제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

주기억장치 상주형 DBMS를 이용한 실시간 캐쉬서버 설계 및 구현

김종춘*, 두용재**, 진성일*
* 충남대학교 컴퓨터과학과, ** (주) 리얼타임테크,
e-mail : jchkim@cs.cnu.ac.kr

Design and Implementation of Real-Time Cache Server Using Memory Resident DBMS

Jong Chun Kim*, Yong Jae Doo **, Seong Il Jin*
Dept. of Computer Science, Chungnam National Univ.

요 약

이질적인 데이터베이스 간의 중복 데이터 일관성을 유지문제를 해결 함으로써 기존의 디스크 기반 범용 데이터베이스를 사용하는 시스템의 방대한 데이타들을 주기억 장치 상주형 데이터베이스에 중복 저장함으로써 디스크 I/O 와 관련된 지연 시간 없이 빠른 성능 향상을 기대할 수 있으므로 네트워크상의 불특정 다수의 서비스 형태에 빠른 응답 시간과 처리 시간을 제공 할 수 있다. 본 본문에서 제시하는 중복 데이터 일관성 유지 기법은 디스크 기반 범용 데이터베이스의 REDO LOG 를 참조하여 갱신트랜잭션에 관한 로그를 주기억 상주형 데이터베이스에 적용하도록 설계하고 두 데이타베이스 간의 일관성 유지를 위해 수행되는 기능들이 성능 향상을 위한 목적을 위배하지 않으면서 이질적인 데이터베이스 간의 일관성을 보장하도록 한다. 또한 두 데이터베이스의 성능 차이로 인해 발생할 수 있는 문제점들을 해결하고, 시스템 붕괴 시를 고려하여 유지되지 못한 데이터 일관성에 대해서도 복구 후에 일관성 유지를 가능케 하는 기법을 제공한다.

1. 서론

최근 초고속 정보 통신망을 기반으로 한 인터넷 서비스의 활성화로 인해, 대부분의 서비스들은 네트워크 환경을 중심으로 이동하였고 자료의 양이 방대해지고 복잡해짐에 따라 이러한 데이터들을 효율적으로 활용하기 위해 응답 시간과 처리 속도가 빠른 데이터베이스 시스템의 필요성이 제기되었다. 하지만 성능적인 면에서 현재 널리 사용되고 있는 디스크 기반 데이터베이스 시스템은 문제성을 가지고 있다. 디스크 기반 데이터베이스 시스템은 기능은 방대하나 규모가 크고, 데이터 처리 성능이 떨어져 인터넷 상에서 폭주하는 데이터를 처리하기에는 데이터를 다루기 위한 디스크 액세스의 오버헤드가 지나치게 크므로 속도 및 성능 면에서 부족한 점들이 많다. 그러므로 네트워크 기반의 응용 시스템의 불특정 다수의 서비스 형태에 디스크 기반 범용 데이터베이스를 활용하기에는 그 한계가 있다. 이러한 한계를 극복하고 서비스 향상을 위해서 최근 도입되고 있는 새로운 솔루션으로 데이터베이스나 웹 콘텐츠에 대해 캐쉬 기법을 활용하는 미들웨어를 사용하거나, 주기억 장치 상주형 데이터베이스(Main-Memory Database Management System, MM-DBMS)를 이용하여 성능을 향상시키는 방법들이 제시되고 있다.[10] 이러한 기법들은 기존의 디스크 기반 범용 데이터베이스에 존재하는 데이터들을 그대로 유지하면서, 빠른 성능을 보장하는 미들웨어(주기억 장치 상주형 데이터베이스)에 데이터를 중복 저장하여 데이터를 제공함으로써 성능 향상을 유도하는 시스템을 설계 및 구현하였다

본 논문에서는 2 장에서 중복 데이터를 위한 일관성 유지 기법에 대하여 알아보고, 3 장에서는 주기억 장치 상주형 데이터베이스 이용한 캐쉬서버 시스템을

이 연구는 충남대학교 소프트웨어연구센터 및 BK21 정보통신인력양성사업단의 지원을 받았음

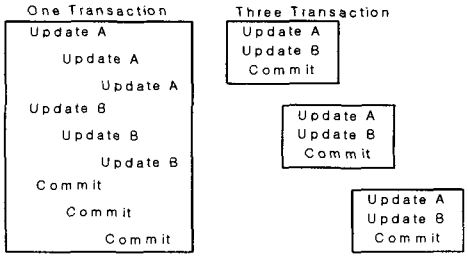
설계 및 구현 하고, 4 장에서는 성능평가을 하고, 5 장에서는 결론 및 향후 연구 방향에 대하여 기술한다.

2. 중복 데이터를 위한 일관성 유지 기법

2.1 데이터베이스에서의 중복 기법

데이터 베이스에서의 중복된 데이터들간의 일관성 유지 기법은 Eager Replication 과 Lazy Replication 기법으로 구분할 수 있다[1][2].

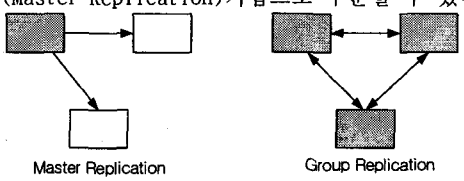
[그림 2-1]에서와 같이 Eager Replication 은 하나의 트랜잭션에 의해 서로 다른 데이터베이스에 존재하는 중복된 데이터들을 모두 갱신하므로 트랜잭션 실행 순서를 보장하고 중복 데이터들간의 동시성 제어를 고려하지 않아도 항상 중복 데이터들간의 일관



[그림 2-1] Eager Replication & Lazy Replication at Three Node

성이 유지될 수 있지만, 모든 데이터 베이스들의 갱신이 실행될 때까지 트랜잭션이 완료되지 않으므로 성능 저하와 함께, 응답시간이 길어질 수 있다. 그러나 Lazy Replication 은 하나의 데이터베이스에 트랜잭션이 실행 완료되었을 때 클라이언트에 응답을 전송하고, 다른 데이터베이스들의 중복 데이터들을 갱신하기 때문에 응답시간을 단축되어 성능을 향상시킬 수 있다.

중복 데이터의 갱신 트랜잭션 실행 주체 관점에서 보면 그룹 중복(Group Replication)기법과 마스터 중복(Master Replication)기법으로 구분할 수 있다[8].



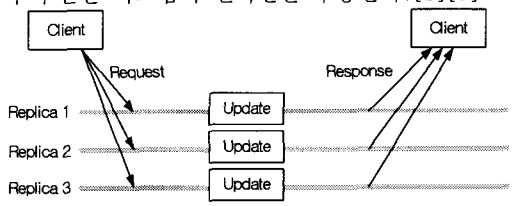
[그림 2-2] Master & Group Replication

마스터 중복 기법은 하나의 주 데이터베이스가 모든 트랜잭션을 요청 받아 다른 데이터베이스들에게 갱신하도록 트랜잭션을 전송하여, 모든 갱신 트랜잭션은 마스터 데이터베이스에 의해 관리되므로 중복 데이터 일관성 유지를 위한 단순한 방법을 제공한다. 하지만 모든 갱신 트랜잭션들은 하나의 데이터베이스에서만 가능하므로 마스터 데이터베이스의 부하량이 높아 질 수 있다. 그룹 중복 기법은 모든 데이터베이스에서 갱신 트랜잭션이 실행 가능하고 중복 데이터 갱신 관리 역시 모든 데이터베이스에서 관리 가능한 형태이다. 그룹 중복 기법은 모든 데이터베이스이 갱신 트랜잭션을 실행할 수 있기 때문에 시스템의 효율성이 좋지만, 중복 데이터 갱신 관리가 복잡하고 데이터베이스들 간의 잠금 기법을 활용한 동시성 제어가 필요하다.

2.2 분산 데이터베이스 시스템의 중복 데이터 일관성 유지 기법

분산 데이터베이스 시스템에서의 중복 데이터 유지는 주로 무고장성(Fault-Tolerance)을 위한 목적으로 사용된다. 분산 데이터 베이스 시스템에서의 중복 데이터 일관성 유지 기법은 분산되어 있는 시스템들이 유지하고 있는 데이터들을 갱신하는 시점에 따라 Active Replication 기법과 Passive Replication 기법으로 구분할 수 있다.

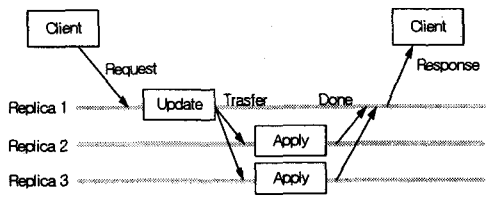
Active Replication 기법은 클라이언트가 모든 분산된 데이터베이스에 질의를 보내면 각 데이터 베이스들은 질의를 실행하면서 자신이 유지하고 있는 데이터들을 모두 변경시킨다. 실행이 완료된 후에 클라이언트에게 실행 응답을 전송하게 되는데 클라이언트는 각 시스템의 모든 응답을 수신하지 않고 가장 먼저 수신한 시스템의 결과만을 수용한다.[2][9]



[그림 2-4] Active Replication

이러한 Active Replication 의 기법은 클라이언트에게 특정 서버의 붕괴 여부와 관계없이 실행을 가능하게 한다. 하지만 Active Replication 기법은 모든 분산된 데이터 베이스가 하나의 클라이언트 요청에 대해서 모두 실행 되어야 하기 때문에 불필요한 실행을 함으로써 리소스의 낭비가 초래된다는 단점이 있다.

Passive Replication 기법은 클라이언트가 하나의 시스템에 요청을 하면 요청을 받은 시스템이 실행을 하고 변경된 데이터를 다른 분산 시스템에 전송하여 변경된 데이터의 적용여부를 수신한 후에 클라이언트에게 실행 응답을 전송한다. Active Replication 기법에 비해 리소스를 효율적으로 활용할 수 있지만 클라이언트에게 요청을 받은 시스템이 붕괴되었을 때 동기화를 위한 복잡한 설정 과정이 필요하다.

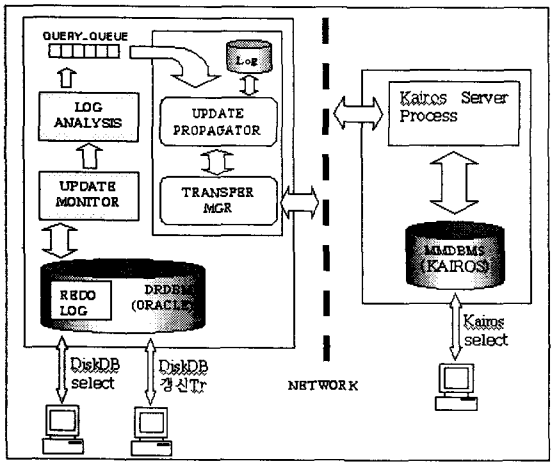


[그림 2-5] Passive Replication

3. MMDBMS를 이용한 캐쉬서버 설계 및 구현

3.1 전체 시스템 구성도

본 논문에서 제안하는 디스크 기반 범용 데이터베이스와 주기억 장치 상주형 데이터베이스의 중복 데이터 일관성 유지 과정은 [그림 3-1]과 같다.



[그림 3-1] 이기종 DBMS 간의 중복 데이터 일관성 유지를 위한 시스템 구조

갱신관련 트랜잭션을 Wi(x)라 하면 Update Monitor는 DRDBMS의 REDO LOG에서 변경 질의(Wi' (x))를 획득하여 마지막으로 QUERY_QUEUE에 저장된 질의보다 최근의 것이면 LOG ANALYSIS에 질의(Wi' (x)) 전송하고 LOG ANALYSIS는 획득한 변경 질의(Wi' (x))를 질의 별(INSERT, DELETE, UPDATE)로 구분하여 MMDBMS에 적용 가능한 질의로 변경(Wi(x))하여 QUERY_QUEUE에 저장한다. UPDATE PROPAGATOR는 QUERY_QUEUE에서 Wi(x) 획득하여 TRANSFER MGR에 전송하여 MMDBMS에 적용한다. 단 MMDBMS에 전송할 수 없는 상황이면 시스템 붕괴나, 네트워크 단절과 같은 문제이므로 LOCAL에 위치한 실패로그 파일에 로그(Wi(x))를 기록하고 주기적으로 재연결을 시도한다.

3.2 Update Monitor Module

Update_Monitor 모듈은 DRDBMS와의 연결을 시도한 후 연결이 정상적으로 수행되면 DRDBMS의 REDO LOG FILE을 분석하여 갱신 트랜잭션에 해당하는 로그를 읽어온 로그(Wi' (x))가 최근에 QUERY_QUEUE에 저장한 로그보다 최근의 것이면 LOG_ANALYSIS 모듈에 전송하고 이전의 것이면 로그(Wi' (x))는 무시된다. 어떠한 로그가 최근의 것인지는 로그(Wi' (x))의 정보 중에서 SCN, RBASQN, RBABLK, RBABYTE(LOG_REC 구조체) 항목을 이용하여 비교한다. DRDBMS의 REDO LOG에서 변경 질의(Wi' (x))를 획득하기 위한 구조체는 다음과 같다.

```
struct LOG_REC
{
  int scn;     //System change number (SCN) when the
               redo record was generated
  int rbasqn; //RBA sequence number of the log that
               contained this redo record
  int rbablk; //RBA block number within the log file
  int rbabyte;//RBA byte offset within the block
  char seg_owner[32]; // Owner of the segment
  char seg_name[256]; // Name of the segment
  char row_id[19];
  int rollback;
  char operation[32];
  char sql_redo[4000];
  char sql_undo[4000];
};
```

3.3 LOG ANALYSIS Module

LOG_ANALYSIS 모듈은 UPDATE_MONITOR에서 전송 받은 갱신 트랜잭션에 관한 로그(Wi' (x))를 MMDBMS에 적용 가능한 질의로 변형한 후 변형된 로그(Wi(x))를 QUERY_QUEUE에 저장한다. 또한 마지막으로 QUERY_QUEUE에 저장한 로그를 LOCAL에 위치한 파일에 기록한다.

```
class Log_Analysis
{
 private:
      void Log_segOwner(tlog_dat* la);
      char* Log_null(char* str);
      char* Log_insert(char* redo, char* rowid);
      char* Log_delete(char* redo, char* rowid);
      char* Log_update(char* redo, char* rowid);
      char* Log_del_colon(char* str);
      char* Log_correct_type(int oper, char* str);
 public:
      Log_Analysis( );
      ~Log_Analysis ( );

      int Log_AnalysisMgr(struct LOG_REC la);
};
```

3.4 UPDATE PROPAGATOR Module

UPDATE_PROPAGATOR 모듈은 QUERY_QUEUE에서 저장된 로그(Wi(x))를 가져온 후 TRANSFER_MGR 모듈에 로그를 전송한다. 전송할 수 없는 상황이면 시스템 붕괴나, 네트워크 단절과 같은 문제이므로 LOCAL에 위치한 실패로그 파일에 로그(Wi(x))를 기록한다. 또한 주기적으로 재연결을 시도하여 연결이 정상적으로 수행이 되면 아직 전송이 되지 못한 로그를 가지고 있는 실패로그 파일에 로그를 읽어와 TRANSFER_MGR에 전송하고 실패로그 파일에서 해당 로그를 삭제한다.

```
class TRANS_MGR {
private:
      SQLHENV henv;
      SQLHDBC hdbc;
      SQLHSTMT hstmt;
      int append_Log(tlog_dat alog);
      int trans_leavedLog(void);
      int flush_Log(void);
public:
```

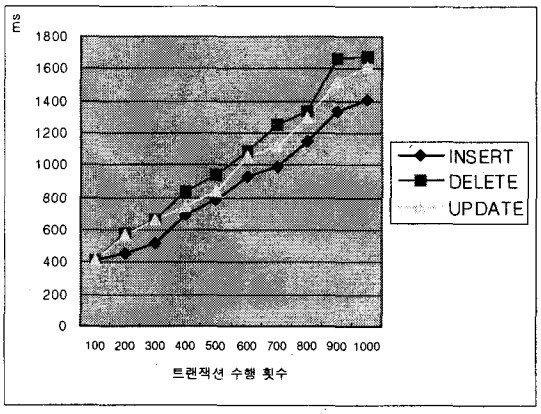
```
        TRANS_MGR();
        ~TRANS_MGR();

        int kairos_Connect(void);
        int kairos_SQLExecute(char* alog);
        int kairos_DisConnect(void);

        tconfig config_read (void);
        int config_write (tconfig cfg);
};
```

4. 성능평가

본 논문에서 제안한 중복 데이터 일관성 유지 기법은 디스크 기반 데이터베이스의 REDO LOG 파일에서 실행 완료된 갱신 트랜잭션을 메시지 큐를 통해 Update Propagator 에 전송한 후 주기억 상주형 데이터베이스에 갱신 트랜잭션을 전송하게 된다. 그러므로 Update Monitor/Update Propagator 에 의해 실행되는 중복 데이터 일관성 유지를 위한 처리 과정은 디스크 기반 데이터베이스의 실행에 어떠한 영향을 주지 않으면서 처리하게 된다. 성능평가 환경은 솔라리스 5.8(SUN V880, 메모리 4Gbyte) 시스템을 사용하였고, 디스크 기반 데이터베이스로 ORACLE 9i, 주기억 상주형 데이터베이스로 KAIROS 2.0 을 사용하였다.



[그림 4-1] 캐쉬서버시스템의 수행시간

[그림 4-1]은 디스크 기반 데이터베이스에서 갱신 트랜잭션이 발생한 경우 Commit 수행시 데이터베이스에 반영되는 시점과 REDO LOG 파일에 기록되는 시점은 동일하다고 가정할 때, 디스크 기반 데이터베이스(ORACLE 9i)의 REDO LOG 파일에서 갱신에 관련된 로그를 주기억 상주형 데이터베이스(KAIROS)에 적용하는데 있어 로그의 수와 각 Operation 별 중복 데이타 일치에 걸리는 시간을 측정한 것이다. 측정에서 보이는 바와 같이 INSERT, UPDATE, DELETE 순으로 수행시간이 길어지는 것으로 나타났다.

5. 결론 및 향후 개발 방향

디스크 기반 범용 데이터베이스를 활용한 시스템의 성능 향상을 위한 기법은 웹 캐쉬나 데이터베이스 캐쉬, 주기억 장치 상주형 데이터베이스 활용 등의 많은 방법이 연구되었다. 이 중 주기억 장치 상주형 데이터베이스를 활용할 때 데이터 검색 관련 트랜잭션의 수행시간을 향상 시킬 수 있고, 특정 데이터베이스에 종속되지 않기 때문에 다른 디스크 기반 범용 데이터베이스와의 연계성을 가질 수 있다. 그러나 주기억 장치 상주형 데이터베이스를 활용하였을 때, 발생하는 기존의 디스크 기반 범용 데이터베이스와의 중복 데이터 일관성 유지 문제가 발생하기 때문에 본 논문에서는 이를 해결하기 위한 기법에 대해 연구하였다.

앞으로 클라이언트에게 더 높은 투명성을 제공하기 위해 호환되지 않는 질의 실행에 대한 방법이 제시되어야 할 것이다. 또한 주기억 장치 상주형 데이터베이스의 분산화, 이중화를 고려하여 중복 데이터 일관성 유지 기법이 연구되어야 할 것이다.

참고문헌

[1] Jim Gray, Pat Helland, Patrick E. O'Neil, Dennis Shasha: The Dangers of Replication and a Solution. SIGMOD Conference 1996: 173-182

[2] Fernando Pedone, Matthias Wiesmann, André Schiper, Bettina Kemme, Gustavo Alonso: Understanding Replication in Databases and Distributed Systems. ICDCS 2000: 464-474

[3] Oracle 9i Application Server : Database Cache, Oracle White Paper

[4] X.Defago, A.Schiper, and N.Sergent. Semi-passive replication. In Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems(SRDS), pages 43-50, West Lafayette, IN, USA, Oct. 1998

[5] D.powell, M. Chereque, and D.Drackley. Fault-tolerance in Delta-4*. ACM Operating Systems Review, SIGOPS, 25(2):122-125, Apr. 1991.

[6] Esther Pacitti, Pascale Minet, and Eric Simon: Replica Consistency in Lazy Master Replicated Databases. Distributed and Parallel Databases 9(3): 237-267 (2001)

[7] JoAnne Holliday, Divyakant Agrawal and Amr El Abbadi: Database Replication: If You Must be Lazy, be Consistent. Symposium on Reliable Distributed Systems 1999: 304-305

[8] Abraham Waksman, "A Model of Replication" JACM 16(1): 178-188 (1969)

[9] Naokazu Nemoto, Hiroaki Higaki, Katsuya Tanaka, Makoto Takizawa: Pseudo-Active Replication Protocol for Reliable Replica Group. ICDCS Workshop on Group Communications and Computations 2000: C48-C55

[10] http://www.realtimetech.co.kr/kairos/main.html "㈜리얼타임 KAIROS 의 활용모델"