P*TIME: 제2세대 고성능 메인 메모리 DBMS

차상균, 김기홍, 유승원, 송창빈, 이주창, 황상용, 권용식, 권근주, 박장호, 이용식, 이윤원, 이용희, 최기린

서울대학교 전기컴퓨터공학부

{chask, next, mokmon, tsangbi, juch, syhwang, yskwon, icdi, jhpark, doogie, kelovon, yhlee, giraffe}@kdb.snu.ac.kr

P*TIME: A 2nd-Generation High-Performance Main-Memory DBMS

Sang K. Cha, Kihong Kim, Jangho Park, Yongsik Kwon, Changbin Song, Sangyong Hwang, Seungwon Yoo,

Keunjoo Kwon, Juchang Lee, Yongsik Lee, Yoonwon Lee, Yoonghee Lee, Kirin Choi

School of Electrical Engineering and Computer Science, Seoul National University

요 약

최근 인터넷 및 이동 통신이 발달하면서 많은 사용자를 동시에 서비스할 수 있는 고성능 데이터베이스 서버가 필요하게 되었다. 또한 DRAM의 가격이 하락하고 64bit 어드레싱이 일반화되어 쉽게 수십 GB의 메모리와 서버 플랫폼을 갖추게 되어 메인 메모리 DBMS에 대한 관심이 높아지고 있다.

본 논문에서는 2세대 고성능 메인 메모리 DBMS인 P*TIME을 소개한다. P*TIME은 CPU에 비해 상대적으로 느린 메모리 성능, 저가의 멀티 프로세서 시스템 등의 현재 하드웨어 아키텍처를 고려한 인덱스 및 동시성 제어 기법을 활용하였고, differential logging 을 사용하여 logging과 회복을 각각 병렬적으로 수행할 수 있다. 이로 인해 검색과 갱신에서 매우 높은 성능을 나타낸다. 또한 간단한 구조로 인하여 시스템 튜닝과 커스터마이징이 용이하며, 다양한 응용 프로그램 서버 구조를 수용할 수 있다.

디렉토리 서버로서 P*TIME의 성능을 실험한 결과 SUN Enterprise 6500 서버에서 내장 디렉토리 서버 환경으로 60~70만 TPS의 검색 성능을 보이며 10만 TPS 이상의 갱신 성능을 보인다. 또한 클라이언트/서버 환경에서도 10만 TPS 이상의 검색 성능을 나타내었다.

1. 서론

최근 인터넷 및 이동 통신이 발달하면서, 주요 통신 사업자나 유명 웹사이트의 경우 초당 수천에서 수만 명의 사용자를 동시에 서비스할 수 있는 고성능 데이터베이스 서버가 필요하게 되었다. 현재 널리 사용되는 디스크 기반 DBMS는 이와 같은 요구 조건을 만족하지 못하므로 고성능 하드웨어를 이용하거나 서버를 여러 대 운용하는 방법을 택하고 있다. 그러나 이 방법은 경제적으로 부담도 크고, 가격에 비해 만족할만한 성능을 얻지 못하여 구조가 간단하면서 높은 성능을 낼 수 있는 메인 메모리 DBMS에 대한 관심이 높아지고 있다.

또한 DRAM 가격이 지속적으로 하락하여 최근에는 서버용 메모리 모듈 가격이 $1000/GB이하로 형성되었고 64bit 어드레싱이 일반화되면서 수십 GB의 메모리를 장착한 멀티 프로세스 서버 플랫폼을 쉽게 갖출 수 있게 되어 메인 메모리 DBMS가 현실적인 대안이 되고 있다.

본 논문에서는 단순히 DB를 메인 메모리로 옮긴 1세대 메인 메모리 DBMS와 다른 2세대 고성능 메인 메모리 DBMS인 P*TIME(Highly Parallel Transact In Memory Engine)을 소개한다. P*TIME은 제한된 하드웨어 cache를 고려한 인덱스[1]와 동시성 제어 기법을 사용하여 멀티 프로세서 환경에 우수한 scalability를 가지며 differential logging 기법[2]을 이용하여 logging과 회복을 완전히 병렬적으로 수행할 수 있으므로 갱신 성능이 뛰어나다. 또한 P*TIME은 현재 하드웨어 아키텍처에서의 메모리 병목현상[3]을 줄이기 위하여 메모리 복사를 최소화하도록 설계되었다. 그리고 간단한 구조를 사용하여 시스템의 튜닝과 커스터마이징을 용이하게 하는 한편, 내장형 응용 프로그램 서버와 클라이언트에서 활용할 수 있는 다양한 API를 제공한다.

이 논문은 다음과 같이 구성된다. 2절에서 관련 연구에 대해 간략히 기술하며 3절에서 P*TIME의 구조에 대해 설명한다. 4절에서 P*TIME의 내장형 응용프로그램 서버와 클라이언트/서버 환경에서의 성능을 측정하고, 5절에서 결론을 내도록 한다.

2. 관련 연구

메인 메모리 DBMS 기술은 1980년대 후반~1990년대 초반의 기초 연구 단계를 거쳐 Dali[4], Monet[5]과 같은 시작품이 개발되었으며 현재 TimesTen[6] 등의 몇몇 제품이 상용화되어 있다. 국내에서는 본 연구실에서 1993년부터 메인 메모리 저장 시스템인 M²RTSS 프로젝트를 시작하여, 1995-1996년, ETRI의 연구개발비를 지원받아 Mr.RT를 개발하였고, 이에 확장성을 추가한 Xmas[7][8]의 시작품을 개발하였다. 최근 국내에서 상용화가 시도되고 있는 알티베이스[9], 심포니 RTDB[10]는 본 연구실에서 개발하여 ETRI에 기술 이전한 Mr.RT를 기반하고 있다.

그러나 TimesTen, Mr.RT 등의 기존의 1세대 메인 메모리 DBMS들은 데이터베이스를 메모리에서 관리한다는 점 외에 특별한 점이 없다. 반면, 메인 메모리 DB에 대한 최근의 연구 결과들은 메인 메모리 DBMS가 디스크 기반 DBMS에 비해 상당한 성능을 내기 위해서는 색인, 동시성 제어, 로깅 및 회복에 있어서 메인 메모리 DB에 최적화된 알고리듬을 필요로 한다는 것을 입증하고 있다. P*TIME은 고전적인 디스크 기반 DB 알고리즘들을 사용하는 제 1세대 메인 메모리 DBMS와는

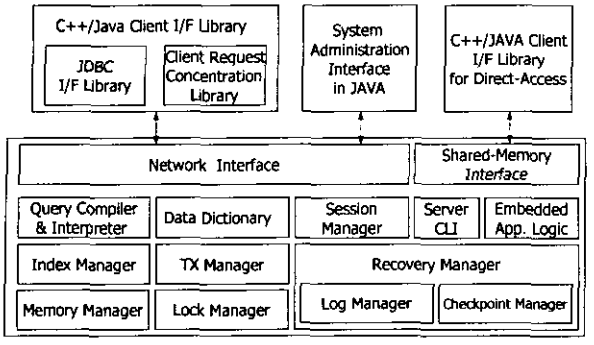

그림 1. P*TIME 모듈

달리, 지난 8년간 본 연구실의 경험과 메인 메모리 DB 분야의 최신 연구 결과를 바탕으로 새로이 설계 구현된 제2세대 메인 메모리 DBMS이다.

3. P*TIME 구조

3.1. 개요

P*TIME은 데이터베이스를 공유 메모리에서 관리한다. 삽입, 삭제, 갱신이 발생할 경우 그 내용을 디스크에 logging하여 시스템이 crash되었을 때 복구할 수 있도록 한다. 그리고 특정 주기마다 메모리에 존재하는 데이터베이스를 디스크로 백업하는 checkpointing을 수행하게 된다.

그림 1는 P*TIME의 커널 모듈과 클라이언트 모듈을 자세히 그린 것이다. Memory manager는 공유 메모리의 할당 및 분배를 관리하며, logging과 checkpointing을 담당하는 recovery manager와 함께 *P*TIME* 커널의 가장 하위 부분이 된다. Lock manager는 데이터와 인덱스의 모든 lock을 관리하며 이들 하위 모듈들을 이용하여 인덱스를 관리하는 index manager, 질의를 수행하는 query compiler 와 query interpreter가 있다. 그리고 시스템의 타입과 메타데이터를 관리하기 위해 *data dictionary*가 존재하며 트랜잭션을 관리, 조정하는 transaction manager와 클라이언트와의 인터페이스를 제공하는 session manager가 존재한다.

P*TIME의 커널 모듈들은 멀티 프로세서 환경에서 성능을 극대화하기 위하여 mutex 등의 latch 및 전역 변수의 사용을 줄였다. 또 질의를 수행할 때 데이터의 복사를 최소화하여 메인 메모리 DBMS에서 발생 가능한 메모리에서의 병목현상을 줄이도록 구현되었다. 그리고, 시스템 구조를 최대한 단순화하여 시스템의 튜닝 및 커스터마이징이 용이하도록 설계되었다.

3.2. 메모리 관리 및 동시성 제어

P*TIME에서는 공유 메모리를 세그먼트로 관리하며 각 세그먼트는 *slotted* 페이지로 이루어져 있다. 각 슬롯에는 *OID*가 부여되며 OID 변환의 오버헤드를 없애기 위하여 가상 메모리 어드레스를 직접 OID로 사용하였다.

동시성 제어는 lock manager에서 관장한다. 메인 메모리 DBMS에서는 한 트랜잭션의 수행 시간이 짧기 때문에 잠금 유지 시간이 짧다. 그러나 동시에 다수의 트랜잭션이 수행되는 멀티 프로세서 환경에서는 잠금 충돌의 매우 비용이 크게 된다. 그러므로 P*TIME에서는 레코드 레벨 잠금을 사용하였다.

3.3. 회복 관리

P*TIME은 logging을 위하여 국내외 특허를 출원한 differential logging[2] 방식을 사용한다. 이 방식은 갱신이 발생할 경우 갱신 전 내용과 후의 내용의 차이를 로깅하여 재시작 시에 순서와 관계없이 로그를 반영하여도 DB를 회복할 수 있도록 한다. Differential logging은 다음과 같은 장점을 가진다.

- 로그를 여러 디스크에 병렬적으로 flush할 수 있다.
- 재시작 시에 백업 데이터베이스를 로딩하는 것과 로그를 반영하는 것을 트랜잭션 순서에 관계없이 수행할 수 있다.
- 물리적 로깅에 비해 로그의 양을 반으로 줄일 수 있다.

즉, *differential logging*을 채용함으로써, P*TIME은 갱신 위주의 분야에도 고성능을 낼 수 있다.

그리고 P*TIME은 fuzzy checkpointing을 사용하므로 checkpointing 중에도 트랜잭션을 수행할 수 있다.

3.4. 인덱스 관리 및 인덱스 동시성 제어

P*TIME은 exact 검색을 위하여 hash 인덱스를 제공하고 1차원 영역 질의를 위해 cache 효과를 고려한 B+-tree계열 인덱스를, 다차원 영역 질의를 위해 본 연구실에서 자체 개발하여 국내외 특허를 출원한 CR-tree 인덱스[1]를 제공한다. CR-tree는 키 압축 기법을 이용하여 검색 시에 cache miss를 줄여 성능을 향상시키게 된다.

또한 매우 저비용의 인덱스 동시성 제어 기법을 사용하여 multiprocessor 환경에서의 scalability가 뛰어나다.

3.5. 데이터 타입 및 질의 처리

P*TIME에서 제공하는 데이터의 타입과 데이터베이스의 메타데이터를 관리하기 위하여 data dictionary가 존재한다.

P*TIME의 질의는 파싱된 후 data dictionary의 인덱스 정보 등에 따라 질의 실행 계획을 수립하여 바이트 코드 형태로 변환된다. 이것은 query interpreter에 의해 실행된다.

또한 질의 수행의 성능을 높이기 위해 사용자가 질의를 data dictionary에 등록시킬 수 있다. 등록된 질의는 바이트 코드 형태로 저장되어 추후에 사용자가 질의 ID와 파라메터를 넘겨주면 query interpreter에 의해 수행된다.

3.6. 제공하는 API

유연한 응용프로그램 서버 구조를 위해 P*TIME은 다양한 인터페이스를 제공한다.

서버 CLI(Call Level Interface)는 내장 응용프로그램 로직을 구현할 수 있는 인터페이스이다. 고성능이 요구되는 응용프로그램이나 DB의 데이터에 잦은 접근이 필요한 응용프로그램의 경우 응용프로그램 로직이 DBMS에 내장되는 것이 성능을 높이는데 도움이 된다.

P*TIME의 클라이언트를 위해 자바의 경우 JDBC를 제공하며 C++의 경우 JDBC와 유사한 객체지향형 인터페이스 라이브러리를 제공한다. 이런 라이브러리들은 session manager를 통하여 네트웍을 통하거나 공유 메모리를 통해 서비스를 제공받는다. 특히 클라이언트와 서버가 같은 호스트에 존재하며 데이터베이스의 접근 많은 경우 공유 메모리를 사용하여 데이터를 주고받으면 네트웍 오버헤드를 줄이는 장점이 있다.

4. 성능 평가

4.1. 실험 환경

P*TIME의 성능을 평가하기 위해 SK Telecom에서 사용하는

```
Customer ( ServiceMgmtNum uint64, ServiceNum uint32,
        RegisterationNum uint64, ServiceCode char,
        ServiceType char, IsTtlSubscriber char,
        SubscriberName char[40], Counter uint64 );

Q1) SELECT * FROM Customer WHERE ServiceNum=?;
Q2) SELECT * FROM Customer WHERE RegistrationNum=?;
Q3) SELECT * FROM Customer WHERE ServiceMgmtNum=?;
Q4) UPDATE Customer SET Counter=1
        WHERE ServiceMgmtNum=?;
```

그림 2. 사용자 데이터베이스 스키마와 질의

1200만 사용자 DB를 이용하여 디렉토리 서버를 구축하였다. 디렉토리 서버가 DBMS에 내장된 경우와 클라이언트/서버 환경에서의 throughput과 response time을 측정하도록 한다. DB는 하나의 테이블과 3개의 hash 인덱스로 이루어지며, 그 크기는 데이터가 850MB, 인덱스가 1GB이다. 스키마와 질의는 그림 2와 같다. 여기서 Q1~Q3은 검색 질의이며, Q4는 갱신 질의이다.

실험은 6개의 UltraSPARC II (400MHz, 8MB L2 cache) CPU와 [illegible] 서버를 이용하여 수행하였다. 어진 A5200 RAID를 3개의 독립된 mirrored disk와 하나의 hot standby disk로 설정하여 동시에 세 디스크에 병렬적으로 logging이 가능하도록 하였다. P*TIME은 로그를 빠르게 쓰기 위해 OS에서 제공하는 UFS(Unix File System)뿐만 아니라 raw 디스크를 사용할 수 있다. 이 논문에서는 UFS를 사용한 보수적 실험 결과를 보인다. 클라이언트는 Compaq Proliant ML570 서버(Xeon 700MHz*4, 2MB cache)를 사용하였다.

4.2. 실험 결과

그림 2의 질의를 P*TIME 내장 디렉토리 서버에서 수행한 결과는 표 1과 같다. 검색 성능은 60~70만 TPS의 throughput을 나타낸다. Q2가 다른 질의에 비해 성능이 떨어진 이유는 RegistratioinNum 값이 동일한 레코드가 많이 있기 때문이다. 갱신의 경우 10만 TPS 정도의 성능을 보인다. 그러나 독립된 디스크를 추가로 장착하여 동시에 더 많은 디스크에서 병렬적으로 logging을 수행하면 더 높은 성능을 보일 수 있으며 로그를 쓸 때 UFS를 사용하지 않고 raw 디스크를 사용하면 역시 성능이 높아진다[2].

한편 클라이언트/서버 환경에서 성능을 측정한 결과는 표 2와 같다. 클라이언트는 질의는 Q1~Q3을 99.6%, Q4를 0.4% 사용하였다. P*TIME에서는 네트웍 비용을 줄이기 위하여 클라이언트가 한번 메시지를 보낼 때에 여러 질의를 포함할 수 있으며 그 결과를 한꺼번에 받을 수 있다. 표 2의 결과와 같이 이 방식을 사용하여 네트웍의 병목 현상을 어느 정도 해소할 수 있다.

클라이언트/서버 환경에서는 세션을 처리하고, 질의와 질의 결과를 통신을 통하여 주고 받는 오버헤드가 포함되어 내장 응용프로그램 서버 환경보다 성능이 떨어지게 된다.

Query	TPS	Response Time
Q1	710,000	6 us
Q2	620,000	6 us
Q3	710,000	6 us
Q4	113,000	130 ms

표 1. 내장형 응용프로그램 서버 성능

Transactions/ Message	Messages/sec	TPS	Response Time
1	7,000	7,000	0.1 ms
10	3,600	36,000	24 ms
50	2,100	100,500	51 ms

표 2. 클라이언트/서버 성능

4.3. 벌크 로딩 및 회복 성능

P*TIME에서 4.1의 DB를 텍스트 파일에서 벌크 로딩하여 인덱스를 구축한 후 전체 데이터베이스를 checkpointing할 때의 성능은 테이블 로딩 시간이 249초 (47221 레코드/초)이며 인덱스 3개를 sequential하게 구축하는데 소요된 시간이 40초(503,920 삽입/초), checkpointing 시간이 58초(14.7 MB/초)이다.

또, 시스템이 crash된 후 전체 데이터베이스를 회복하는데 소요되는 시간은 백업 데이터베이스 로딩 시간이 13초(65 MB/초), 로그 처리 시간이 30~40 MB/초, 인덱스 구축 시간이 15초가 소요되었다. 재시작 시에는 인덱스를 병렬적으로 구축하므로 벌크 로딩할 때보다 높은 성능을 나타낸다.

5. 결론

논문에서는 제2세대 고성능 메인 메모리 DBMS인 P*TIME을 소개하였다. P*TIME은 기존의 1세대 메인 메모리 DBMS와 달리 cache, 멀티 프로세서 등의 현재 하드웨어 아키텍처를 고려한 인덱스, 동시성 제어 기법 등을 사용하였고, differential logging 기법을 이용하여 logging과 회복을 완전히 병렬적으로 수행할 수 있다. 즉 검색 성능과 갱신 성능이 모두 뛰어나다. P*TIME을 디렉토리 서버로 사용할 경우 검색은 60~70만 TPS의 성능을 내며 갱신의 경우에도 10만 TPS 이상을 낼 수 있다.

6. 참고 문헌

[1] Kihong Kim, Sang K. Cha, Keunjoo Kwon, "Optimizing Multidimensional Index Trees for Main Memory Access," In *Proceedings of ACM SIGMOD Conference*, 2001.

[2] Juchang Lee, Kihong Kim, Sang K. Cha, "Differential Logging: A Commutative and Associative Logging Scheme for Highly Parallel Main Memory Databases," In *Proceedings of IEEE ICDE Conference*, 2001.

[3] Peter Boncz, Stefan Manegold, Martin Kersten, "Data Architecture Optimized for the new Bottleneck: Memory Access," In *Proceedings of VLDB Conference*, 1999.

[4] H. V. Jagadish, D. Lieuwen, R. Rastogi, and A. Silberschatz, "Dali: A High Performance Main Memory Storage Manager," In *Proceedings of VLDB Conference*, 1994.

[5] Peter A. Boncz, Wilko Quak, Martin L. Kersten, "Monet And Its Geographic Extensions: A Novel Approach to High Performance GIS Processing," In *Proceedings of EDBT Conference*, 1996.

[6] TimesTen Performance Software, http://www.timesten.com

[7] J. H. Park, Y. Sik Kwon, K. Kim, S. Lee, B. D. Park, and S. K. Cha, "Xmas: An Extensible Main-Memory Storage System for High-Performance Applications," In *Proceedings of ACM SIGMOD Conference*, 1998.

[8] J. H. Park, K. Kim, S. K. Cha, M. S. Song, S. Lee, and J. Lee, "A High-performance Spatial Storage System Based on Main-Memory Database Architecture," In *Proceedings of DEXA Conference*, 1999.

[9] Altibase Co., http://www.rtbase.com

[10] Real Time System Technology Inc., http://www.realsystech.com

[11] Jun Rao and Kenneth Ross, "Making B+-trees Cache Conscious in Main Memory," In *Proceedings of ACM SIGMOD Conference*, 2000.