TIBERO

TECHNICAL WHITE PAPER

Tibero InfiniData
Overview

October 2012

TIBERO

목차

1.개요
2.Bigdata 와 RDBMS
2.1BigData 열풍과 RDBMS
2.2 BigData 처리를 위한 요구사항
2.2.1. 수평적 확장
2.2.2 병렬 처리
2.2.3. 짧은 장애 복구 시간 (무정지 서비스)
2.2.4. SQL 지원
2.2.5. Multi-row Transaction 지원
2.3. RDBMS 로의 회귀
3. Tibero InfiniData
4. 결론

1. 개요

기존 RDBMS 로는 처리할 수 없는 BigData 가 넘치고 있다. 몇 대 규모의 분산 RDBMS 기술로 버티던 Database 시장은 SNS, Web log 등 BigData 의 등장으로 인해 균열이 생기고 말았다. 그 잠깐의 틈새를 Hadoop, MongoDB, Voldemort 와 같은 NoSQL 에서 차지한다. 드라마틱한 수평적 확장성과 싼 가격으로 NoSQL 제품이 일시적인 대안으로 떠오른 것은 사실이지만, NoSQL 제품은 SQL 과 RDBMS 기능에 익숙해진 개발자들을 완전히 흡수하지 못했다. 기존 RDBMS 벤더들도 Exadata, greenplum, HANA 와 같은 BigData 확장 솔루션을 출시했서 NoSQL 의 확산을 견제하는 상황이다.

Tibero InfiniData 는 BigData 를 처리할 수 있으면서도 기존 RDBMS 의 사용성을 그대로 지원하는 새로운 개념의 제품이다. 범용 RDBMS 인 티베로를 Shared Nothing 구조로 구성해서 기존의 클러스터 구성에 사용되었던 MySQL 이나 PostgreSQL 보다 더 다양한 기능과 성능을 제공하면서도 탁월한 수평적 확장을 지원한다. RDBMS 에 좌절하고 NoSQL 에 실망한 개발자와 DBA 들에게 Tibero InfiniData 는 탁월한 대안이 될 것이다.

2.Bigdata 와 RDBMS

2.1BigData 열풍과 RDBMS

바야흐로 BigData 전국시대다. 구글이 만든 도화선에 트위터, 페이스북이 불을 붙이고 다양한 업체가 그 불에 뛰어들고 있다. 꺼질 줄 모르고 있는 BigData 라는 큰 화두에 접근하는 방식은 다양하지만 Database 라는 측면에서만 보자면 크게 NoSQL 과 RDBMS 로 방법론을 나눌 수 있을 것이다. 기존 RDBMS 에서 감당할 수 없는 크기의 Data 를 위해 데이터 마이닝 회사에서는 Hadoop, MongoDB, Voldemort 와 같은 NoSQL 에 기반한 데이터 분석 솔루션을 내놓았다. 기존 RDBMS 벤더들은 Exadata, greenplum, HANA 와 같은 BigData 확장 솔루션을

출시했다. NoSQL 과 RDBMS 의 장점만을 모으려는 시도인 NewSQL 이란 개념까지 등장한다.

이 혼돈의 와중에 우리는 Tibero InfiniData 라는 새로운 개념의 제품을 선보인다.

Tibero InfiniData 를 이해하려면 왜 NoSQL 이 등장했는지, 왜 NoSQL 만으로는 안되며 RDBMS 가 다시 등장하고 있는지부터 먼저 살펴볼 필요가 있다.

2.2 BigData 처리를 위한 요구사항

2.2.1. 수평적 확장

어디부터 BigData 불러야 하는가? 기준에 대해서는 여러 가지 의견이 분분하지만 절대적인 수치가 아니라 '기존 시스템에서 저장/처리하기 힘든 크기' 라는 것에 대체적으로 동의한다. 당연히 디스크의 수평적 확장은 BigData 처리의 필수 요건이 된다. 한 발 더 나아가서, 디스크 추가의 속도와 편의성도 중요한 관심사이다. 디스크를 증설 하려면 전체 시스템을 멈춰야 한다거나, 증설 작업 후에 데이터를 재배치해야만 한다면 확장성이 충분히 제공된다고 할 수 없다. 오라클, MySQL 등으로 대표되는 기존 RDBMS 는 이부분에서 매우 취약했으며, NoSQL 이 등장하게된 메인 이슈라고 할 수 있다.

물론 기존 RDBMS 들에도 이미 분산 DBMS 기술이 탑재되어 있다. 다만 Shared everything 형태의 분산 DBMS 기술은 고가의 SAN 이 전제되어 가격면의 확장성이 떨어지며, Shared nothing 형태의 기존 기술은 증설 및 장애 복구가 복잡하다는 단점 때문에 실질적인 확장성이 매우 낮았다.

2.2.2 병렬 처리

BigData 의 단순 저장에는 굳이 DBMS 가 필요하지 않다. 빠른 분석 처리가 필수 요건이지만 디스크의 수평적 확장에 따라 CPU 와 메모리도 수평적으로 확장되므로 처리 속도를 쉽게 향상시키기 어렵다. 특히 OLAP 부하의 핵심인 aggregation/analytic 함수에 대한 극도의 병렬 처리가 필요하다. 기존 단일 서버에서 사용하던 알고리즘을 버려야 할 수도 있다. Global critical section 을 얼마나 줄여서 성능을 높일 수 있는가가 쟁점 사항이다. Hadoop 으로 대표되는 NoSQL 기술은 극도의 병렬처리를 지원하긴 하지만 기존 RDBMS 와 같은 복잡한 operation 자체를 지원하지 않기 때문에 응용프로그램에 병렬처리의 책임을 대부분 떠넘기고 있다. 실제로 다수의 고객들은 기존 RDBMS 가 /*+ PARALLEL(t, 8) */ 키워드 하나로 지원하던 병렬처리를 서비스 단에서 구현하기 위해 다수의 개발 인력을 투입하는 어려움을 겪는다.

2.2.3. 짧은 장애 복구 시간 (무정지 서비스)

장애 복구 기능은 상용 DBMS 라면 필수 요소이기에 따로 강조할 필요는 없다. 특히 수평적 확장이 거듭된 BigData 환경에서는 특정 서버나 디스크에 장애가 발생할 확률이 기하 급수적으로 상승하기 때문에 장애 복구는 더 이상 가끔 생기는 이벤트가 아니다. 이 경우 정확한 데이터 정합성 복구 뿐만 아니라 복구 시간도 중요한 요소가 된다. 특히 분산 시스템에서 복구 시간과 평소 성능은 반비례하는 경향을 보이기 때문에 중요 설계 포인트이다. 서비스 전체 정지 시간을 최소화하거나, 부분 복구가 가능하도록 시스템을 설계하는 것이 필요하다.

분산시스템 CAP 이론에 따르면, 무정지 서비스를 위해서는 데이터 정합성을 희생하거나 병렬성을 희생하는 수 밖에 없다. 대부분의 NoSQL 들은 Eventually Consistency 라는 개념을 도입해 데이터 정합성을 희생해 왔기 때문에 중요한 OLTP 업무에 사용되지 못하는 한계에 부딪혀 왔다.

2.2.4. SQL 지원

RDBMS 가 오랜 기간 사용되어 왔기 때문에 대부분의 Data Mining 알고리즘들은 SQL 로 작성되어 있다. NoSQL 을 사용하는 많은 기업들이 이구동성으로 가장 힘들었던 것으로 손꼽는 것은 SQL 을 MapReduce 프로그램으로 변환해줄 프로그래머를 구하는 것이다. SQL 은 절차적 언어가 아니라 얻고 싶은 결과를 묘사하는 언어이기 때문에 MapReduce 에 직접 1:1 대응되지도 않고 효율적으로 변환하기도 힘들다.

SQL 지원이라는 요구사항은 지원/미지원으로 딱 잘라 나누어지지는 않는다. Dremel, Hive 와 같이 SQL-like

언어를 지원함으로써 MapReduce 의 한계를 우회하기도 하고, 설계상의 어려움 때문에 SQL 의 일부 문법만 지원하기도 한다. 형태는 다르지만 NoSQL 제품들도 MapReduce API 형태가 아닌 High-level 언어의 필요성을 절감하고 저마다 맞는 옷을 찾는 중이다. 하지만 이미 오랫동안 다듬어져 쓰인 SQL 만큼 data 처리에 적합한 언어는 없을 것이다.

2.2.5. Multi-row Transaction 지원

Multi-row 트랜잭션을 지원하지 않더라도 서비스를 작성할 수 없는 것은 아니다. 서비스 개발자가 매번 정합성 체크&해결 코드를 삽입해야하는 비효율성과 불편함이 따를 뿐이다. 트랜잭션 지원을 느슨하게 하는 대신 성능 향상에 중점을 둔 NoSQL database 는 초창기에는 좋은 설계 선택이었지만, 이제는 NoSQL database 가 쓰일 수 있는 영역을 규정짓는 한계선이 되었다.

기존 RDBMS 들은 일반적으로 2-phase commit 을 통해 Multi-row transaction 을 지켜주지만 시스템 별로 이를 사용하지 않거나 향상된 프로토콜로 지원해 주기도 한다. consistency 는 시스템마다 각자 다르게 지원한다. 완벽한 Consistency 를 지켜주기 위해서는 각 유닛노드의 DBMS 가 지원하는 MVCC 모델 뿐만 아니라 분산 트랜잭션으로 커밋한 데이터가 동시에 보여야 한다. 분산 환경에서의 MVCC 모델은 다양한 연구가 되어왔지만, 실제 제품들은 이를 완벽하게 지켜주지는 않는 편이다. 이를 지켜주기에는 성능상의 제약이 크기 때문이다.

"Multi-row 트랜잭션을 어느 정도 격리 수준에서 어느 정도 성능으로 지원하는가" 하는 기준은 해당 RDBMS 가 쓰일 수 있는 시장의 규모를 결정할 것이다.

2.3. RDBMS 로의 회귀

몇 대 규모의 기존 분산 RDBMS 기술로 BigData 를 처리할 수 없는 것은 명확하다. 드라마틱한 수평적 확장성과 싼 가격으로 NoSQL 제품이 일시적인 대안으로 떠오른 것도 분명한 사실이다. 하지만 NoSQL 제품은 SQL 과 RDBMS 기능에 익숙해진 개발자들을 완전히 흡수하지 못했다. 패턴, 프레임워크를 이용한 코드 최소화를 추구하는 요즘 개발 패러다임에도 역행한다. 비정형 데이터만을 다루는 작은 시장을 제외한다면, BigData 를 처리할 수 있으면서도 기존 RDBMS 의 사용성을 그대로 지원하는 제품이 필요한 시점이다.

3. Tibero InfiniData

최근 티베로에서는 Tibero InfiniData 라는 제품을 개발 중이다. Tibero InfiniData 는 범용 RDBMS 인 티베로를 Shared Nothing 구조로 구성한 분산 데이터베이스 시스템이다. Shared Nothing 구조의 기능상의 한계점은 각 노드의유닛 DBMS 의 기능과도 밀접하다. 티베로는 기존의 클러스터 구성에 사용되었던 MySQL 이나 PostgreSQL 보다 더 다양한 기능과 성능을 제공한다. 티베로에서는 RDBMS 에서 제공하는 메타 정보 관리 기술 자체를 이용하여 분산 저장된 데이터를 다루어 효율적인 활용을 할 수 있다.

Shared Nothing 구조의 클러스터 시스템의 가장 좋은 점은 수평적 확장이 가능하다는 데에 있다. Shared Everything 구조로 구성한다면 가격적인 측면에서 많은 부담이 된다. 같은 용량을 구성하는 비용 차이가 나게 된다. 또한 Shared Everything 구조는 Shared Nothing 구조에 비해 더 많은 locking 을 필요로 한다. RAC 의 경우 내부 동기화를 필요로 하기 때문에 사실상 노드 숫자를 늘리는 데에 한계가 있다. Shared Nothing 구조는 저장 매체와 각 유닛노드의 가격이 합리적일 뿐만 아니라, 불필요한 locking 을 제거할 수 있고, 각 노드에서는 경합 없이 disk IO, 네트워크 Bandwith 등을 모두 활용할 수 있다.

가) 수평적 확장

티베로는 데이터 저장에 대한 메타 데이터를 효율적인 구조로 관리하며, 이에 대한 캐쉬도 유지한다. 이러한 캐쉬와 내부 구조를 통하여 더 빠르게 메타 데이터 접근이

가능하다. global schema 기술은 시스템의 수평적 확장을 위하여 가장 중요한 부분이다. 수 PB 단위의 데이터를 어떻게 저장하느냐는 가장 기본적인 기능이며, 데이터를 어떻게 분산하느냐에 따라, 병렬 처리의 성능이 결정되기 때문이다.

기존의 RDBMS 에서 제공하던 파티션 테이블 기능을 노드 단위로 확장한다. 이미 티베로에 있는 기술이다. 파티셔닝 방법 또한 분산 환경에서 발생하는 제한이 있다. 대용량 데이터에 비해 각 노드별 공간은 터무니없이 작다. 일반 RDBMS 의 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 Bucket 의 개수가 정해진 후, 공간 부족으로 인하여 Bucket 개수를 늘려야 하는 상황이 발생한다. 이런 경우 매번 새롭게 데이터를 rebalancing 하게 되면 성능 상에 문제가 될 수 있다. Tibero InfiniData 에서는 파티셔닝을 하되, 해당 노드에서수용가능한 공간이 다 차게 되면 다른 노드에 같은 파티션 조건을 가지는 파티션을 만든다. 이런식의

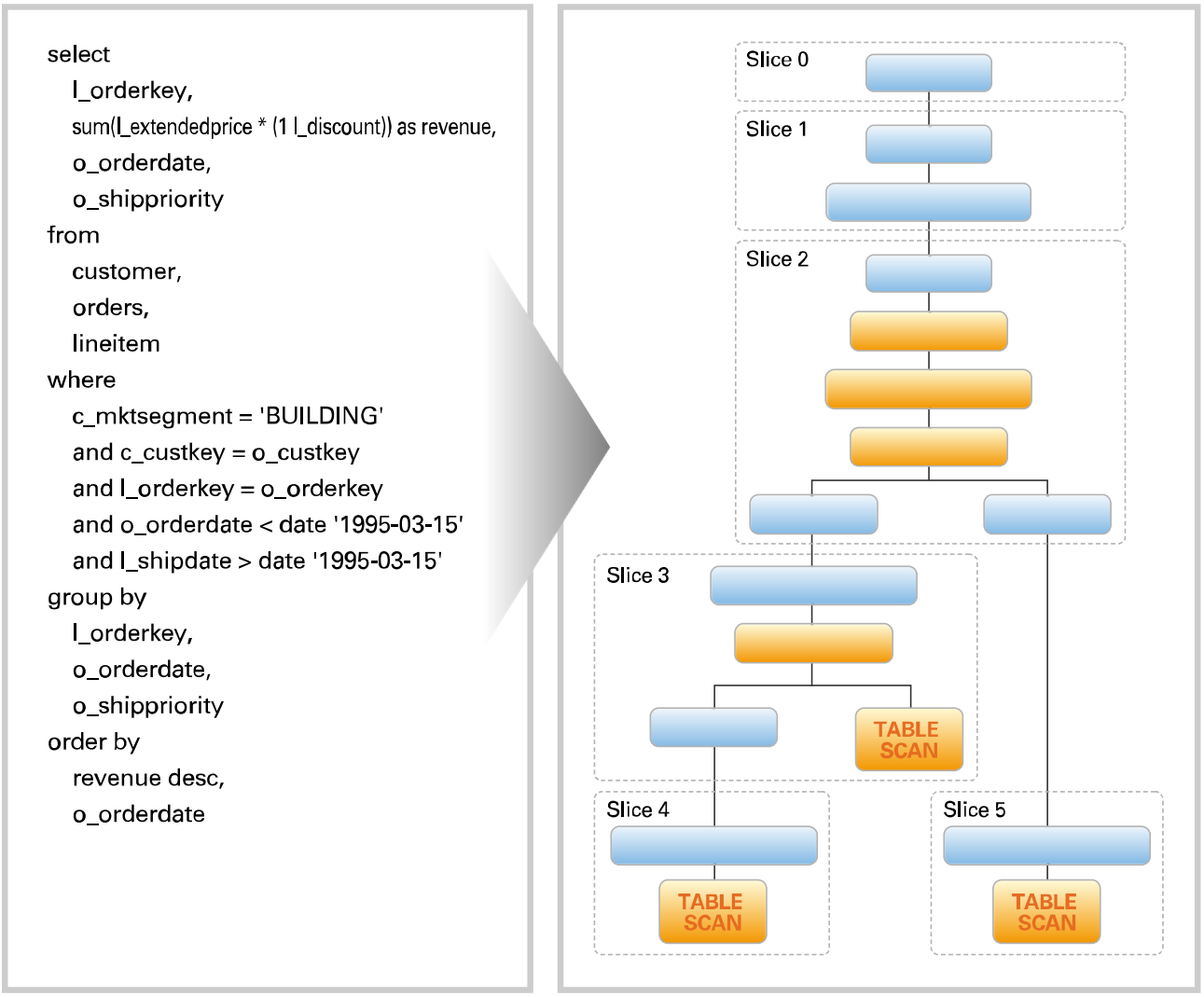

그림 설명

multi-level 파티셔닝을 통하여 테이블의 사이즈 제한을 없애고 있다. 또한 Bucket 개수가 처음 설계한 것보다 훨씬 많아지게 된다면 global table 전체를 rebalancing 하는 기능도 제공한다.
Global Table 은 한번 생성되고 나면 공간 관리 등이 자동으로 수행되게 된다. 이러한 작업에 대해서만 Locking 이 필요하게 되고, 각 노드에서 로컬 쿼리와 DML 을 수행하는데에는 이러한 과정이 불 필요하다. 각 노드는 Global Table 의 파티션을 Local Table 로 관리함으로써 이러한 문제를 해결한다. Local Table 의 안정성을 위해 Lock 이 필요한 경우는 내부적인 Locking 을 통하여 해결한다.
내부 메타 정보 관리 방식의 확장을 통해 유저 입장에서 Tibero InfiniData 는 단지 큰(Big) RDBMS 라고 느끼게 케이스들을 제외하고는 기존에 돌던 어플리케이션을 모두 돌릴 수 있다는 장점이 있다.

나) SQL 병렬 처리

티베로의 보유중인 parallel query 기술을 노드 단위로 확장한 개념이다. 사용자는 Tibero InfiniData 임의의 노드에 접속하여 SQL 수행을 요청할 수 있다. SQL 요청을 받게 되면 Global Table 에 대한 스키마 정보를 담고 있는 스키마 마스터 노드에 수행계획을 요청한다. 마스터 노드에서는 SQL 대해 Tibero InfiniData query optimizer 를 이용하여 수행계획을 만들게 된다. Tibero InfiniData query optimizer 는 tibero optimizer 의 확장판으로 tibero 에서는 노드 하나의 통계정보를 이용했다고 한다면 Tibero InfiniData 에서는 전체노드에

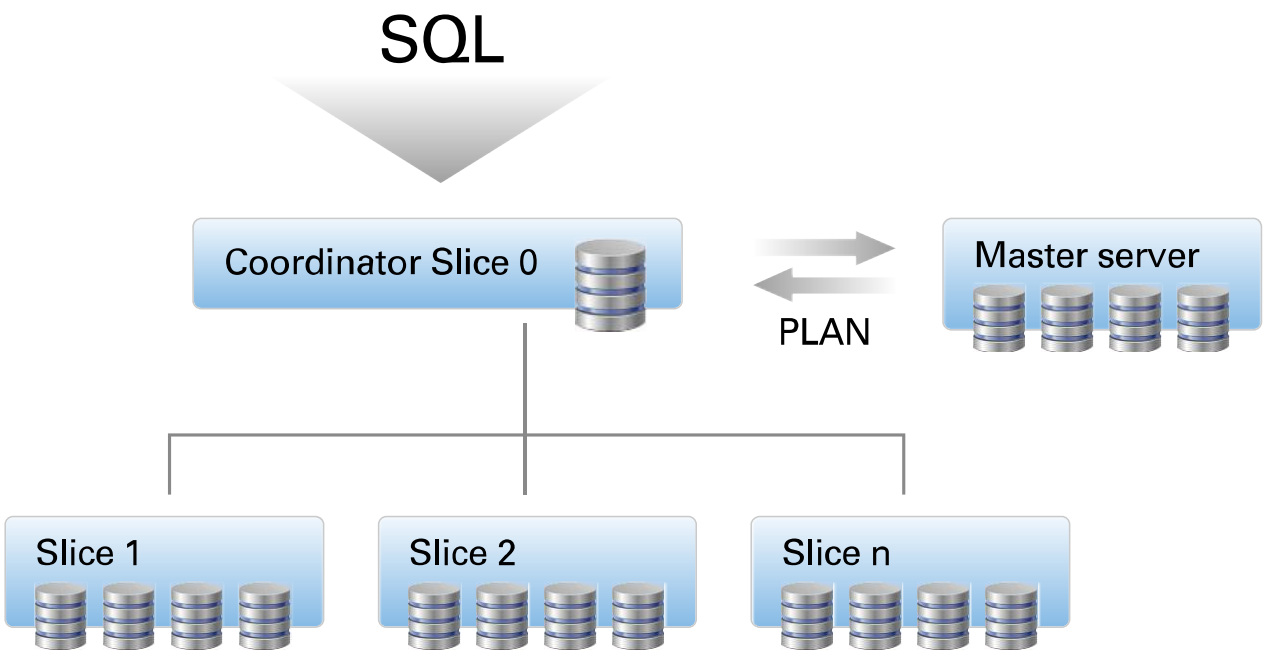

그림 설명

된다. 일반적인 테이블 접근 방식을 통해 접근할 수 있다. 기존에 하던 방식으로 SQL 을 통해 접근 가능하다. 일부

대한 통계정보를 이용한다. 또한 파티션 정보와 데이터를 전송하는 네트워크 코스트를 추가한 비용기반 최적화기(cost based optimizer)이다. 이와 같은

구조는마스터 노드에 로드가 몰리는 기존 구조의 단점 해결을 위함이고 기존의 Shared Nothing 구조와 차별되는 점이다.

마스터에서 만들어진 수행계획을 받아서 전체 수행계획을 노드 하나에서 수행할 수 있는 부분(slice)로 나누어서 slice 를 수행할 노드를 선정하게 된다. 노드 선정기준은 slice 마다 다르겠지만 보편적으로는 자원상황이 넉넉한 노드를 우선적으로 선정한다. 노드 선정이 되면 각각의 slice 에 대한 실행계획을 나누어 해당 노드로 보내서 수행하게 된다.

가장 효율적인 쿼리는 각 노드에 보내는 부분 실행 계획에 filter 가 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.

다) 장애 복구

일반적인 클라우드 시스템에서는 데이터의 영속성 보장을 위하여 mirroring 기술을 사용한다. 백업-리커버리를 하기에는 너무 많은 데이터가 존재하며, 리커버리를 위한 추가 정보를 남기는 것도 쉬운 일이 아니다.
RDBMS 에서는 active-standby 모델의 replication 기술을 이용하여 고가용성을 보장해 준다. 티베로에서 제공하고 있던 active-standby replication 은 많은 장점이 있다. 물리적으로 동일하게 구성되어 있으므로 active 노드가 다운되더라도 standby 노드가 짧은 복구 시간 후 정상 서비스를 진행할 수 있다.
Tibero InfiniData 에서는 이러한 active-standby 모델을 이용하여 영속성을 보장해 준다. 가장 큰 장점은 정상 운행까지 걸리는 시간이 매우 짧다는 점이다. mirroring factor 가 줄어들게 되면 자동으로 factor 를 복구하는 작업을 진행한다. 노드 단위의 복구이기 때문에 복구에 필요한 시간이 문제될 수는 있지만 1GBps 네트워크 환경에서는 1~2 시간 정도에 factor 를 정상 복구할 수 있다. 그동안에 정상적인 운영이 가능하다.
RDBMS 의 경우는 기본적으로 Redo Log 를 이용하여 데이터베이스를 복구하는 시간이 필요하다. 이 과정을 최소한으로 하기 위하여 mirroring factor 가 줄어든 노드에 가급적 DML 이 발생하지 않도록 한다. update 나 delete 와 같은 DML 은 어쩔 수 없지만 대용량 시스템의 특성상 insert 나 Loading 이 더 많이 발생한다. 해당 노드에 위와 같은 작업을 할당하지 않게 함으로써 복구 시간을 줄이고 있다.

라) SQL 지원

big data 처리 시장에서 가장 큰 장점을 취할 수 있는 부분이 SQL 사용 여부이다. 요즘의 추세는 NoSQL 진영에서도 SQL-like 언어들을 추가로 사용하여(Hive, etc) 사용자 편의성에 초점을 맞추고 있다. 허나 이미 RDBMS 는 표준화되고, 익숙하고, 기능이 충실한 SQL 을 지원하고 있다. Tibero InfiniData 에서도 기존 티베로에서 제공하던 SQL 스펙을 대부분 지원한다. SQL92 표준을 대부분 지켜준다. PSM 기능도 동등하게 제공하여 활용성이 매우 높다.

마) multi-row TX 지원

RDBMS 의 큰 장점중의 하나이다. 기본적으로 atomicity 와 durability 를 지원하기 위하여 분산 환경에서는 2phase-commit 을 사용한다. 이런 경우 prepare 단계와 commit 단계 사이에 in-doubt status 가 발생할 수 있는데, Tibero InfiniData 의 경우는 이를 자동으로 해결해 줄 수 있다. 단지 상위에서 서로 다른 시스템을 묶은 것이 아니라, 내부적으로 티베로를 클러스터링하고, 각 노드에서의 로컬 트랜잭션의 상태 조회가 가능하기 때문에 시스템에서 이를 해결해 줄 수 있다. 또한 MVCC 를 지원한다. 수정중인 데이터를 대기 없이 쿼리 할 수 있다. 이는 티베로의 기본적인 MVCC 모델을 똑같이 따른다.

기본적인 트랜잭션 처리는 2PC 로 동작하기 때문에 성능상의 문제를 짚어볼 필요가 있다. RDBMS 는 데이터 수정의 atomicity 를 매우 중요하게 생각하기 때문에 분산 트랜잭션은 2PC 으로 동작하게 된다. 대용량 데이터를 적재하는 트랜잭션의 경우는 크게 문제되지 않는다. 전체 트랜잭션 수행시간에 비해 commit 에 걸리는 시간은 미미하다. OLTP 환경에서는 트랜잭션이 하는 일 자체가 적기 때문에 상대적으로 commit 시간이 중요하다. 특히 반응 시간(response time)이 중요한 작업의 경우 문제제기가 될 수 있다.

이를 해결하기 위하여 Tibero InfiniData 에서는 많은 최적화를 고민하고 있다. active-standby 구조로 인하여 항상 mirroring factor 가 지켜진다는 가정 하에서 2PC protocol 은 수정될 수 있다. 최적화 방안 중 하나는 다음과 같다. 2PC 의 commit 단계의 경우 기존에는 Redo Log 를 반드시 저장 매체에 기록해야 했지만, mirroring factor 로 인하여 다른 mirroring 노드에 전달되었다는 것만 확인하게 되면 저장 매체 기록을 기다릴 필요가 없다. 2PC 를 진행한 노드가 다운했을 때 다른 노드에는 commit 정보가 전달되었으므로, 정상적으로 진행할 수 있다.

또한 엄격한 2PC 를 지키지 않고, commit wait 자체를 기다리지 않는 방식도 제공할 의향이 있다. 이는 prepare 단계인 데이터를 만났을 때 commit 이 곧 올 것이므로 짧은 시간 기다리는 방식으로 해결할 수 있다. 대부분의 케이스에서 2PC 는 성공하므로 이러한 접근 방식이 응답 시간을 짧게 하는 해결책이 될 수 있을 것이다.

4. 결론

대용량 데이터 처리 시스템에서의 DBMS 사례들을 들어보았다. 최근 들어 DBMS 가 대용량 데이터 처리에 부적합하다는 인식들을 이겨내는 성공 사례들이 종종 보인다. 또한 NoSQL 진영에서도 유저 편의성을 위하여 SQL-like 언어를 제공하려고 하고, 이러한 방향으로 나아가는 경향이 보인다. 결국 양 진영에서 모두 하나의 그림으로 나아갈 것으로 보인다.

DBMS 는 수십 년간 진보되어온 기술이다. 각 세부 기능과 성능에서 깊은 연구와 개발이 되어왔으며, 표준화를 이루고 있다. 기존의 아키텍처로 수용 불가능한 규모의 데이터를 처리하기 위하여 멀티노드에 기반한 분산환경을 DBMS 에 접목시키고 있다.

분산 DBMS 영역은 학계에서도 오래전부터 논의되고 있던 분야이다. 단순히 동형, 또는 이형의 DBMS 를 클러스터링을 하고 분산 처리하는 것을 넘어서서 내부 구조를 통합하고 실제 제품으로 인정받기 위한 다양한 기능들이 추가되고 있다.

또한 가상화 서비스를 위한 클라우드 시스템으로서의 연구도 활발히 진행되고 있다. 기존의 DBMS 가상화 전략은 아마존 EC2 와 같은 기존 인프라에 DBMS 를 설치해 주는것에 지나지 않았지만, DBMS 자체의 클러스터링으로 가상화 서비스를 제공한다면 유연한 자원 제공과 규모면에서 장점을 가질 것으로 보인다.

문서번호 넣으실 곳