대용량 데이터스트림 실시간 압축 저장을 위한 저장관리자 설계 및 구현

(Design and Implementation of Storage Manager for Real-Time Compressed Storing of Large Volume Datastream)

이 동 욱* 백 성 하* 김 경 배** 배 해 영***

(Dong Wook Lee) (Sung Ha Baek) (Gyoung Bae Kim) (Hae Young Bae)

요 약 유비쿼터스 환경에서 발생하는 실시간 데이터스트림의 처리 및 관리에 대한 요구사항이 증가되고 있다. 특히 인터넷 보안, 공장 자동화 기기설비 관리 등에서 발생하는 데이터스트림은 대용량 데이터가 실시간으로 발생하는 특징을 가지고 있어 이를 처리하기 위한 특화된 DSMS의 저장관리자의 개발이 요구된다. Coral8과 같은 기존의 DSMS의 경우 일반적인 데이터스트림의 처리가 가능 하지만 초당 10만 건 이상의 대용량의 실시간 데이터스트림이 발생하는 특수한 상황의 경우 이에 대한 처리 성능이 부족하며, 이에 대한 저장이 불가능하다. 관련 분야에서 일반적으로 사용되는 오라클10g의 경우 저장, 관리 성능은 우수하지만 실시간 데이터스트림 처리에 대한 고려가 되어 있지 않다. 본 논문에서는 삼성전자, 하이닉스, HP 등에 반도체, LCD 제조공장의 생산 기계에서 발생하는 대용량 데이터스트림을 실시간 압축 저장이 가능한 DSMS 저장관리자를 설계 및 구현하였다. 본 문에서는 시스템의 구조 및 주요 컴포넌트에 대해 설명하며, 관련시스템과 비교 평가를 통해 제안 시스템의 우수성을 보인다.

키워드 : 대용량 데이터스트림, 실시간 압축저장, DSMS

Abstract Requirement level regarding processing and managing real-time datastream in an ubiquitous environment is increased. Especially, due to the unbounded, high frequency and real-time characteristics of datastream, development of specialized stroge manager for DSMS is necessary to process such datastream. Existing DSMS, e.g. Coral8, can support datastream processing but it is not scalable and cannot perform well when handling large-volume real-time datastream, e.g. 100 thousand over per second. In the case of Oracle10g, which is generally used in related field, it supports storing and management processing. However, it does not support real-time datastream processing. In this paper, we propose specialized storage manager of DSMS for real-time compressed storing on semiconductor or LCD production facility of Samsung electronics, Hynix and HP. Hynix and HP. This paper describes the proposed system architecture and major components and show better performance of the proposed system compared with similar systems in the experiment section.

Keywords : Large Volume Data Stream, Real-time Compressed Storing, DSMS

1. 서 론

최근 유비쿼터스 컴퓨팅 환경에서의 RFID 및 센서 뿐만 아니라 네트워크 모니터링, DTV 방송 스트림, 공장 자동에 따른 생산과정 모니터링 등에서 실시간으로 발생하는 데이터스트림 처리에 대한 요구사항과 그 중요성이 높아지고 있다. 이는 실시간으로 발생하는 데이터스트림들에 대한 이상치 발견, 긴급 데이터 분류, 사용자 데이터 필터링과 같은 질의 처리가 가능한 DSMS(Data Stream Management System)가 요구된다[2, 6, 9].

특히, Equipment Engineering 분야는 자동화 공장에서 생산과정 관리나 기계고장 예방, 이상원인 분석을 위한 실시간으로 발생하는 대용량 데이터스트림 처리 및 관리를 위한 연구가 진행 중이다[1, 8, 10, 15]. 이러한 환경에서 DSMS(DataStream Management System)는 초당 10만 이상의 고성능 실시간 질의처리 성능 및 저장 기술이 요

† 본 연구는 건설교통부 첨단도시기술개발사업- 지능형국토정보기술혁신 사업과제의 연구비지원(07국토정보C05)에 의해 수행되었음.

* 인하대학교 정보공학과 박사과정,dwlee@dblab.inha.ac.kr, shbaek@dblab.inha.ac.kr

** 서원대학교 컴퓨터교육학과 조교수, gbkim@seowon.ac.kr(교신저자)

*** 인하대학교 정보공학부 교수, hybae@inha.ac.kr

논문접수 : 2009.07.14 수정일 : 1차 2009.08.31 / 2차 2009.09.11 심사완료 : 2009.09.18

구된다[5, 11].

본 논문에서는 삼성전자의 반도체 및 LCD 제조공장의 생산 기계설비에서 데이터스트림으로 발생하는 로그 데이터를 실시간으로 압축 저장이 가능한 DSMS의 저장관리자를 설계 및 구현하였다. 그림 1은 반도체 및 LCD 생산장비의 관리를 위한 기반 환경 구조도이다.

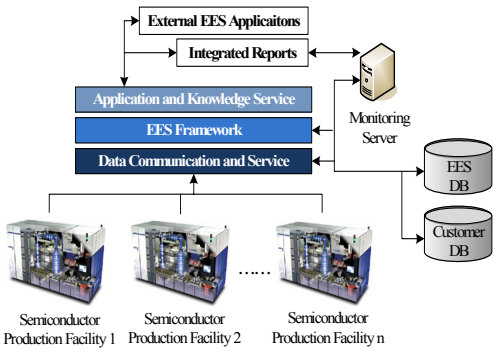

그림 1. 반도체 및 LCD 생산설비 관리를 위한 기반 구조

그림 1에서, 각각의 생산 장비는 작업 시 장비의 온도, 습도, 생산 개수, 작업시간 등의 정보를 포함하는 로그 데이터를 생성하여 제안하는 DSMS에 전송한다. 이러한 로그 데이터는 데이터스트림 형태로 초당 10만 ~ 50 만개가 발생하며, 온도, 습도 등의 이상치 검출, 장비 오류 메시지 검출, 일정한 주기에 따른 정보 모니터링 등의 응용서비스를 위해 이러한 대용량 데이터스트림은 실시간 연속질의 처리와 데이터 이력관리 또는 패턴 분석 등을 위해 저장되어져야 한다. 그러나 초당 10만개 이상 발생하는 데이터 스트림 저장을 위해서는 많은 저장 비용이 요구되는데, 예를 들어 제안 시스템의 기반환경이 되는 (주)비스텔의 EES Framework에서 초당 10만개의 임의의 데이터를 생성하여 실험을 한 경우 하루에 약 1TB의 데이터가 생성되었다.[7, 12] 따라서 이러한 대용량 데이터 저장을 위한 70% 이상의 고효율 압축 저장 기술이 필요하다.

본 논문의 구성은 다음과 같다. 2장에서는 대용량 데이터스트림 관리를 위한 관련연구들에 대해 설명을 하고, 3장에서는 제안 저장 관리자 구조를 설명한다. 4장에서는 기존의 관련 시스템과 성능평가를 보이고, 5장에서는 결론 및 향후 연구에 대해 기술한다.

2. 관련 연구

본 장에서는 기존의 데이터스트림 처리를 위한 관련 연구와 제안시스템의 기반 환경인 EES Framework에 대해 설명한다.

2.1 기존의 데이터스트림 처리를 위한 DSMS

데이터스트림 처리를 위한 대표적인 관련연구는 Coral8, StreamBase, Stream이 있다. Coral8 엔진은 동적으로 변화하는 환경에서 데이터에 대한 빠른 처리, 분석 기능을 제공하고 이벤트들을 관리하는 선도적인 툴로써, CEP (Complex Event Processing), ESP(Event Stream Processing)를 위한 기능을 제공한다[3, 4]. 또한 대용량의 fast-moving 데이터를 동적으로 처리할 수 있는 응용 프로그램 개발을 위해 관계형 데이터베이스와의 메시지 기반 인터페이스 기술을 제공하며, 스트림 데이터의 질의를 위한 SQL과 유사한 형태의 질의 처리 언어를 지원한다. Coral8의 컴포넌트는 크게 두 가지로 분리되며, 실제적인 데이터들을 처리하는 Coral8 서버와, Coral8 Studio로 나뉜다. Coral8 서버는 입력된 데이터 스트림에 대하여 등록된 연속질의를 수행한다. Coral8 Studio은 사용자가 CCL질의, view streams, windows를 용이하게 사용할 수 있게 하는 GUI로써, Coral8 서버를 제어하고 모니터링을 한다. 하지만 Coral8은 처리된 데이터스트림에 대한 저장을 지원하지 않아 저장이 필요할 경우 이를 위한 별도의 DBMS을 설치해야 한다. 이는 처리된 데이터스트림의 저장이 필요한 경우 기존의 DBMS에 의존하게 되며, DBMS의 벌크 삽입 성능 이상의 데이터가 실시간으로 발생할 경우 이에 대한 저장이 어려울 뿐만 아니라 대용량 데이터 저장을 고려한 실시간 압축 저장을 지원하지 않는다.

StreamBase는 실시간 비즈니스 이벤트 분석을 위한 데이터 관리의 새로운 플랫폼으로써, 스트림 처리를 지원하며, 대용량 데이터 처리를 지원하기 위하여 설계된 실시간 데이터 처리 시스템이다[13, 14]. StreamBase는 효율적인 메모리 기반 데이터 구조를 사용하고 동시에 다른 어플리케이션과의 의존성을 최소화 하거나 제거하여 시스템의 성능을 향상시킨다. 또한 StreamBase는 고가용성의 시스템 자원 사용과 다중 어플리케이션간의 데이터 충돌 현상을 피함으로써 기존의 데이터베이스를 사용하는 시스템보다 데이터 스트림을 더욱 효과적으로 처리 할 수 있다. StreamBase는 외부 프로세스와의 상호 교환을 위한 표준 API와 일반 데이터 공급을 위한 풍부한 어댑터 셋 및 높은 수준의 프로그래밍 언어를 사용하여 표준 규격 하드웨어 상에서 구동할 수 있도록 하는 공동 사용 가능한 플랫폼이다. 하지만 Coral8과 같이 대용량 데이터의 실시간 압축저장을 지원하지 못한다.

2.2 반도체 및 LCD 생산설비 관리를 위한 EES Framework

EES Framework은 국내 비스텔사의 상용화 소프트웨어 제품으로 현재 삼성전자, 하이닉스, HP 등의 반도체와 LCD 생산 공정에 설치되어 사용 중인 시스템이다[7, 12]. EES은 반도체 및 LCD 생산 장비의 관리를 위한 Framework으로, 생산성 증가 및 효율적인 장비 관리를 목적으로 한다[7]. 이를 위해 EES Framework은 장비의 상태분석, 문제 발생의 원인 분석, 실시간 장비 모니터링 등의 기능을 제공한다. 그림 2는 EES Framework의 시스템 구조이다.

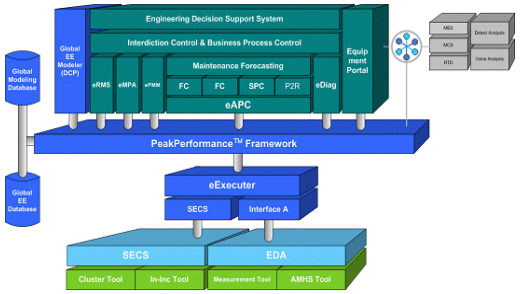

그림 2. 비스텔사의 EES Framework 시스템 구조

주요 컴포넌트로 PeakPerformanceTM Framework은 다양한 EES Applications 개발을 위한 Domain-Specific Application Framework과 Applications간의 데이터 공유를 제공한다.

PeakPerformanceTM eFDC는 생산 설비의 소스 파라미터 및 센서 데이터들을 실시간으로 모니터링하여 설비의 이상 여부를 조기에 발견, 최적의 장비상태 유지 및 여러 파라미터들 간의 상호 연관성을 분석 및 개선하여 장비가동 생산성을 높인다.

PeakPerformanceTM ePMM 은 설비를 사전에 점검하여 고장방지와 고장빈도를 감소시켜 관리 비용을 최소화하고, 정기적인 유지보수 관리 및 고장을 예방 한다. 또한, PeakPerformanceTM eMPA은. 유닛 단위까지 설비의 모든 상태를 실시간으로 모니터링하고 다양한 분석기능을 제공 한다.

위와 같이 EES Framework이 동작하기 위해서는 반도체, LCD 생산에 필요한 모든 장비로부터 로그 데이터를 전송 받아, 이에 대한 실시간 처리가 요구 된다. 이를 처리하기 위한 DSMS는 초당 10만 건 이상의 연속질의 처리 성능과 실시간 압축 저장이 가능해야 한다.

3. 대용량 데이터스트림의 실시간 압축 저장을 위한 저장 관리자

반도체 및 LCD 제조공정을 위한 EES Framework에서 각각의 생산설비에서 발생하는 로그 데이터의 모니터링에 필요한 데이터스트림 처리를 위해서는 초당 10만 건 이상의 질의 처리성능 보유, 디스크 기반의 DBMS보다 빠른 저장성능, 디스크 저장 공간 절약을 위한 데이터 압축 기능이 필요하다.

본 논문에서는 위와 같은 환경의 DSMS에서 대용량 데이터스트림의 실시간 압축 저장을 위한 저장 관리자를 설계 및 구현 하였다.

3.1 대용량 데이터스트림 실시간 압축 저장을 위한 저장 관리자

본 논문에서 제안하는 저장관리자는 처리된 대용량의 데이터스트림을 입력 받아 실시간 압축하여 파일을 관리한다. 기존의 저장 관리자와는 다르게 데이터 삽입을 중점적으로 설계가 되었으며, 이를 위해 데이터는 클라이언트에서 레코드 셋 단위로 모아서 전송을 해야 한다. 입력되는 레코드들은 반드시 생성 시간을 가지기 때문에 여러 레코드들은 거의 시간별로 정렬되며, 데이터가 파일로 쓸 때 디스크의 용량을 고려하여 데이터는 압축 후에 파일로 쓰게 된다. 저장관리자에서 데이터 저장에 쓰이는 데이터는 모두 레코드들의 집합인 레코드 셋 단위로 이루어진다. 버퍼 관리자에 여러 개의 레코드 셋이 입력되면 그 레코드 셋들을 모아서 보다 큰 크기의 레코드 셋 단위로 합친다. 이는 연산의 단위를 크게 해서 연산 속도를 높이기 위함이다. 그림 3은 저장 관리자에 속한 각 관리자와 쓰레드들 사이에 레코드 셋이 어떻게 이동하는 지 설명한다.

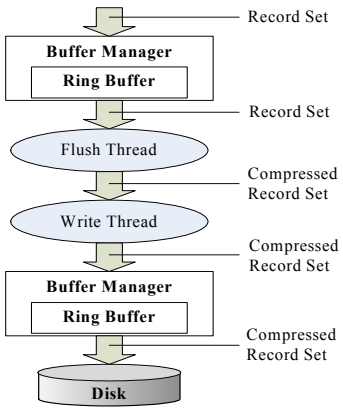

그림 3. 저장관리자와 각 매니저의 스레드 사이의 압축 저장과정

3.1.1 버퍼 관리자

버퍼 관리자는 버퍼 풀로부터 버퍼 블록을 할당 받아 레코드 셋을 입력 받고, Select 질의에 필요한 버퍼를 전달하고, 디스크에 안전하게 써진 레코드 셋을 메모리에서 삭제하는 Flush를 수행한다. 비스텔 로그 시스템은 데이터 입력의 속도를 높이는 것을 중점적으로 설계가 되었다. 따라서 레코드 셋의 입력과 Flush, 또한 레코드 셋의 입력과 Select 질의 시에 버퍼 전달은 비동기적으로 수행될 수 있다. 그러나 Select 질의 시의 버퍼전달과 Flush 연산 사이는 동시에 수행 될 수 없다. 버퍼 관리자는 이러한 연산들의 동시성 제어를 위해 버퍼 풀로부터 할당받아 사용되는 각 버퍼 블록의 Work Flag를 제어하는 역할도 한다.

버퍼 관리자는 데이터 입력에 사용할 버퍼를 버퍼 풀 구조를 사용하여 관리하며 전체 구조는 그림 4와 같다. 각각의 버퍼 블록은 고유의 아이디를 가지고 있으며 각 테이블들은 이러한 버퍼 블록의 고유 아이디로 버퍼 블록을 관리한다. 또한 각 버퍼 블록은 동시성 제어를 위한 Work Flag 정보와 시간 정보, 헤더정보 그리고 데이터 부로 구성된다.

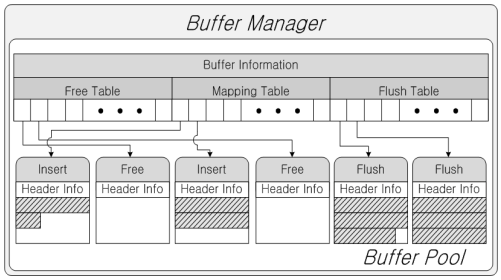

그림 4. 실시간 압축 저장을 위한 저장관리자에서 버퍼 관리자 전체 구조

버퍼 풀은 버퍼 블록을 미리 생성하여 각각의 클라이언트가 데이터 삽입을 위해 버퍼 블록을 필요로 할 때마다 할당해주는 역할을 한다. 또한 레코드 셋의 입력과 Flush, Select 등의 연산들의 동시성 제어를 위해 각 버퍼 블록은 Work Flag를 갖는다. 그림 5는 버퍼 관리자의 버퍼 풀 구조를 나타낸다.

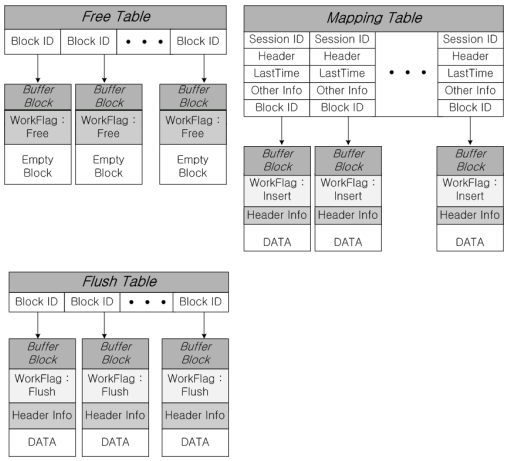

그림 5. 버퍼 관리자의 버퍼 풀 구조

각각의 버퍼 블록은 고유의 아이디를 가지고 있으며 각 테이블들은 이러한 버퍼 블록의 고유 아이디로 버퍼 블록을 관리한다. 각 테이블의 구조로 Free Table은 현재 할당 가능한 상태의 버퍼 블록의 블록 아이디로 관리하고 Flush Table 또한 블록 아이디로 Flush 대기 중인 버퍼 블록을 관리한다. Mapping Table의 세션 아이디는 버퍼 블록과 맵핑되어 있는 클라이언트의 고유 아이디를 나타내며 헤더는 현재 삽입중인 레코드 셋의 헤더 정보를 갖는다. LastTime은 최종 삽입 시간을 나타내어 최종 삽입 시간이 일정 시간 지난 대기 상태의 버퍼 블록을 Flush 대기 상태로 변경하도록 돕는다. 기타 정보(Other Info)에는 레코드 셋의 삽입 위치 등의 정보를 갖는다. 그리고 다른 테이블들과 마찬가지로 블록 아이디를 통해 버퍼 블록을 관리한다.

버퍼 블록은 크게 WorkFlag와 레코드 셋의 헤더 정보 그리고 레코드 셋으로 구성된다. WorkFlag는 Free, Insert, Flush, Wait의 작업 상태를 나타내는 Flag 값을 갖는다. 그리고 레코드 셋의 헤더 정보는 레코드 셋 내의 레코드들의 가장 오래된 생성 시간을 나타내는 minTime과 가장 최근의 생성 시간을 나타내는 maxTime을 갖는 시간 정보와 레코드 셋의 테이블 이름, 레코드 셋 내의 레코드 개수, 레코드 셋의 전체 크기로 구성된다. 그림 6은 버퍼 블록 구조이다.

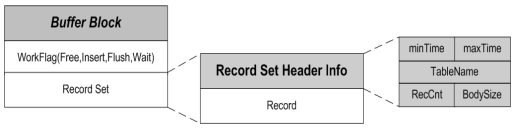

그림 6. 버퍼 관리자의 버퍼 블록 구조

본 논문의 저장 관리자에서 버퍼 데이터의 삽입은 레코드 셋 단위로 이루어지며 Session Manager로부터 분할된 레코드 셋을 입력 받아 버퍼 블록에 저장하여 결합한다. 또한 저장된 레코드 셋은 새로 입력되는 레코드 셋과 합쳐져서 더 큰 크기의 레코드 셋이 된다.
Session Manager로부터 삽입 요청을 받으면 삽입 Flag를 검사하여 분할된 레코드 셋을 순서대로 알맞은 버퍼 블록에 삽입한다. 삽입 Flag는 START, CONTINUE, FINISH, FAILURE 네 가지로 구분되며 START는 레코드 셋의 삽입을 알린다. 첫 번째는 최초로 버퍼의 데이터가 저장되는 경우로 알고리즘 1과 같다.

[알고리즘 1] 버퍼관리자에서 초기 데이터 삽입 알고리즘

Input
DATA : 분할 전송된 데이터
SID : 세션 식별자

Output
TRUE : 분할 전송 성공
ERR_TOOMANYINPUTS :
전송될 데이터의 크기가 제한 크기를 초과함

Variables
BID : 버퍼 블록 식별자
DSIZE : 전송될 전체 데이터의 크기
MAX_DATA : 데이터의 최대 크기

begin

```
01  DSIZE := getDataSize(DATA)
02  if(DSIZE >= MAX_DATA)
03  {
04    return ERR_TOOMANYINPUTS
05  }
06  BID := GetMappingBuffer(SID)
07  if(BID == -1)
08  {
09    BID := AllocMappingBuffer(SID)
10    InitBuffer(BID)
11    ChangeState(BID,STATE_INSERT)
```

```
12   }
13   else
14   {
15      if(CheckBufferSize(BID,DSIZE)==TRUE)
16      {
17         ChangeState(BID,STATE_INSERT)
18      }
19      else
20      {
21         ChangeState(BID,STATE_FLUSH)
22         BID := AllocMappingBuffer(SID)
23         InitBuffer(BID)
24         ChangeState(BID,STATE_INSERT)
25      }
26   }
27   InsertHeader(BID, Data)
28   return TRUE
29   end
```

최초로 버퍼의 데이터가 삽입되는 경우 데이터의 헤더 정보를 전송받아 디스크에 삽입될 레코드 셋의 크기를 검사한다. 삽입이 가능하면 버퍼의 할당 유무를 검사하여 버퍼가 할당된 경우 버퍼 공간을 검사하여 flag를 'insert'로 변경한 다음 헤더 정보를 복사한다. 만약 버퍼가 할당되지 않은 경우 버퍼 헤더를 초기화하여 삽입을 재실시 한다. 또한 할당 된 버퍼의 공간이 삽입할 레코드의 크기보다 작은 경우 flag를 'flush'로 변경하여 현재 버퍼를 flush 테이블에 등록하고, 새로운 버퍼 블록을 할당받아 삽입 과정을 재실시 한다.
다음은 저장될 레코드의 헤더 정보가 복사 된 이후의 과정으로 알고리즘 2에 기술한다.

[알고리즘 2] 헤더정보 획득 이후의 데이터 삽입 알고리즘

```
Input
   DATA : 분할 전송된 데이터
   SID : 세션 식별자
Output
   TRUE : 분할 전송 성공
   ERR_ALLOCATE : 할당된 버퍼가 존재 안함
Variables
   BID : 버퍼 블록 식별자
   DSIZE : 전송될 전체 데이터의 크기
   MAX_DATA : 데이터의 최대 크기
begin
01   BID := GetMappingBuffer(SID)
02   if(BID == -1)
03   {
04      return ERR_ALLOCATE
05   }
06   InsertData(BID,DATA)
07   return TRUE
08   end
```

삽입 레코드의 헤더 정보가 복사된 이후에는 헤더 정보를 이용하여 해당 버퍼의 데이터를 디스크로 분할 전송하여 삽입하는 과정이 반복된다. 더 이상 삽입할 버퍼의 데이터가 없다면, 버퍼의 헤더 정보를 갱신하고 flag를 'wait'로 변경하고 삽입 과정을 끝낸다. 하지만 삽입 과정 중 오류가 발생한다면, 현재 버퍼를 flush 테이블에 등록하여 . 알고리즘 3은 버퍼 관리자에서 삽입이 정상적으로 끝나는 경우이고, 알고리즘 4는 오류가 발생한 경우를 나타낸다.

[알고리즘 3] 정상적인 종료인 경우 데이터 삽입 알고리즘

```
Input
   DATA : 분할 전송된 데이터
   SID : 세션 식별자
Output
   TRUE : 분할 전송 성공
   ERR_ALLOCATE : 할당된 버퍼가 존재 안함
Variables
   BID : 버퍼 블록 식별자
   DSIZE : 전송될 전체 데이터의 크기
   MAX_DATA : 데이터의 최대 크기
begin
01   BID := GetMappingBuffer(SID)
02   if(BID == -1)
03   {
04      return ERR_ALLOCATE
05   }
06   InsertData(BID,DATA)
07   UpdateHeader(BID)
08   ChangeState(BID,STATE_WAIT)
09   return TRUE
10   end
```

[알고리즘 4] 데이터 삽입 시 오류처리 알고리즘

```
Input
   DATA : 분할 전송된 데이터
   SID : 세션 식별자
Output
   TRUE : 분할 전송 성공
   ERR_ALLOCATE : 할당된 버퍼가 존재 안함
Variables
   BID : 버퍼 블록 식별자
   DSIZE : 전송될 전체 데이터의 크기
   MAX_DATA : 데이터의 최대 크기
begin
01   BID := GetMappingBuffer(SID)
02   if(BID == -1)
03   {
04      return ERR_ALLOCATE
05   }
06   ChangeState(BID,STATE_FLUSH)
07   return TRUE
08   end
```

3.1.2 Flush Thread와 Write Thread

비스텔의 EES 시스템에는 많은 양의 데이터가 입력되고, 이러한 대용량의 데이터를 손실 없이 안전하게 디스크에 기록 및 관리해야 한다. 그러나 디스크의 양은 한정되어 있어서 데이터를 실시간으로 압축하여 디스크에 저장 방법이 요구된다. 그러나 데이터를 압축하는 연산과 디스크에 기록 연산은 모두 고비용의 처리 속도가 느리다. 따라서 본 논문의 저장 관리자는 두 연산을 각각 다른 쓰레드로 구성한다. Flush Thread는 버퍼 관리자에서 레코드 셋을 가져와서 압축을 하는 역할을 한다. 그리고 Write Thread는 압축이 완료된 레코드 셋을 디스크에 저장하는 역할을 한다.

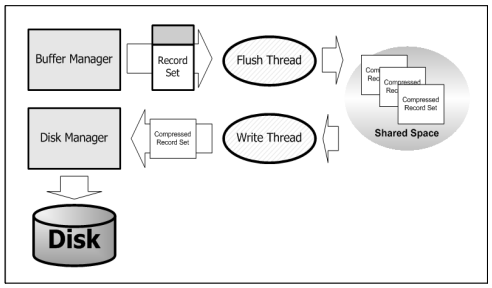

그림 7. 저장 관리자의 Flush Thread와 Write Thread 구성도

Flush Thread와 Write Thread가 공유하는 데이터는 그림 8과 같은 데이터를 사용한다. bHaveData는 현재 압축된 레코드 셋을 가지고 있는지를 나타내며, GeneratedTime은 압축된 레코드 셋이 입력된 시점의 시간, CompRecSet은 압축된 레코드 셋을 가지고 있다. 또한 bWrite은 해당하는 압축된 레코드 셋이 디스크에 안전하게 기록 되었는지 나타낸다.

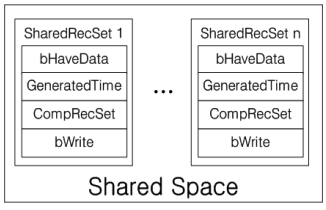

그림 8. 저장 관리자에서 Flush/Write Thread를 위한 공유 자료구조

Flush Thread는 데이터를 가져와서 압축을 하기 전에 공유 공간에 디스크에 안전하게 기록된 SharedRecSet이 있는지 검색한다. bHaveData와 bWrite가 TRUE인 SharedRecSet 중에서 GeneragedTime이 가장 작은 데이터, 즉 가장 처음 생성된 SharedRecSet을 찾고, CompRecSet에 해당하는 레코드 셋을 버퍼에서 삭제하게 된다. 이 과정은 그림 9와 같다.

디스크에 쓰기 완료된 레코드 셋을 버퍼 관리자에서 삭제 한 후, 버퍼 관리자에서 완성된 레코드 셋 한 개를 가져온다. 그림 9의 (a)에서 압축을 실행하고 공유 레코드 셋에서 bHaveData가 FALSE인 것을 확인해서 SharedRecSet 한 개를 얻는다. 다음으로 CompRecSet에 압축된 레코드 셋을 저장하고 GeneratedTime을 현재 시간으로 설정 한 후, bHaveData가 TRUE로 변경된 것을 그림 9의 (b)에서 확인할 수 있다.

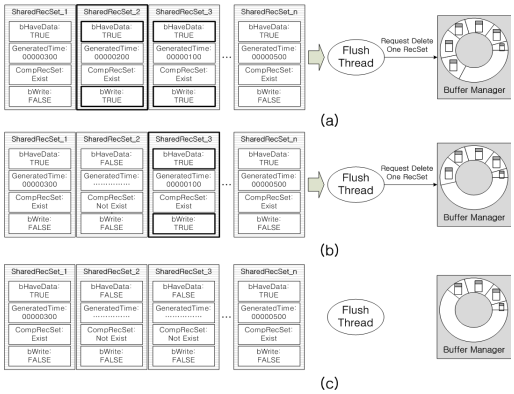

그림 9. 공유 공간에서 레코드 셋 삭제 과정

Write Thread는 반대로 공유 레코드 셋에서 bHaveData가 TRUE인 SharedRecSet 중에서 가장 오래 사용된 것을 가져와서 CompRecSet에 저장된 압축된 레코드 셋을 디스크에 저장한다. 그림 10의 (a)는 레코드 셋이 저장되는 과정을 나타낸다. 그림 10의 (b)는 bWrite을 TRUE로 설정하는 과정을 설명한다.

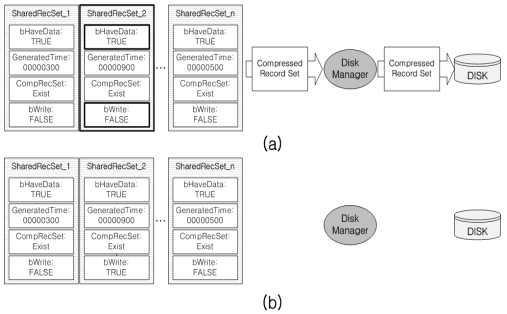

그림 10. 공유 공간에서 레코드 셋 저장 과정

3.1.3 디스크 관리자

디스크 관리자는 레코드 셋들을 디스크로 저장하고, Select를 위한 데이터 전송 및 디스크에서 오래된 파일들을 삭제하는 역할을 담당한다. 비스텔 로그 시스템에서는 실시간으로 대용량의 데이터가 입력되어 모든 데이터를 디스크에 저장하는 것은 비효율적이다. 따라서 데이터들을 압축하여 디스크에 기록하며, 디스크 관리자는 이러한 압

축된 레코드들을 셋 단위로 연산이 이루어진다.

디스크에 실시간 압축 저장을 위한 파일은 크게 헤더, 인덱스 부분, 데이터 부분으로 나뉘어져 있다. 하나의 파일은 한 테이블에 관한 정보만 기록된다. 따라서 헤더에는 파일에 저장되고 있는 테이블의 이름이 관리된다. Select 질의에는 시간 정보가 존재하고, 파일에는 현재 파일에 기록된 레코드들의 가장 최근 생성 시간과 가장 오래된 생성 시간이 기록되어 있다. 이러한 파일 정보의 시간 값을 이용하여 Select 질의의 수행 시 질의 시간에 포함된 파일만을 선택하여 검색할 수 있다. 파일에 저장되는 데이터는 압축된 레코드 셋이다. 따라서 이러한 압축된 레코드 셋들을 가리키는 파일 오프셋 정보가 색인에 기록된다. 또한 인덱스의 개수가 많아서 인덱스 Part의 크기가 최대치가 되었을 경우 현재의 데이터 Part 뒤에 인덱스 Part 한 개를 추가적으로 생성한다. 그리고 이전 인덱스 Part에 있는 Next Index Part Position을 새로 생성된 인덱스 Part로 연결한다. 인덱스에는 인덱스가 가리키는 압축된 레코드 셋의 위치뿐만 아니라 참조하는 압축된 레코드 셋에 존재하는 레코드들의 최대, 최소 시간이 관리된다. 이는 Select 질의 시 압축된 레코드 셋을 디스크에서 읽을 것인지 또는 다음 인덱스를 검색 할 것인가를 판별하는 정보로 사용된다. 또한 인덱스 정보 중 bReal은 Select 질의 시에 버퍼를 읽고 디스크를 읽을 때, 중복된 레코드 셋을 읽는 빈도를 줄이는 목적으로 사용된다.

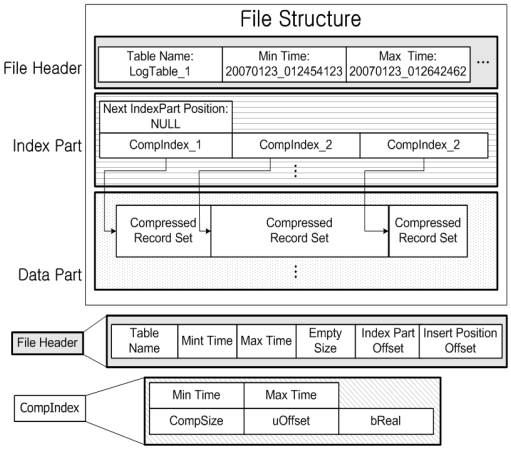

그림 11. 디스크에서의 파일 저장 구조

데이터가 입력 될 때마다 파일의 크기를 확장하는 것보다는 요구되는 크기의 파일을 미리 생성해서 데이터를 저장하는 방법이 실행 속도를 높일 수 있다. 그러나 디스크 관리자에서 파일이 필요할 때마다 일정한 크기의 파일을 할당 받아오는 것 또한 시간 비용이 큰 연산이므로, 파일이 필요하기 전에 시스템에 정의된 충분한 크기의 파일을 생성하고, 저장 공간이 필요할 때 생성되어져 있는 파일을 돌려주는 파일 풀을 사용한다.

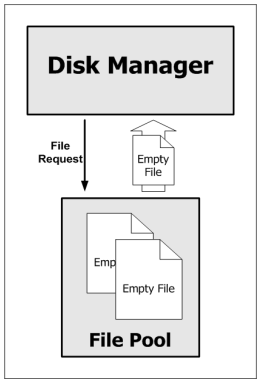

그림 12. 파일 생성 비용을 줄이기 위한 파일 풀 구조

데이터 입력의 단위는 압축된 레코드 셋이다. 압축된 레코드 셋이 입력되면 현재 파일에 입력을 할 수 있는지 파일의 크기를 읽는다. 현재 파일에 입력 할 수 있으면 파일 헤더를 읽어 현재 입력될 파일의 오프셋을 얻어낸다. 다음으로 압축된 레코드 셋을 파일에 입력하고 인덱스 Part에 새로운 인덱스를 생성 한다. 이때 인덱스의 개수가 많아서 인덱스 Part의 크기를 초과 하는 경우에는 새로운 인덱스 Part를 생성하여 인덱스를 생성하게 된다.

데이터 검색을 위해 레코드 셋을 요구할 경우에는 원하는 시간 범위와 파일 이름이 필요하다. 파일 이름이 들어오면 해당 파일을 열고, 질의 범위에 포함되는 인덱스들을 선택한다. 그 다음 레코드 셋 한 개씩 Query Executor에 넘겨주게 된다. 그림 13은 한 파일에서 Select 질의에 필요한 레코드 셋을 넘겨주는 과정을 나타낸 것이다. 쿼리에 있는 시간 범위에 해당하는 인덱스를 찾은 다음, 그 인덱스를 이용하여 압축된 레코드 셋을 찾는다. 그리고 그 압축된 레코드 셋의 압축을 풀고 Query Executor로 전송하여 준다.

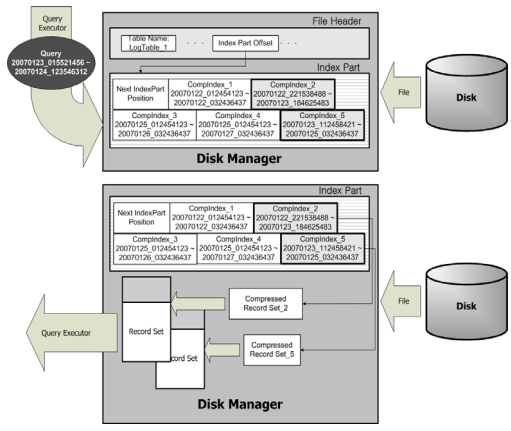

그림 13. 파일 Select 질의 수행 시 필요한 레코드 셋 전달 과정

파일의 삭제는 Delete Thread로부터 한 파일의 삭제를 요청 받으면 해당 파일이 디스크에 존재하는지와 현재 사용 중 인지를 검사한 후 삭제연산을 수행한다.

4. 성능평가

본 장에서는 제안된 저장 관리자의 성능비교 평가를 위해 실시간 대용량의 데이터스트림 환경에서 기존의 데이터 저장을 위한 DBMS와 대용량 데이터의 벌크 삽입 성능을 비교 평가 하였다. 실험 환경은 표 1과 같다.

표 1. 실험 환경

CPU	Intel Pentium Core2Quad Q6600 2.4GHz
RAM	4GB DDR2(PC2-6400)
OS	Microsoft Windows XP Service Pack3
Development	Microsoft Visual Studio 2005

본 실험은 벌크 삽입이 가능한 Oracle10g, MSSQL Server2003과 제안 시스템을 비교 평가 하였다.

실험 방법은 삽입할 레코드 셋의 크기를 증가 시키면서 삽입 프로세스가 완료되는 시간을 측정하였으며, MSSQL Server 2003과 Oracle10g의 경우 인덱스를 구축하지 않고 벌크 데이터 셋을 삽입하여 평가 하였다.

삽입 성능 테스트는 표 2의 데이터를 사용하여, 버퍼 메모리에 레코드 셋 사이즈를 실험할 횟수만큼 메인 메모리에 미리 복사하여 준비된 조건에 실행되었다. 실험 평가는 같은 조건에서 20회 반복 실험을 실시하였으며, 그림 14는 이에 대한 평균값의 결과이다.

표 2. 삽입 성능 평가를 위한 데이터 환경

필드 수	7
레코드 크기	1107Byte
전체 레코드 수	10000 ~ 1000000
레코드 사이즈	1MB ~ 100MB

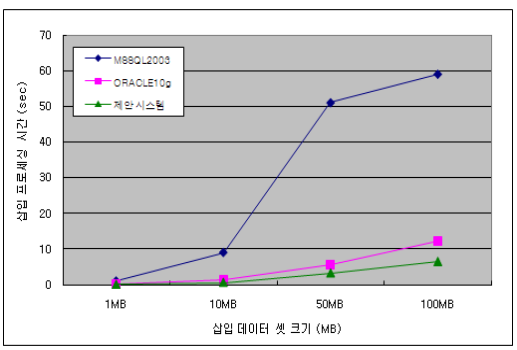

그림 14. 벌크 삽입 성능 실험결과

벌크 데이터 셋에 대한 실험 결과에서 MSSQL Server 2003의 경우 삽입 데이터의 크기가 10MB 이상이 되는 조건부터 성능의 큰 차이를 보이면서 가장 좋지 않은 결과를 보여 주었다. Oracle10g의 경우 비교적 제안시스템과 유사한 성능을 보였지만 100MB 데이터 셋을 삽입할 경우 제안시스템이 45% 우수한 성능을 보였다. 제안 시스템은 블록 단위로 디스크 트랜잭션을 실행하기 때문에 보다 우수한 성능 보인 것으로 판단된다. 또한 제안 시스템은 실시간 데이터 압축과 동시에 벌크 삽입을 한 결과이며, 압축률은 평균적으로 78%를 보여 주었다.

5. 결론 및 향후연구

본 논문에서는 반도체, LCD 생산설비 등에서 발생하는 대용량의 데이터스트림을 실시간 압축 및 벌크 삽입이 가능한 시스템을 설계 및 구현하였다. 제안된 DSMS의 저장 관리자는 비스텔 사의 EES framework 테스트 환경을 기반으로 테스트 되었으며, 실제 관련된 비즈니스 필드에서 일반적으로 사용되는 오라클 보다 성능의 우수함을 실험 평가를 통해 증명하였다. 이는 고속의 대용량 스트림데이터의 처리를 위해 링 버퍼 기반의 독립적인 Flush Thread와 Write Thread 지원하며, 대용량 벌크 데이터의 삽입 성능 최적화 위한 블록단위의 저장 관리자를 설계하여 이를 해결하였다. 이는 반도체 및 LCD 생산 공정에서 관리를 위한 DSMS에서 하루에 처리된 1TB 이상의 데이터를 압축 저장할 수 있으며, 기존에 같은 목적으로 사용된 Oracle10g보다 우수한 삽입 성능과 작은 저장 공간을 요구하였다.

향후 연구로는 압축 저장된 대용량 데이터에 대한 패턴 검색 및 분석에 효과적인 시공간 색인에 대한 연구가 필요하다.

참 고 문 헌

[1] An e-Manufacturing Strategy Need Developed from the Manufacturing Strategy, AMR Research, Inc, August, 2000.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, “Models and Issues in Data Stream Systems”, PODS, 2002.

[3] Coral8CclReference.pdf, www.coral8.com.

[4] Coral8TechnologyOverview.pdf, www.coral8.com

[5] D. Carney, U. Cetinternel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams - A New Cless of Data Management Applications," VLDB 2002.

[6] D. J.Abadi, D. Carney, U. Centintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik, “Aurora : A New Model and Architecture for Data StreamManagement.”

VLDBJournal, 2003.
[7] EES Framework.pdf, www.bistel-inc.com.
[8] Equipment Engineering Capabilities (EEC) guidelines, v2.5, International SEMATECH, ismi.sematech.org.
[9] L. Golab, M. TamerOzsu, "Issues in Data Stream Management", SIGMOD Vol. 32, 2003.
[10] M. Koc, J. Ni, and J. Lee, "Introduction of e-Manufacturing", NAMRC 2003 e-Manufacturing Panel, McMaster University, May, 2003.
[11] S. Chandrasekharan, O. Cooper, A. Deshpande, M. Franklin, J. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Coninuous Dataflow Processing for an UncertainWorld", In Processdings of CDIR, 2003.
[12] Solution Peak Perfromance Overview.pdf, www.bistel-inc.com.
[13] Streambase_RealTimeFederal.pdf,www.streambase.co m.
[14] StreamBaseRealTimeWhitepaperJan2006.pdf, www.streambase.com.
[15] Y. C. Su, F. T. Cheng, M. H. Hung, and H. C. Huang, "Intelligent Prognostics System Design and Implementation", IEEE Transactions on Semiconductor Manufacturing, vol. May 2006.

이 동 욱
2003년 상지대학교 전자계산공학과 (이학사)
2005년 인하대학교 컴퓨터 정보공학과 (공학석사)
2005년~현재 인하대학교 정보공학과 (박사과정)
관심분야 : 유비쿼터스 환경을 위한 공간 데이터베이스 및 데이터스트림 관리시스템, 공간 데이터웨어하우스

백 성 하
2005년 인하대학교 수학통계학부 (이학사)
2007년 인하대학교 컴퓨터 정보공학과 (공학석사)
2007년~현재 인하대학교 정보공학과 (박사과정)
관심분야 : 데이터 스트림 관리 시스템, 데이터베이스 클러스터

김 경 배
1992년 인하대학교 전자계산공학과 (공학사)
1994년 인하대학교 전자계산공학과 (공학석사)
2000년 인하대학교 전자계산공학과 (공학박사)
2004년~현재 서원대학교 컴퓨터교육학과 조교수
관심분야 : 이동실시간 데이터베이스, 스토리지 시스템, GIS

배 해 영
1974년 인하대학교 응용물리학과 (공학사)
1978년 연세대학교 전자계산학과 (공학석사)
1989년 숭실대학교 전자계산학과 (공학박사)
1982년~현재 인하대학교 정보공학부 교수
1999년~현재 인하대학교 지능형GIS연구센터 센터장
관심분야 : 공간 데이터베이스, 멀티미디어 데이터베이스 등