특집 04

대용량 데이터 처리를 위한 RDBMS 최신 기술 동향

박헌영·송용주 (Tibero)

목 차 »

1. 빅데이터 열풍과 RDBMS

빅데이터라는 업계의 큰 화두에 접근하는 방식은 다양하다. 망이나 플랫폼 회사에서는 아마존 S3, KT uCloud와 같은 Big Storage 서비스를 제공한다. 데이터 마이닝 회사에서는 Hadoop과 같은 NoSQL에 기반한 데이터 분석 솔루션을 내놓았다. facebook, twitter 같은 서비스 회사에서는 MySQL, memcached 등을 조합한 자체 솔루션으로 서비스를 유지한다. 기존 RDBMS 벤더들은 Exadata[1], greenplum[2]과 같은 빅데이터 확장 솔루션을 출시했다. NoSQL과 RDBMS의 장점만을 모으려는 시도인 NewSQL[3]이란 개념까지 등장했다.

우리는 약 1년 전, DBMS를 빅데이터 환경에 맞게 설계하려면 고려해야할 점과 해결해야할 이슈들을 소개했었다[4]. 이 글에서는 당시 제기했던 문제들을 최근의 RDBMS 제품들에서 어떻게 해결하고 있는지 점검해보고 필자들이 개발 중인 Tibero Massive Cluster (TMC)와 비교해보려고 한다.

2. 빅데이터를 위한 RDBMS 기술 현황

대용량 데이터 처리에 대한 시장의 요구는 많은 벤더들에게 큰 자극제가 되었다. NoSQL진영과 RDBMS진영 간의 가장 큰 차이는 그 시작점에 있다. 대용량 데이터 처리에서 시작하여 언어와 사용 환경을 발전시켜 나가는 방법과, 표준형 언어(SQL)을 사용하는 RDBMS를 발전시켜 대용량 데이터 처리를 가능하게 나가는 방법으로 나뉜다. 초기에 대용량 데이터 분야의 주된 관심사는 전자의 방법에 있었으나, 기존의 RDBMS벤더들이 대용량 데이터 처리 분야에 뛰어들면서 초점이 바뀌고 있다.

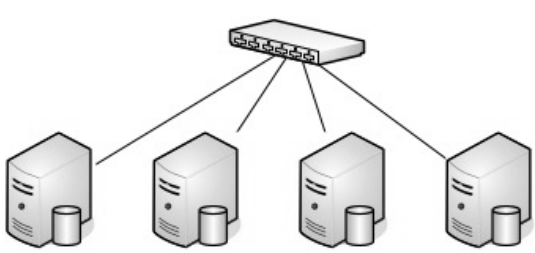
RDBMS를 기반으로 함으로써 얻을 수 있는 가장 큰 장점은 SQL처리가 가능하다는 데에 있다. 시스템마다 지원하는 SQL의 범위는 각 노드의 유닛으로 들어가는 RDBMS의 스펙으로 제한된다. SQL표준의 양은 방대하지만 실제로 사용하는 부분은 그리 많지 않다. SQL에서도 core-level로 지정된 부분정도만 대부분 사용되고 있

다. 특히 대용량 처리를 위해서라면 필요 없는 표준이 많다. 대부분의 벤더에서는 필요 없는 표준을 과감하게 버리고 대용량 처리에 초점을 맞추어 지원하고 있다. 여기서는 기존의 RDBMS들을 발전시켜서 대용량 데이터 처리를 가능하게 하는 제품들을 살펴보도록 하겠다.

2.1 Shared Nothing 구조

대용량 데이터에 대한 인식이 기존의 TB급에서 PB, EB급으로 늘어나게 되었다. 이러한 데이터를 처리하기 위해서 분산 환경이 필요하다는 것에 큰 이견은 없을 것이다. 가장 쉽게 생각할 수 있는 방식은 Shared Nothing 구조로 하나의 클러스터로 만들어주는 것이다. 많은 벤더에서 이러한 방식으로 대용량 데이터 처리 시스템을 만들고 있다. 대표적인 주자로는 Greenplum, teradata 등이 있다. 세부 구조는 대동소이하므로 여기서는 greenplum을 기반으로 이야기를 하겠다.

Shared Nothing구조의 가장 큰 장점은 수평적 확장에 있다. Shared Everything구조에서는 global lock이 필요해진다. 또한 데이터의 동기화도 큰 문제가 된다. 이러한 문제를 효율적으로 해결하기 위하여 특별한 하드웨어(eg. InfiniBand, SSD)나 소프트웨어가 필요하게 된다. Shared Nothing 구조에서 특정 데이터에 대한 처리는 특정 노드에서 이루어지므로, 해당 데이터에 대한 global lock이 필요하지 않다. global lock이 없다는 말은 확장하기에 쉽다는 의미이다. (물론 스키마와 같은 메타 데이터에 대해서는 global lock이나 이에 해당하는 프로토콜이 필요하다.)



(그림 1) Shared Nothing 구조

대표적인 예인 Greenplum은 오픈소스인 PostgreSQL을 이용하여 구성한 Shared Nothing 분산 데이터베이스 시스템이다. OLAP 전용 시스템답게 분산 저장된 데이터를 이용한 MPP (Massively Parallel Processing)에 최적화 되어 있다. 데이터의 적제 및 조회 시 별다른 global lock 없이 접근이 가능하다. 유저가 보는 테이블은 각 노드(세그먼트)에 hash 알고리즘을 통해 분산되어 있다. 노드 추가시 hash 알고리즘에 대한 보안을 위하여 재분배를 해 주어야 한다. 또한 쿼리를 수행하는 노드는 마스터 노드로 한정적이다. Greenplum 외에도 대부분의 제품이 global schema에 대한 제약에 의해 마스터 쿼리 수행 방식을 사용하고 있다. 이는 유연한 확장을 위하여 스키마 구조를 제한하게 되는 부분을 과감히 포기하고, MPP 성능을 최적화에 초점을 맞추었기 때문으로 보인다.

Shared Nothing구조의 병렬 처리는 매우 효율적으로 이루어진다. 기본적으로 global lock이 없기 때문에 수행 계획을 잘 나누기만 한다면 각 노드에서는 아무런 방해 없이 데이터에 접근할 수 있다. greenplum의 모든 유저 쿼리는 마스터 노드에서 이루어진다. OLAP에 최적화 시킨 제품이기 때문에 쿼리의 개수 자체는 많지 않다고 볼 수 있다. 쿼리는 데이터가 있는 노드에서 실행하는 형태로 나뉘어져 수행되게 된다. MPP의 모토는 '많은 노드의 자원을 활용' 하는 것이므로 분산되어 있는 대부분의 노드가 참여한다. 물론 hash 값에 의한 prunning도 이루어진다.

Shared Nothing구조의 복구는 크게 백업을 이용한 복구와 mirroring을 이용한 복구로 나눌 수 있다. 백업을 이용한 복구는 Shared 유무와는 무관하게 지원가능하다. 다만 복구 시에 일부 노드

만 동작이 제한될 것인지, 전체 시스템이 제한될 것인지는 각 벤더마다 지원하는 스펙이 다르다. mirroring은 Shared Nothing구조에서 필수인 요소로 뽑고 있는데, 특히 대용량 데이터의 규모로 인한 백업의 어려움과, 많은 노드 수에 따른 노드 다운의 확률이 높아짐에 따라 mirroring을 통해 고가용성(HA)을 보장해 준다. 보통 mirroring factor를 2~3으로 설정하여 일부 노드가 다운되더라도 다른 노드에서 계속 작업을 진행하게 해 주거나, 최소한 데이터의 유실을 막아주게 된다.

mirroring기법은 NoSQL에서 사용하는 것보다 훨씬 복잡하다. NoSQL에서는 보통 데이터에 대한 접근이 Load와 Read로 이루어지는 반면, RDBMS의 경우는 Insert, Update, Delete, Query로 이루어지므로 한번 적재된 데이터가 변경되는 것에 대한 대비가 필요하다. 즉, 똑같은 데이터를 복사해 놓는 것뿐만 아니라, 이 중 하나가 수정되었을 때 이를 다른 mirroring에 적용할 수 있는 방법이 필요하다. 각 노드의 유닛 RDBMS에서 지원하는 mirroring을 이용하게 되므로 유닛 RDBMS의 기능에 따라 시스템의 고가용성이 결정된다.

Greenplum은 PostgreSQL에서 지원하는 mirroring과 backup을 이용하여 데이터의 영속성을 보장해 준다. 문제 발생시 backup을 통해 시스템을 재구성하던가, mirroring을 이용하여 서비스를 정상 유지되게 해 준다. 관리자가 개입하여 추가로 mirroring factor를 올려줄 수 있다.

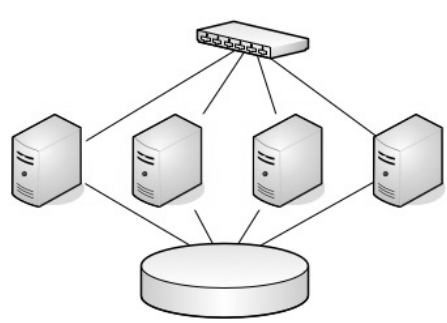
대부분의 시스템이 멀티 로우 트랜잭션을 지원해준다. RDBMS의 강력한 장점 중 하나인데, 대량의 데이터 변경에 대한 atomicity를 지켜준다. Consistency는 시스템마다 각자 다르게 지원한다. 완벽한 Consistency를 지켜주기 위해서는 각 유닛 노드의 RDBMS가 지원하는 MVCC (Multi-Version Concurrency Control) 모델뿐만 아니라 분산 트랜잭션으로 커밋한 데이터가 동시에 보여야 한다. 분산 환경에서의 MVCC모델은 다양하게 연구 되어왔지만, 실제 제품들은 성능 상의 제약 때문에 이를 완벽하게 지켜주지는 않는 편이다.

2.2 Shared Everything 구조

Shared Everything구조의 RDBMS 클러스터는 대용량 처리를 위한 또 다른 시도일 수 있다. 오라클의 Exadata는 대용량 처리를 위해서는 분산 Shared Nothing구조가 필수일거라는 생각을 깨고 공유 환경에서 대용량 데이터 처리를 위한 노력을 해 왔다. 오라클이 내세우는 Active-Active cluster인 RAC (Real Application Cluster)를 확장하여 다수의 RAC 노드를 띄우고, ASM기술을 이용하여 대용량 데이터 저장 및 처리를 가능하게 하였다.

대다수의 벤더들은 데이터 공간의 수평적 확장을 위해 Shared Nothing 구조를 택했다. Hadoop에서 촉발한 모든 NoSQL 진영이 그러했고 뒤늦게 빅데이터 시장에 뛰어든 Greenplum과 같은 RDBMS도 동일했다.

반면에 Exadata는 전통적인 Shared Nothing 구조와 완전히 다른 접근 방식을 택한다. Exadata가 택한 Shared Everything 구조는 빅데이터이전에



(그림 2) Shared Everything 구조

cluster DB 제품인 RAC에서부터 유지해온 방식이다. 다른 DB 벤더들과의 구조적인 차이점으로 인해 IBM DB2 등과 일찌감치 비교되어왔다.

Shared everything 구조는 서버로부터 디스크를 떼어 공유 디스크 장비로 옮김으로써 유연한 확장이 가능하다. CPU/Memory 가 필요하면 서버만 확장할 수 있으며, Disk가 모자라면 공유 디스크만 확장하면 된다. 모든 서버가 모든 데이터를 항상 볼 수 있기 때문에 확장 후에 데이터 재배치 등의 후처리 작업도 전혀 필요 없다. 하지만, 공유 디스크라는 고가의 장비를 전제로 하기 때문에 기본 가격도 비싸고 용량 증설에 따른 비용도 많이 드는 단점이 있다. 즉, 확장은 쉽지만 비용이 많이 들고 확장에 한계가 있다.

Exadata는 빅데이터를 저장하기 위해서 RAC에 Shared Nothing 구조의 장점을 접목한다. SAN이라는 고가의 스토리지 대신 최대 14대의 저장 서버를 두어서 확장성을 더욱 높였다. Shared Everything 구조의 근본적인 문제인 네트워크 병목은 Infiniband 라는 고성능 네트워크를 사용해서 극복하려하지만 렉 단위를 넘어선 확장에는 여전히 문제가 된다.

오라클이 택한 Shared Everything 구조는 장단점이 명확하다. 하나의 DB 서버가 모든 데이터를 볼 수 있기 때문에 Join 등의 복잡한 쿼리 속도가 빠른 반면에, 역시 모든 데이터를 봐야한다는 점 때문에 Global lock 메카니즘이 필요하다는 제약이 생긴다. RAC 규모의 cluster에서는 큰 문제가 없지만 Exadata 규모의 대형 클러스터에서는 모든 DB Server가 공통의 Global lock을 공유해야 한다는 것은 큰 병목이 될 위험이 있다. 2009년 HP 버전 Exadata를 기준으로 DB 서버를 1 랙 당 8대로 한정한 것도 공유 구조의 한계로 보인다.

RAC 시절부터 오라클은 다른 DB에 비해 장애 복구시간이 비교적 짧다고 알려졌다. 장애 서버의 데이터를 생존 서버에서 즉시 볼 수 있는 Shared Everything 구조 최대의 장점 때문이다. Exadata도 동일 구조로서 DB 서버 장애 시에 빠른 복구가 가능하다. Global lock 복구 시간 정도만이 추가 오버헤드이다. 다만 DB 서버 장애와 달리 Exadata에서 새로 추가된 Storage 서버 장애는 Shared nothing 구조와 동일하게 추가 복제 시간이 필요하다.

Exadata는 오라클 RAC를 기반으로 하기 때문에 동일한 격리 수준의 Multi-row transaction을 지원한다. 단, Read-committed를 지원하긴 하지만 한 쪽 DB 서버에서 commit된 데이터가 다른 DB 서버에서 즉시 보이지는 않는다. 물리적 시간을 완벽하게 동기화 하려면 희생해야하는 성능이 너무 크기 때문에 정확히 맞추지는 않는다.

2.3 NewSQL

DBMS의 대가 Stonebraker 가 2011년 주창한 NewSQL[3]은 NoSQL의 확장성과 RDBMS의 편의성을 합쳐보려는 시도이다. 수평적 확장, SQL 지원, Transaction 지원을 필수 요소로 꼽는 다는 점에서 우리와 관점이 같다고 할 수 있다. 대표적인 NewSQL로 꼽히는 Clustrix[5], NimbusDB[6], VoltDB[7]를 비교해 보자.

NewSQL이라고 해서 뾰족한 수가 있는 것은 아니다. Shared Nothing 구조를 기반으로 데이터를 분산시키는 세부 메카니즘에서 약간씩 차이가 날 뿐이다. VoltDB는 전통적인 hash key 기반 분산을 택하지만 디스크 기반이 아니라 메인 메모리 기반 DBMS이기 때문에 노드 당 수십GB가 최대인 메모리 크기에 데이터 크기가 한정된다는 큰 태생적 한계를 가진다. NimbusDB는 데이터를 아예 P2P 방식으로 완전 분산 시키는 방식을 택한다. 메타데이터와 Transaction 정보까지 P2P로

분산시키는 극단적인 분산 기법은 장애에 매우 강하지만 100대 이하의 중규모 클러스터에서 성능이 제대로 나올지 의문이다. Clustrix sierra는 NewSQL에 분류되면서도 오라클 Exadata와 유사한 어플라이언스 개념의 제품이다. infiniband network, SSD, NVRAM이라는 특수한 하드웨어로 구성해서 Shared Nothing 구조를 구현했다. Exadata와 유사하게 infiniband 가 병목이될 가능성이 많고 확장 비용이 큰 것이 단점으로 지적된다.

VoltDB는 데이터를 모두 메모리에 올리는 극단적인 방식을 써서라도 처리 속도를 극한으로 올리는 것을 목표로 했다. 1core-1thread-1data 구조를 택해서 Global lock을 없앤 것은 tpc-C와 같은 OLTP 부하에는 좋은 선택이다. 하지만 특정 데이터에 몰리는 부하의 경우 병렬처리가 전혀 안 된다는 단점이 있고 메모리 기반이라 진정한 빅데이터와는 거리가 있다.

Clustrix는 SQL을 각 data node에 맞게 잘라서 local SQL로 수행하는 방식으로 global lock문제를 해결한다. Exadata의 storage node가 간단한 필터링 정도만 수행하는 것에 비해 Join 등의 복잡한 작업도 data node에서 실행해서 병렬성을 높인다. 데이터를 가져와서 처리하지 않고 데이터가 있는 곳에서 처리한다는 개념은 좋으나, 어떤 데이터가 어디에 있는지 관장하는 hash map 관리가 필수적인데 이 부분에 대한 정보가 숨겨져 있어서 실제로 어느 정도 성능이 나올 수 있을지는 고개를 갸우뚱하게 한다.

K개의 복제를 통한 복구는 모든 NewSQL 시스템이 공통적으로 채용한 복구 방식이며 대동소이한 장애 복구 시간을 가질 것으로 예상된다. 여기에 VoltDB는 메모리 기반 DBMS이기 때문에 Disk 로그를 별도로 남기지 않아서 Cluster 전체 장애에 매우 취약한 단점이 추가된다. Cluster 전체 장애에 대비하기 위해 VoltDB는 주기적으로 Checkpoint를 수행하지만 비동기적으로 돌기 때문에 장애 직전까지 복구하지는 못한다. Procedure log를 남겨서 직전에 수행하던 procedure만 재실행하는 추가 기능은 procedure가 항상 같은 결과를 낸다는 보장이 없기 때문에 정합성에 문제의 소지가 있다.

NewSQL이 NoSQL과 크게 차별화 포인트로 삼은 부분이 SQL 지원이다. 모두 '대부분의 SQL 문법' 지원을 광고하고 있으나 세부 사항에서 많은 차이를 보인다. Clustrix와 NimbusDB는 MySQL 기반으로 만들어졌기 때문에 SQL99의 상당 부분을 지원하지만 MySQL이 지원하는 이상은 지원하지 않는다. Clustrix 는 좀 더 제약이 심한데, partition, trigger, stored procedure 도 지원하지 않는다. 이들 기능은 일반적인 RDBMS에서 굉장히 많이 사용하는 것들이라 NewSQL이라는 이름이 퇴색되어 보인다.

VoltDB는 자체 제작 SQL 엔진이기에 조금 다르다. 모든 SQL을 Stored procedure 형태로만 수행할 수 있으며 Stored procedure 가 끝날 때 commit, 실패하면 rollback 이라고 정의한다. 50MB 이상의 결과를 보지 못하며 inner join만 지원하고 subquery도 사용하지 못한다. 지원하는 SQL 문법을 떠나서 프로그래밍 모델적 제약이 심하기에, 어차피 응용프로그램을 재개발 해야 한다면 NoSQL에 비해 과연 편하다고 할 수 있을지 모르겠다.

NoSQL 제품들은 성능을 위해 transaction을 지원하지 않거나 single row transaction에 그치는 경향이 있다. NewSQL은 ACID 특성을 전제 조건으로 하기 때문에 모든 제품이 Multi-row transaction을 지원한다. VoltDB와 Clustrix는 Single node transaction에 최적화된 모습을 보인다. Single node transaction은 기존의 log 기반 방식으

로 구현되지만 Multi-node transaction 은 크게 언급이 없다. Multi-node 로 가면 어쩔 수 없이 2PC를 써야 하기 때문으로 짐작된다. VoltDB는 procedure가 끝날 때 commit되는 방식이기 때문에 procedure 선언부에 single/multi 여부를 명시하도록 해서 구현을 단순화 했다.

Transaction에 대해서는 NimbusDB에서 특수한 방식을 고안했다. 분산 transaction에서 발생할 수 밖에 없는 locking을 notification 방식으로 해결한다. MVCC 모델을 택했기 때문에 locking 없이 예전 데이터는 자유롭게 가져갈 수 있고, 새로운 데이터가 commit되면 예전 데이터를 가져간 노드에게 notification을 날려서 알려주는 방식이다. data 관리와 Transaction 관리를 분리해서 얻는 이점이지만, commit 자체의 속도는 많이 느릴 것으로 예상된다.

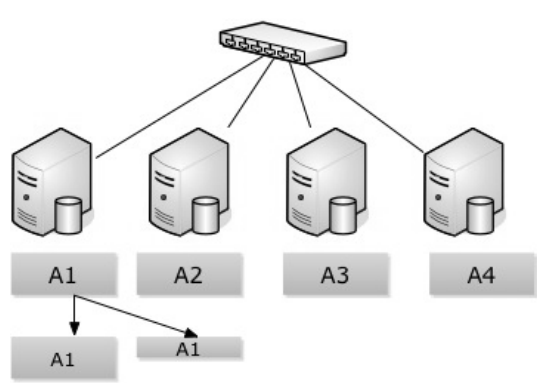
3. Tibero Massive Cluster

최근 우리는 Tibero Massive Cluster(이하 TMC)라는 제품을 개발 중이다. TMC는 범용 RDBMS인 티베로를 Shared Nothing구조로 구성한 분산 데이터베이스 시스템이다. 기존 Shared Nothing 구조 빅데이터 RDBMS의 기능상 한계점은 각 로컬 노드에서 사용하는 유닛 RDBMS의 기능과 밀접하다. 티베로는 기존의 클러스터 구성에 사용되었던 MySQL이나 PostgreSQL보다 더 다양한 기능과 성능을 제공한다. 티베로에서는 RDBMS에서 제공하는 메타 정보 관리 기술 자체를 이용하여 분산 저장된 데이터를 다루어 효율적인 활용을 할 수 있다.

TMC는 기존의 RDBMS에서 제공하던 파티션 테이블 기능을 노드 단위로 확장한다. 파티셔닝 방법 또한 분산 환경에서 발생하는 제한이 있다. 대용량 데이터에 비해 각 노드별 공간은 터무니없이 작다. 일반 RDBMS의 경우는 공간이 부족할 때 디스크를 추가하면 되지만(이는 처리할 데이터가 적기 때문이다.) 많은 노드가 있는 클러스터 시스템에서 일부 노드에만 디스크를 추가하는 것은 관리의 측면에서 좋지 않고, 범용 서버에서 사용 가능한 디스크 개수는 한계가 있다. 고로 기존의 파티셔닝 기술을 그대로 도입할 수는 없다.

예를 들어 Hash 파티셔닝의 경우에 Hash Bucket의 개수가 정해진 후, 공간 부족으로 인하여 Bucket개수를 늘려야 하는 상황이 발생한다. 이런 경우 매번 새롭게 데이터를 rebalancing하게 되면 성능 상에 문제가 될 수 있다. TMC에서는 파티셔닝을 하되, 해당 노드에서 수용가능한 공간이 다 차게 되면 다른 노드에 같은 파티션 조건을 가지는 파티션을 만든다. 이런식의 multi-level 파티셔닝을 통하여 테이블의 사이즈 제한을 없애고 있다. 또한 Bucket 개수가 처음 설계한 것보다 훨씬 많아지게 된다면 global table 전체를 rebalancing하는 기능도 제공한다.

Global Table은 한번 생성되고 나면 공간 관리 등이 자동으로 수행되게 된다. 이러한 작업에 대해서만 Locking이 필요하게 되고, 각 노드에서 로컬 쿼리와 DML을 수행하는데에는 이러한 과정이 불 필요하다. 각 노드는 Global Table의 파



(그림 3) TMC Data Partitioning

티션을 Local Table로 관리함으로써 이러한 문제를 해결한다. Local Table의 안정성을 위해 Lock이 필요한 경우는 Global이 아닌 내부적인 Locking을 통하여 해결한다.

TMC는 티베로가 보유중인 parallel query 기술을 노드 단위로 확장해서 더욱 고도화된 병렬 처리를 지원한다. 실행 계획(Execution plan)은 Global Table의 각 파티션들에 대한 정보를 포함한다. 실행 계획은 Global Table에 대한 스키마 정보를 담고 있는 스키마 마스터 노드에서 만들어진다. 하지만 직접 수행하는 것은 임의의 노드에서 수행할 수 있다. 이는 마스터 노드에 로드가 몰리는 기존 구조의 단점 해결을 위함이고 기존의 Shared Nothing 구조와 차별되는 점이다.

실행 계획이 만들어지면 이를 각 파티션을 가지고 있는 노드에 나누어 실행하게 된다. 전체 실행 계획은 매우 클 수 있으므로 slice단계를 거쳐 실행 계획을 잘게 쪼갠다. 그런 후 각 노드에 뿌려서 수행해야 하는 부분만을 수행하게 된다.

가장 효율적인 쿼리는 각 노드에 보내는 부분 실행 계획에 filter가 포함되는 경우이다. 이런 경우 네트워크 소모량도 매우 적고, 시스템의 전체 리소스를 활용하여 매우 빠른 결과를 얻을 수 있다. 뿐만 아니라 join 작업과 같은 경우는 중간 노드를 활용하여 수행한다. 네트워크 사용량이 많다고 판단된다면 테이블 조회를 수행하는 노드에서 join 작업을 함께 수행하여 최적화를 꾀한다.

TMC에서는 이러한 active-standby 모델을 이용하여 영속성을 보장해 준다. 가장 큰 장점은 정상 운행까지 걸리는 시간이 매우 짧다는 점이다. 장애로 mirroring factor가 줄어들게 되면 자동으로 factor를 복구하는 작업을 진행한다. 또한 복구중인 노드에 Loading과 Insert가 할당되지 않게 함으로써 복구 시간을 줄이고 있다.

트랜잭션 부분에서는 기본적으로 atomicity와 durability를 지원하기 위하여 분산 환경에서는 2phase-commit을 사용한다. 또한 MVCC를 지원하여 수정중인 데이터를 대기 없이 쿼리 할 수 있다. 이는 티베로의 기본적인 MVCC 모델을 똑같이 따른다.

기본적인 트랜잭션 처리는 2PC로 동작하기 때문에 이로 인한 성능 저하를 해결하기 위하여 TMC에서는 많은 최적화를 고민하고 있다. active-standby구조로 인하여 항상 mirroring factor가 지켜진다는 가정 하에서 2PC protocol은 수정될 수 있다. 2PC의 commit 단계의 경우 기존에는 Redo Log를 반드시 저장 매체에 기록해야 했지만, mirroring factor로 인하여 다른 mirroring 노드에 전달되었다는 것만 확인하게 되면 저장 매체 기록을 기다릴 필요가 없다. 2PC를 진행한 노드가 다운했을 때 다른 노드에는 commit정보가 전달되었으므로, 정상적으로 진행할 수 있다.

4. 결 론

대용량 데이터 처리 시스템에서의 RDBMS 사례들을 들어보았다. 최근 들어 DBMS가 대용량 데이터 처리에 부적합하다는 인식들을 이겨내는 성공 사례들이 종종 보인다. 또한 NoSQL진영에서도 유저 편의성을 위하여 SQL-like 언어를 제공하려고 하고, RDBMS진영에서도 대용량 분산 처리 기능에 초점을 맞추고 있다. 결국 양 진영에서 모두 하나의 그림으로 나아갈 것으로 보인다.

RDBMS는 수십 년간 진보되어온 기술이다. 각 세부 기능과 성능에서 깊은 연구와 개발이 되어 왔으며, 표준화를 이루고 있다. 기존의 아키텍처로 수용 불가능한 규모의 데이터를 처리하기 위하여 멀티노드에 기반한 분산 환경을 RDBMS에 접목시키고 있다.

또한 가상화 서비스를 위한 클라우드 시스템으로서의 연구도 활발히 진행되고 있다. 기존의 DBMS 가상화 전략은 아마존 EC2와 같은 기존 인프라에 DBMS를 설치해 주는것에 지나지 않았지만, DBMS자체의 클러스터링으로 가상화 서비스를 제공한다면 유연한 자원 제공과 규모면에서 장점을 가질 것으로 보인다.

티베로에서는 대용량 데이터 처리에 대한 큰 관심을 가지고 연구 중이며, 이를 위한 첫 번째 결과물인 TMC는 2012년 하반기에 출시될 티베로 6로 선보일 예정이다.

참 고 문 헌

[1] Oracle, Exadata, http://www.oracle.com/us/products/database/exadata/overview/index.html

[2] Greenplum, http://www.greenplum.com/products/greenplum-database/

[3] Michael Stonebraker, "New SQL: An Alternative to NoSQL and Old SQL for New OLTP Apps", http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

[4] 송용주, "대용량, 클라우드 서버 환경에서 DBMS가 고민해야할 문제", 정보과학회지 제29권 제 5호, 2011.5, page:3-107

[5] Clustrix, "A New Approach: Clustrix Sierra Database Engine", 2012, http://clustrix.com/Default.aspx?app=LeadgenDownload&shortpath=docs%2fClustrix_A_New_Approach.pdf

[6] Nimbus DB, "How it Works", http://nimbusdb.com/how_it_works.html

[7] Tim Callaghan, "VoltDB: an SQL Developer's Perspective", 2010, http://technocation.org/files/doc/2010_09_VoltDB.ppt

저 자 약 력

박 헌 영

이메일 : hypark@tibero.com

- 2003년 KAIST 전산학과 학사
- 2005년 KAIST 전산학과 석사
- 2005년~현재 (주) 티베로 빅데이터팀 팀장
- 관심분야 : DBMS, 빅데이터 처리, cloud computing

송 용 주

이메일 : yongjoo_song@tibero.com

- 2000년 KAIST 전자전산학과 학사
- 2002년 KAIST 전자전산학과 석사
- 2007년 KAIST 전자전산학과 박사
- 2007년~현재 (주) 티베로 빅데이터팀 재직 중
- 관심분야 : DBMS, 클라우드, 대용량 데이터, 파일시스템, 분산시스템