한다. 데이터 파일을 저장하는 스토리지 자체에서 안정성을 제공할 수도 있으며(이런 스토리지는 일반적으로 가격이 비싸다), 주기적으로 백업을 받거나, RDBMS 솔루션 자체에서 제공하는 replication 기능을 이용할 수도 있

다. 이런 모든 작업이 관리자를 필요로 하게 된다. 특정 DB 서버에 장애가 발생할 경우 백업 받은 데이터를 이용하여 복구하거나 replication 된 서버가 다시 active한 master로 인식하게 하는 등의 작업이 필요하다. RDBMS의 기본 전제는 DBMS가 수행된 서버에 장애가 발생하지 않는다는 가정하에 만들어진 솔루션이기 때문에 안정적인 스토리지, 백업 등과 같은 DBMS와 관련 없는 솔루션이나 기법들이 추가되어야 한다. 이렇게 추가되는 항목들은 비용을 발생시키거나 더 많은 관리를 필요로 하게 된다. 또한 대용량 데이터 저장을 위해 다수의 데이터베이스 서버를 운영하는 경우라면 이런 관리가 아주 복잡하게 엉켜버릴 수도 있다. 수 백대의 MySQL 서버를 Master-Slave 관계로 구성하고 그 중 5% 서버에 장애가 항상 발생할 수 있다고 상상해보라.

Hadoop이나 대부분의 Key-Value 스토리지는 특정 서버의 장애를 일반적인 상황으로 간주하여 설계되어 있기 때문에 솔루션 자체 내에 안정성을 보장하고 있다. 데이터의 복제, 서버 장애 발생 시 self-healing 기능 등이 이미 구현되어 있기 때문에 특정 서버에 장애가 발생하여도 정상적인 데이터 서비스가 가능하며, 장애가 발생한 서버를 빼고 새로운 서버를 넣고 몇 개의 단순한 명령만 수행하면 새로운 노드가 추가된다.

적합한 스토리지의 선택

아직까지 많은 프로젝트에서 요구사항은 대용량임에도 불구하고 NAS, Local Disk, RDBMS 만을 이용하여 시스템을 설계한다. 당장 시스템을 만들어 오픈 할 수는 있지만 1 ~ 2년 내에 쌓여가는 데이터를 보면서 증설 계획이나 시스템 업그레이드 계획을 수립하게 된다. 이제 개발자 NAS, Local Disk, RDBMS뿐만 아니라 추가로 분산파일시스템, 분산데이터저장소를 이용할 수 있다. 시스템의 아키텍처 단계에서 요구사항 분석 후 시스템을 구성하는 데이터의 특성을 분석하고 각 특성에 맞는 적절한 스토리지를 선택함으로써 개발 -> 관리 -> 시스템 중지 -> 확장 -> 관리가 반복되는 악순환을 제거할 수 있다.

적합한 스토리지를 선택하기 위해서는 먼저 데이터가 요구하는 속성부터 정의해야 한다. 일반적으로 다음과 같은 속성을 정의할 수 있다.

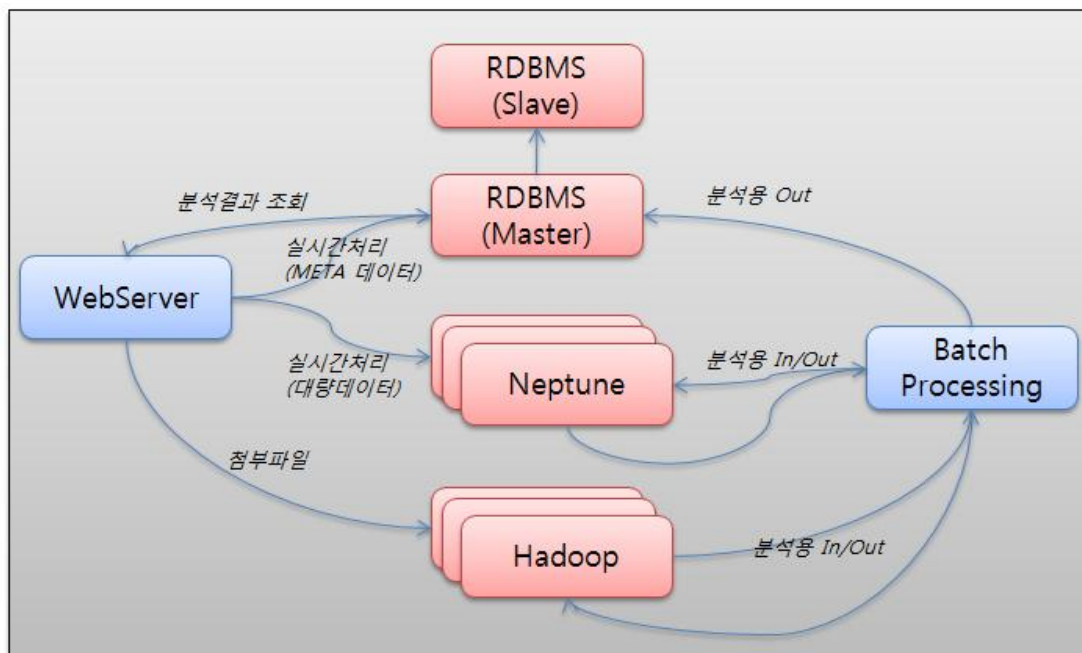
- 데이터 용량
스토리지에 저장될 데이터가 수백 GB 이상 이거나 한대의 장비에 저장할 수 있는지 여부
- 실시간 데이터 처리
데이터의 처리(select, insert)가 실시간 처리를 요구하는지 여부
- 데이터 복잡성
다양한 데이터가 존재하고 데이터 간의 관계가 복잡한지 여부
- 안정성
아주 중요한 데이터라서 고 가용성을 요구하는지 여부
- 분석 작업과 연계
저장된 데이터를 이용하여 분석 작업이 수행되는지 여부
- 비용

이런 데이터 특성에 대해 각 스토리지는 다음과 같이 지원한다.

구분	데이터 용량 (확장성)	실시간 데이터 처리	데이터 복잡성	안정성	분석작업 연계	비용
Local Disk	X	X	X	X	X	Low
NAS	O	X	X	O	X	Middle
RDBMS	X	O	O	△ (or Difficult)	X	High
Distributed RDBMS	O	O	O	△ (or Difficult)	△	Very High
Hadoop	O	X	X	O	O	Low
Bigtable 계열 (Neptune 등)	O	O	△	O	O	Low
Dynamo 계열 (Dynomite 등)	O	O	X	O	△	Low

위의 스토리지 특성과 요구사항에서 도출된 데이터 특성을 비교하여 적합한 스토리지를 선택한다. 최근 시스템은 다양한 특성을 가진 데이터를 관리해야 한다. 따라서 시스템에서 사용되는 스토리지 솔루션도 과거와 같이 NAS, RDBMS 두 종류로만 구성되는 것이 아니라 위에서 언급한 다양한 스토리지 구성을 고려해야 한다.

예를 들어 실시간 데이터 처리가 필요하고, 대용량인 반면 비용이 싼 스토리지가 필요하다면 Neptune이나 Dynomite 등을 고려할 수 있다. 다음은 RDBMS, Hadoop, Neptune으로 구성된 시스템 사례이다.



[그림] 애플리케이션의 구성 사례

위의 시스템 구성은 다음과 같은 요구사항을 만족하는 시스템 구성이다.

- 시스템으로 요청되는 데이터는 대부분 실시간으로 처리되어야 하며 데이터는 계속 누적되어야 한다(Neptune).
- 누적된 데이터는 실시간으로 조회되어야 한다(Neptune).
- 데이터 중 일부는 파일 형태의 데이터도 있다(Hadoop).
- 저장된 데이터는 주기적으로 배치 작업을 통해 분석되어야 하며(Hadoop MapReduce) 분석된 결과 중 일부는 데이터 크기는 작지만 복잡한 query 등에서 활용 가능해야 한다(RDBMS).
- 분석 결과 중 일부는 대용량 데이터이며 실시간으로 화면에서 조회 가능해야 한다(Neptune).
- 데이터 분석은 요구사항이 수시로 변경될 수 있으며 요구사항 변경에 따라 기존 저장된 데이터를 이용하여 빠르게 분석 가능해야 한다(Hadoop MapReduce).

결론

최근 Hadoop, Neptune 등과 같은 다양한 스토리지가 나오고 있으며 일부 개발자는 이를 자신의 시스템에 적용해보고자 기술적 욕심을 가지고 있다. 하지만 스토리지의 선택은 신중해야 하며 데이터의 특성에 적합만 스토리지를 선택해야 한다. 그렇지 않으면 개발 시 많은 공수가 필요하거나 시스템 오픈 후 1 ~ 2년 후에 문제가 발생할 수 있다.

Neptune이나 HBase와 같은 Bigtable 기반의 데이터 저장소를 사용하고 싶다는 문의가 많은데 요구사항을 자세히 분석해보면 Hadoop + RDBMS로 충분히 해결 가능한 구성인 경우가 많다. Neptune과 같은 저장소가 사용되는 기본 전제조건은 실시간 처리 여부와 데이터 용량이다. 실시간 처리가 필요하고 데이터가 작으면 RDBMS 사용하는 것이 최상의 선택이다. 개발자도 익숙하고 다양한 데이터 연산을 제공하기 때문이다. 실시간 처리가 필요하고 데이터 용량이 많은 경우라면 Neptune이나 분산 RDBMS를 고려할 수 있다. 저비용이며 상대적으로 덜 중요한 데이터이면 Neptune을 사용할 수 있다. 대용량이면서 실시간 처리가 필요 없는 경우라면 Hadoop 만으로도 충분하다. 저장된 데이터를 배치로 분석하였는데 분석된 결과가 다시 실시간으로 서비스 되어야 하고 대용량인 경우라면 Neptune과 같은 저장소를 사용해야 한다.

레퍼런스

- [1] <http://www.itworld.com/open-source/70146/no-sql-anti-database-movement-gains-steam>
- [2] <http://www.metabrew.com/article/anti-rdbms-a-list-of-distributed-key-value-stores>
- [3] 클라우드 컴퓨팅을 위한 분산 데이터 관리 시스템 및 데이터 서비스 기술(정보처리학회지, 2009년3월)
- [4] <http://dev.naver.com/projects/neptune>
- [5] <http://hadoop.apache.org/core>
- [6] <http://hadoop.apache.org/hbase>
- [7] <http://hadoop.apache.org/hive>
- [8] <http://github.com/cliffmoon/dynomite/tree/master>