블록 단위 트랜잭션을 이용한 대용량 데이터의 실시간 저장관리기†

Real time Storage Manager to store very large datausing block transaction

백성하* / Baek Sung-Ha, **이동욱**** / Lee Dong-Wook, **어상훈***** / Eo Sang-Hun
정원일**** / Chung Warn-ill, **김경배******* / Kim Gyoung-Bae
오영환****** / Oh Young-Hwan, **배해영********* / Bae Hae-Young

요 약

초당 최소 5만 건에서 50만 건이 넘는 삽입트랜잭션이 발생하는 반도체 자동 생산 공정 시스템은 대량의 데이터를 실시간으로 저장하는 저장관리시스템을 필요로 한다. 대용량의 데이터를 빠르고 안정적으로 저장하기 위해서 많은 저장관리시스템이 연구되었다.

기존의 저장관리시스템은 대표적으로 전형적인 디스크 기반 DBMS가 있다. 그러나 디스크 기반 DBMS는 초당 50만 건의 삽입트랜잭션 처리는 매우 어렵다. 그래서 디스크 기반 DBMS의 성능을 향상시키기 위해 데이터를 디스크가 아닌 메인메모리를사용하는 메인메모리 DBMS가 등장하였다. 그러나 메인메모리 DBMS는 메인메모리 용량의 한계로 인해 대용량 데이터를 저장하는 것은 어렵다.

본 논문에서는 초당 5만 건 이상의 삽입트랜잭션을 지원하고 대용량 데이터를 저비용으로 저장하기 위해 블록단위의 삽입 트랜잭션을 사용한 저장관리시스템을 제안한다. 블록단위의 삽입 트랜잭션은 개별 튜플 단위의 로그기록 비용과 인덱스 생성비용을 블록단위로 변경시켜 비용을 크게 감소시킬 수 있다. 또한 제안시스템은 데이터를 압축 저장하여 저장 비용을 감소시킬 수 있다. 그러나 압축기법은 데이터의 필드정보가 유실되어 모든 데이터의 압축을 해제하는 비용이 발생한다. 이 문제를 해결하기 위해 제안시스템은 압축 시 압축되는 블록의 인덱스를 생성하여 데이터 검색 속도를 향상시켰다. 본 제안시스템은 반도체 공정에서 빠르게 발생하는 대용량 데이터를 고속으로 저장할 수 있고, 디스크 저장비용을 감소시킬 수 있다.

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

■ 논문접수 : 2008.1.23 ■ 심사완료 : 2008.4.15

* 인하대학교 대학원 컴퓨터 정보공학과 박사과정(shbaek@dblab.inha.ac.kr)
** 인하대학교 대학원 컴퓨터 정보공학과 박사과정(dwlee@dblab.inha.ac.kr)
*** 인하대학교 대학원 컴퓨터 정보공학과 통합과정(eosanghun@dblab.inha.ac.kr)
**** 교신저자 호서대학교 정보보호학과 전임강사(wnchung@hoseo.edu)
***** 서원대학교 컴퓨터교육학과 조교수(gbkim@seowon.ac.kr)
****** 나사렛대학교 컴퓨터공학부 조교수(yhoh@kornu.ac.kr)
******* 인하대학교 대학원장(hybae@inha.ac.kr)

Abstract

Automatic semiconductor manufacture system generating transaction from 50,000 to 500,000 per a second needs storage management system processing very large data at once. A lot of storage management systems are researched for storing very large data.

Existing storage management system is typical DBMS on a disk. It is difficult that the DBMS on a disk processes the 500,000 number of insert transaction per a second. So, the DBMS on main memory appeared to use memory. But it is difficultthat very large data stores into the DBMS on a memory because of limited amount of memory.

In this paper we propose storage management system using insert transaction of a block unit that can process insert transaction over 50,000 and store data on low storage cost. A transaction of a block unit can decrease cost for a log and index per each tuple as transforming a transaction of a tuple unit to a block unit. Besides, the proposed system come cost to decompress all block of data because the information of each field be loss. To solve the problems, the proposed system generates the index of each compressed block to prevent reducing speed for searching. The proposed system can store very large data generated in semiconductor system and reduce storage cost.

주요어 : 반도체, 자동화 시스템, 저장관리시스템, 저장관리기, 벌크 삽입, 삽입 트랜잭션, 대용량 데이터, 실시간 저장

Keyword : Semiconductor, automatic system, storage management system, Storage Manager, Bulk Insert, Insert Transaction, very large data, real-time storage

1. 서 론

반도체나 FPD 생산 공정은 생산성 향상을 위하여 무인 자동화 생산을 많이 요구한다. 이런 자동화는 생산 장비에서 발생하는 데이터를 저장 관리하고 생산성 향상을 위해 각종 응용에 이용한다.[1] 응용시스템은 실시간으로 에러를검출하여 자동화 공정을 최적화하는FDC(Fault Detection and Classification)나 진보된 PM(Preventive Maintenance), BM (breakdown maintenance)을 지원하는 PMM 및 기타 응용들이 있다.[2] 이와 같은 각 응용시스템에 장비에서 발생한 데이터를 제공하는 저장관리시스템이 필요하다.

응용에서 사용되는 이 데이터는 데이터의 규모가 매우 크고 빠르게 발생한다. 특히 초 미세 나노공정에 적용되는 차세대 공정제어 장비는 초당 5만에서 50만 건 이상의 삽입 트랜잭션이 발생한다.[2] 각 응용에서는 이와 같은 데이터들을 실시간으로 요청할 수 도 있고 필요할 때마다 요청할 수도 있다. 효율적인 자동화 공정을 위해 초 대용량 데이터를 실시간으로 저장하고 검색이 가능한 저장 관리기가 필요하다.

대용량 데이터를 처리하는 기존의 시스템은 오라클, INFOMIX, MYSQL과 같은 디스크 기반 데이터 베이스 관리 시스템 (DBMS) 이 있다.[3] 그러나 기존 디스크 기반 DBMS는 초당 5만 건 이상, 최대 50만 건 정도의 삽입 트랜잭션의 처리는 거의 불가능하다. 그래서 보다 빠른 트랜잭션 처리를 위해 데이터 저장장치로 접근성이 느린 디스크 대신 메인메모리를 사용하는 메인메모리 DBMS가 등장하였다.[4,5] 메인메모리 DBMS는 디스크 기반 DBMS에 비해 트랜잭션 처리 속도를 크게 향상시켰다. 그러나 메인메모리 DBMS역시 삽입 트랜잭션 발생시 디스크에 로그를 기록해야 하고 개별 트

랜잭션마다 로그를 기록하는 것은 큰 부하를 발생시킨다. 또한 메인 메모리의 용량 한계로 인해 대용량 데이터를 모두 메모리에 상주시키는 것은 불가능하다. 최근에는 대용량 데이터를 처리하기 위해 데이터 스트림 관리 시스템(DSMS)가 등장하였다.[8,9,10,11] 그러나 DSMS는 시스템에 입력되는 대용량 데이터를 모두 저장하는 것이 목적이 아니고, 입력된 데이터 중 필요한데이터를 최대한 빠르게 검색하는 것이 목적이다.

따라서 본 논문에서는 초당 5만 건 이상의 삽입 트랜잭션을 처리 하고, 대용량 데이터를 저렴한 비용으로 디스크에 저장할 수 있는 저장관리기를 제안한다. 본 논문에서는 로그 기록 횟수를 감소시켜 삽입 트랜잭션의 처리 속도를 향상시키기 위해 튜플 단위의 트랜잭션을 사용하지 않고 튜플을 모아서 한번에 처리하는 블록 단위의 트랜잭션을 사용한다. 본 논문에서 제안하는 저장관리기는 이 블록 단위의 트랜잭션을 사용하기 때문에 기존 DBMS의 버퍼 관리기의 구조와 파일구조를 블록 단위 트랜잭션 처리에 적합하도록 개선해야 한다. 또한 대용량 데이터를 디스크에 저렴한 비용으로 저장하기위해서 저장 시 압축기법을 사용한다. 그런데 데이터를 압축하면 데이터의 필드 정보가 유실되어 검색 트랜잭션 처리시 모든 데이터의 압축을 해제해야 한다. 이와 같은 문제를 해결하기 위해서 압축 시에 데이터의 시간정보를 기반으로 로컬인덱스를 생성하여 데이터의 일부만 압축해제가 가능하도록 한다. 본 논문에서 제안하는저장관리자를 반도체 자동화 공정에 적용하면 데이터 저장 속도를 향상 시킬수 있고, 데이터 저장 비용 또한 감소 시킬 수 있다.

2. 관련연구

2.1 메인메모리 데이터 베이스

기업 서비스의 전산화로 인하여 증가하는 트랜잭션의 실시간처리 요구가 증가하면서 전통적인 디스크 기반 DBMS는 처리능력의 한계에 도달하였다. 이와 같은 한계를 극복하기 위하여 고성능 하드웨어를 사용하기 시작했지만, 처리 능력 향상에도 한계가 있고 하드웨어 비용이 증가하는 문제가 있었다. 그래서 디스크보다 처리 속도가 빠른 메모리를 저장장치로 사용한 DBMS가 등장하였다.[4,5] 메인메모리 DBMS는 데이터베이스를 디스크가 아닌 메모리에 상주시켜 디스크 IO를 크게 감소시켜 디스크기반 DBMS에 비해 트랜잭션 처리 속도를 크게 향상시켰다. 그러나 메모리의 한정된 공간으로 인해 대용량 데이터의 저장 시 매우 많은 비용을 소모하게 된다. 특히 대용량 데이터의 고속 처리를 요구하는 반도체 생산 공정 환경에서는 저장 공간이 한정적인 메인메모리 DBMS는 적합하지 않다

2.2 튜플 단위의 삽입 트랜잭션

전형적인 DBMS는 기본적으로 튜플 단위의 삽입 트랜잭션을 지원한다. Insert Into 테이블명 values (val1, val2, …)와 같은 형태의 SQL 질의를 가지고 하나의 튜플을 디스크에 반영한다. 이와 같은 삽입 트랜잭션의 요청이 있는 경우 DBMS는 버퍼관리기의 여분에 버퍼블록에 디스크에 저장된 페이지 중 저장 가능한 페이지를 하나 읽어 들이고 이 메모리블록에 튜플을 기록한다. 그리고 트랜잭션 처리 내역을 로그로 디스크에 기록하고 후에 이 버퍼블록을 디스크에 반영하는 방식으로 튜플을 기록한다.[12,13] 이와 같이 트랜잭션마다 로그를 디스크에 남기는 방식은 로그 저장을 위한 비용이 많이 필요하다. 또한 전형적인 DBMS는 인덱스로 B+트리를 사용하는데, B+트리는 삽입 시 인덱스 노드를 재구성 하는 경우가 발생한다.[14] 그러므로 검색을 위해 인덱스를 생성하는 경우 저장 성능이 크게 감소한다. 이와 같은 인덱스 노드 재구성 비용을 줄이기 위해 벌크 삽입기법을 사용하여 인덱스를 한번에 구성하여 재구성 비용을 감소시킬 수 있다.[14,15] 그러나 이 방식 역시 메모리에서 인덱스를 재구성하는 인덱스 생성 비용이 필요하다.

그러므로 튜플 단위의 삽입트랜잭션 처리는 인덱스 재구성 비용과 로그기록 비용으로 인해초당 5만

건이 넘는 반도체 생산 장비의 데이터를 실시간으로 저장 할 수가 없다. 그래서 전형적인 튜플 단위의 삽입트랜잭션과 다른 트랜잭션 처리 방법이 필요하다.

2.3 데이터 스트림 관리 시스템

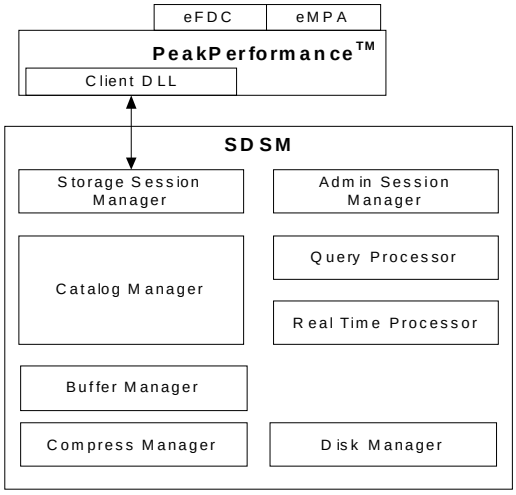
증권 시세 표시기, 네트워크 트래픽, 웹 로그 분석, 실시간 센서 데이터와 같이 시간에 따라 변하고 연속적으로 생성되는 대용량의 데이터를 데이터 스트림이라 한다. 새롭게 요구되고 있는 이 데이터 스트림을 처리하기 위해서 데이터 스트림 관리 시스템(DSMS)이 연구되었다.[8,9,10,11] 데이터 스트림 관리 시스템은 시간에 따라 변하는 실시간 대용량 데이터 모델을 처리 하기 위해 새로운 형태의 질의인 연속질의의 사용을 제안하였다. 또한 시스템의 가용 작업량을 초과하는 데이터 스트림의 처리 요청 시, 작업에 일부를 포기하는 부하분산 기법(Load Shedding)을 사용 할 수 있다. 그리고 정확한 데이터 보다 근사치의 통계량을 원하는 경우 근사화 기법을 사용할 수 있다. 이와 같은 기능을 포함하는 데이터 스트림 관리 시스템은 새롭게 등장한 데이터 스트림 처리에 매우 적합한 시스템으로 현재 많은 연구가 진행되었다. 대표적인 연구로 NiagaraCQ, TelegraphCQ, Stream, Aurora 등이 있다.

그러나 데이터 스트림 관리 시스템은 연속질의를 이용하여 연속적으로 발생하는 대용량 데이터 중 필요한데이터를 실시간으로 검색하는 것이 목적이고, 이 데이터를 모두 디스크에 저장하는 것이 목적이 아니다. 그러므로 데이터 스트림 관리시스템을 실시간으로 모든 데이터를 저장가능 하도록 확장하는 연구가 필요하다.

3. 본론

3.1 시스템 구조

본 논문에서는 반도체 생산 장비에서 발생하는 대용량 데이터를 실시간으로 저장하고, 검색기능을 제공하는 “반도체 데이터 저장관리자(SDSM: Semiconductor Data Storage Manager)”를 제안한다. SDSM는 반도체 장비로부터 수집된 데이터를 가지고 로그를 생성하는 프로세서인 PeakPerformance에서 초당 약 5~10만개의 로그 데이터를 받아서 저장하고, 검색 및 필터링 기능을 제공한다. PeakPerformace는 장비와 접속하여 장비에서 발생하는 데이터를 1차적으로 수집하고 그 중 대량의 로그 데이터를 SDSM에 전송하는 시스템이다. SDSM은 이 대량의 데이터를 받아 디스크에 실시간으로 압축 저장한다. 또한 SDSM은 반도체 장비의 에러를 실시간으로 검출하기 위해서, 에러에 해당하는 필터링 조건을 등록하고 이 조건에 해당하는 데이터를 실시간으로 추출하는 기능과 디스크에 저장된 과거의 데이터를 검색하는 기능을 제공한다.



<그림 1> SDSM의 시스템구성도

<그림 1>은 SDSM의 시스템 구성도이다. 대량의 로그 데이터를 저장하기 위한 SDSM의 중요한 컴포넌트는 버퍼관리기(Buffer Manager), 압축관리기(Compress Manager), 디스크관리기(Disk Manager)이다.

SDSM은 디스크 IO감소를 통한 고속저장을 위

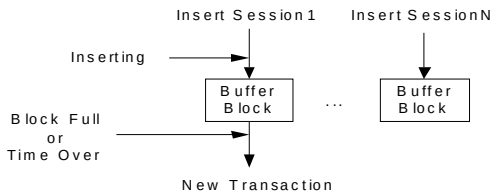
해 튜플의 삽입 요청이 발생할 때즉시 디스크에 기록하지 않고, 저장되는 튜플을 메모리의 일정 크기의 버퍼블록에 저장하고 블록이 가득 차거나 특정 주기 마다 동시에 디스크에 기록하는 게으른(Lazy) 저장방식을 사용한다. 버퍼관리기는 이 게으른 저장방식을 지원하는 자료구조를 사용한다. 이 버퍼관리기는 튜플 별로 트랜잭션의 커밋 롤백을 지원하지 못하고, 버퍼블록단위의 트랜잭션을 생성하여 커밋 롤백을 지원 할 수 있다. 반도체 장비들은 빠르고 대량으로 삽입요청을 하기 때문에 블록단위의 트랜잭션 관리는 저장 속도의 향상을 보장한다.

그리고 SDSM는 주기적으로 디스크에 기록할 때 해당 버퍼블록을 압축한다. SDSM가 저장하는 데이터는 하루에 수 기가바이트 이상이기 때문에, 데이터를 압축하지 않으면 디스크 비용이 크게 증가한다. 또한 대용량 데이터 저장을 위해서는 디스크 IO가 매우 많이발생하기 때문에 버퍼블록을 압축하면 디스크 IO수를 감소시킬 수 있다. 그러나 버퍼블록이 압축되면 검색 시 반드시 압축을 해제해야 하는 추가 비용이 필요하다. 만약 압축된 모든 버퍼블록을 해제하고 데이터를 검색하면 매우 많은 시간이 소모된다.

이 문제를 해결하기 위해서 SDSM에서는 데이터의 발생 시간으로 구성된 로컬인덱스를 사용한다. 모든 장비의 로그 데이터는 데이터 발생 시간을 포함하고 있다. 그리고 이 로그 데이터의 정보를 요청하는 응용시스템은 로그의 양이 매우 거대하기 때문에 일정 시간범위를 가지고 데이터를 검색한다. 그러므로 버퍼블록에 저장된 모든 로그데이터들의 시간 중 가장 빠른 시간과 늦은 시간을 버퍼블록의 시간 범위로 저장하고, 이 시간범위를 가지고 로컬 인덱스를 구성한다. 로컬 인덱스를 이용하여 시간범위를 포함한 검색 질의 처리 시, 모든 압축된 버퍼블록의 압축을 해제하지 않고 필요한 블록만 해제 할 수 있다.

3.2. 대용량 데이터 저장을 위한 버퍼 관리기

SDSM은 고속 저장을 위해 게으른 저장방식을 사용한다. 전통적인 데이터베이스의 버퍼관리기는 메모리에 페이지 구조를 연속적인 배열 형태로 할당하고 LRU, FIFO와 같은 버퍼 교체정책을 사용하여 다수의 트랜잭션이 이용 가능하도록 한다. 그러나 SDSM은 트랜잭션의 단위가 튜플 단위가 아니고 버퍼블록 내에 저장된 다수의 레코드의 모임과 같은 집합형태 이므로 이와 같은 트랜잭션 형태에 최적화된 버퍼 관리구조를 사용한다.

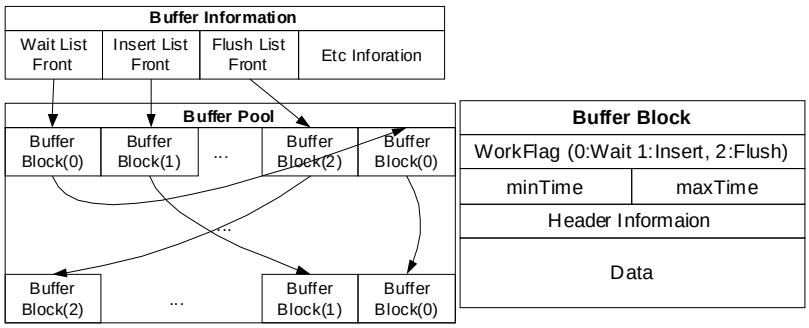


<그림 2> 삽입 트랜잭션의 생성과정

SDSM은 전형적인 DBMS와 다른 버퍼블록단위의 삽입트랜잭션을 사용한다. <그림 2>는 SDSM의 트랜잭션의 생성과정이다. SDSM은 클라이언트가 데이터 삽입을 위한 삽입세션(Insert Session)을 생성하여 데이터를 삽입하면 바로 삽입트랜잭션을 생성하지 않고 버퍼블록에 저장한다. 그리고 이 데이터가 버퍼블록의 공간을 모두 채우거나 삽입되는 레코드가 지정된 삽입가능 시간을 초과하는 경우, 현재까지 버퍼블록에 저장된 데이터를 가지고 하나의 삽입트랜잭션을 생성한다. 그리고 생성된 삽입트랜잭션 단위로 디스크에 데이터를 반영한다.

<그림 3>은 버퍼관리자의 구조와 버퍼블록의 자료구조이다. 버퍼블록은 시간기반의 로컬인덱싱 구성을 위해 minTime과 maxTime을 가지고 있고, 버퍼블록의 상태를 위한 WorkFlag를 가지고 있다. WorkFlag는 0일 때 대기 상태인 버퍼를 나타내고, 1인 경우 삽입 세션을 통해 삽입트랜잭션이 생성중임을 나타내고, 2인 경우는 삽입 트랜잭션으로 저장 과정을 수행 중임을 나타낸다. Header Information은 버퍼블록의 기타 정보가 저장되고, Data에는 실제 데이터가 저장된다.

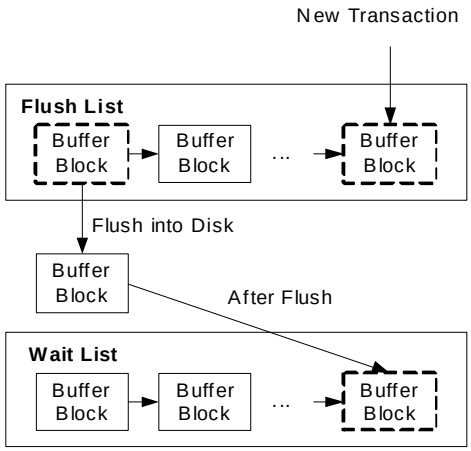
버퍼 풀(Buffer Pool)은 버퍼블록을 메모리에 생성해둔 것이다. 버퍼 정보(Buffer Information)



<그림 3> 버퍼 블록의 자료구조와 버퍼관리자 구조

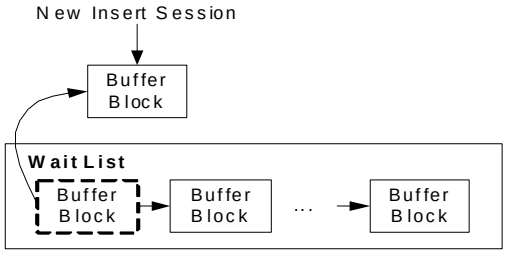
에는 Wait List Front, Insert List Front, Flush ListFront의 3가지 정보가 저장된다. Wait List Front는 버퍼 풀에서 사용 중이 아닌 버퍼블록의 리스트의 첫 번째 위치를 가리킨다. Insert List Front는 현재 삽입세션에 의해 삽입트랜잭션이 생성중인 버퍼블록들의 리스트의 첫 번째 위치를 가리킨다. Flush List Front는 디스크에 저장 작업을 수행중인 삽입트랜잭션들의 첫 번째 위치를 가리킨다.

버퍼 관리자는 위와 같은 자료구조와 삽입 트랜잭션을 가지고 버퍼 풀의 한정된 버퍼블록을 관리한다. 위에 3가지 리스트와 같이 버퍼관리기는 3가지 버퍼 상태를 가지고 버퍼교체 정책을 지원한다.



<그림 4> 삽입 트랜잭션의 수행과 Flush List 관리

<그림 4>는 삽입트랜잭션이 발생 했을 때 Flush List에 버퍼블록이 추가되는 모습이다. 삽입트랜잭션은 버퍼블록의 상태를 Insert에서 Flush로 바꾸고 Flush List의 가장 마지막에 추가된다. 그리고 Flush가 완료되면 이 버퍼블록을 Wait List의 마지막에 추가하여 다른 삽입세션이 이용 가능하도록 한다.



<그림 5> 새로운 삽입세션을 위한 빈 버퍼블록의 할당

<그림 5>는 새로운 삽입세션을 위해 버퍼 풀 내의 빈 버퍼블록을 할당하는 과정이다. Wait List 내의 있는 첫 번째 버퍼블록을 삽입세션에 할당하고 버퍼블록의 상태를 Insert로 변경한다. 그리고 할당 받은 버퍼에 데이터를 삽입하여 새로운 삽입 트랜잭션을 생성한다. 이와 같이 SDSM은 대용량 데이터 저장을 지원하기 위해 블록단위의 트랜잭션을 지원하고 이 트랜잭션을 위한 버퍼관리 정책을 지원한다.

<알고리즘 1>은 데이터 삽입을 위한 알고리즘이다. 데이터 저장을 위해서는 우선 GetEmptyBlock 함수를 이용하여 Wait List에서 빈 블록을 할당 받

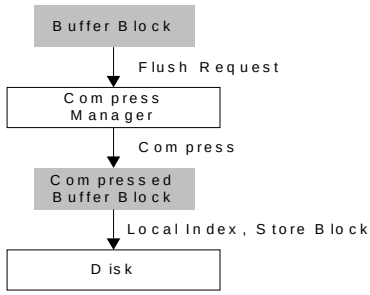
는다. 그리고 주어진 삽입 세션에 연결 하고(3줄), GetRecord를 이용하여 해당세션에서 레코드 Rec를 입력 받는다.(6줄) 그리고 입력받은 레코드 Rec를 버퍼블록 B_E에 저장한다.(7줄) 이 과정을 반복하면서 버퍼블록이 가득 차거나(8줄), 저장 알고리즘 시간이 지정된 삽입시간(maxTime)을 초과하면(10줄) 레코드 삽입을 중지하고, 레코드들이 저장된 버퍼블록(B_E)을 가지고 새로운 트랜잭션을 생성한다.(13줄) 그리고 FlushTransaction 함수를 통해 생성된 트랜잭션 TID를 Flush요청 할 수 있다. Flush 요청을 하면 트랜잭션 TID의 버퍼블록 B_E가 Flush List의 마지막에 기록된다.

<알고리즘 1> 데이터 삽입 알고리즘

```
Input
  SE : Insert Session
Output
  True or False
Procedure Insert()
Initialize
01 B_E = GetEmptyBlock ();
Begin
01 Insert()
02 {
03  ConnectInesrtSession(SE)
04  while(1)
05  {
06    Rec = GetRecord(SE)
07    InsertRec(B_E,Rec)
08    if( B_E.IsFull() == true )
09       break;
10    if( Time > maxTime )
11       break;
12  }
13  TID = MakeNewTransaction(B_E)
14  FlushTransaction(TID)
15 }
End
```

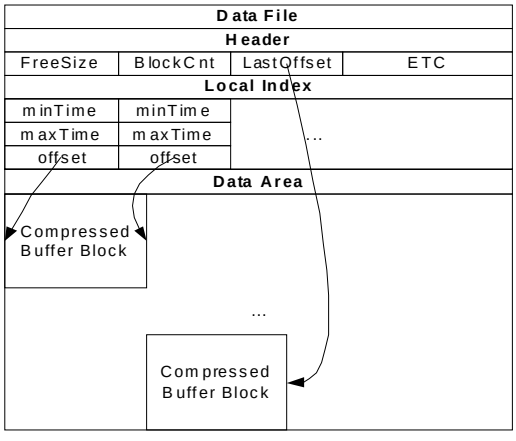
3.3 압축 저장 구조와 로컬 인덱싱

SDSM은 버퍼블록 단위의 저장을 지원하고 시간 기반 검색의 고속 탐색을 지원하고 증가하는 대용량 데이터를 최저비용으로 저장하기 위해 압축 저장과 디스크 관리 구조를 사용한다. SDSM는 디스크 IO감소와 디스크 공간을 절약하기 위해 저장시 데이터 압축을 사용한다. 또한 검색 시 모든 압축블록을 해제하지 않기 위해 압축된 블록에 대한 로컬 인덱스를 생성한다.



<그림 6> SDSM의 압축저장 과정

<그림 6>는 SDSM의 압축저장 과정이다. 3.2절에서 버퍼관리자에서 생성된 삽입 트랜잭션은 Flush List에 추가되고 순차적으로 디스크에 반영한다고 설명하였다. 이 디스크의 반영 과정이 <그림 6>이다. 먼저 압축관리자(Compress Manager)에서 삽입 요청을 한 버퍼블록을 압축하여 압축된 버퍼블록(Compressed Buffer Block)을 생성한다. 이때 압축된 버퍼블록의 minTime과 maxTime을 임시로 보관하고 디스크에 압축된 버퍼블록을저장한다. 그 후에 이 minTime과 maxTime을 저장한 압축된 버퍼블록의 로컬인덱스로 생성하여 인덱스 영역에 저장한다.



<그림 7> SDSM의 디스크 저장 구조

<그림 7>은 SDSM의 디스크 저장구조이다. 하나의 데이터 파일(Data File)은 헤더(Header)와 로컬 인덱스 영역(Local Index)과 데이터 영역(Data Area)으로 나누어져 있다. 헤더에는 데이터 파일의 운영정보가 저장된다. 헤더에 저장되는 값은 파일의 남은 공간(FreeSize)과 저장된 블록 수(BlockCnt)와 파일내의 마지막 위치(LastOffset)가 있다. 로컬인덱스 영역에는 각 압축블록의 시간정보(minTime, maxTime)와 해당블록의 디스크 내의 위치(offset)을 저장한다. 그리고 데이터 영역에는 실제로 압축된 버퍼블록이 저장된다.

이와 같은압축저장 방식과 디스크 저장 구조를 사용하면 반도체 저장환경의 특징인 대용량 데이터의 실시간 저장 및 시간 기반 검색에 대한 요구사항을 처리하기 유용하다. 또한 압축 저장은 저장되는 데이터의 양을 대폭 감소시켜 디스크 비용과 디스크 IO를 감소시킨다..

<알고리즘2> 삽입트랜잭션의 Flush 알고리즘

```
Input
TID : Insert Session
Output
True or False
Procedure FlushTransaction()
Initialize
01 B_F = GetFlushBlock (TID);
Begin
01 FlushTransaction ()
02 {
03  B_C = CompressBlock(B_F);
04  Header.FreeSize = Header.nFreeSize +
    GetBlockSize(B_C)
05  Header.BlockCnt = Header.BlockCnt + 1
06  LastOffset = Header.LastOffset
07  minTime = GetMinTime(B_F);
08  maxTime = GetMaxTime(B_F)
09  Index = MakeIndex(minTime,maxTime,
     LastOffset)
10  Store(Header)
11  Store(Index)
12  Store(B_C,LastOffset)
15 }
End
```

<알고리즘 2>는 디스크에 버퍼블록을 반영하기 위한 알고리즘이다. 우선 입력된 TID에 해당하는 버퍼블록 B_F를 얻는다. 그리고 압축 함수 CompressBlock을 이용하여 버퍼블록 B_F를 B_C로 압축한다.(2줄) 그리고 데이터파일의 헤더의 남은크기(FreeSize)와 블록의수(BlockCnt)를 갱신한다.(4~5줄) 그리고 헤더에서 데이터파일의 마지막 위치를 얻고, 블록의 시작시간과 마지막 시간을 얻고 인덱스를 생성한다.(6~9줄) 그리고 헤더와 인덱스, 압축된 블록을 차례로 디스크에 저장한다.(10~12줄)

지금까지 본 논문에서 제안 하는블록단위의 트랜잭션을 사용하고 압축 저장을 하여 저장 속도를 향상시키고 저장비용을 감소시키는 저장관리자에 대해 설명하였다. 본 논문에서 자세히 언급하지는 않았지만 제안된 저장관리자도 일반적인 DBMS 에서 지원하는 회복기법을 사용 가능하다. 제안된 저장관리기는 블록 단위의 트랜잭션을 사용하기 때문에 기존의 회복기법을 블록 단위의 트랜잭션으로 적용이 가능하다.

4. 성능분석

4.1 평가환경

실험 평가에 사용된 시스템 환경은 CPU가 Pentium 4 3.0 GHz이고 메모리는 4GB이다. 실험에 사용된 데이터는 반도체 생산 공정 데이터 처리 솔루션을 개발하는 ㈜비스텔에서의 샘플 데이터를 가지고, 초당 10만 건의 데이터를 인위로 생성하는 프로세스에 의해 생성되는 데이터이다.

데이터는 기본적으로 데이터의 생성시간을 포함하고 있고, 장비의 ID와 SLOT ID, 그리고 기타 정보들과, 가변길이의 로그데이터(최대 2MB)를 포함하고 있다. 레코드의 스키마는 <TIME, EQPID(INT), SLOT ID(INT), …. , LOG(VARCHAR)>과 같다.

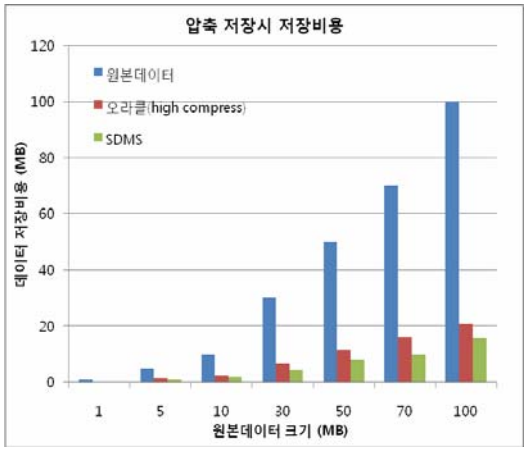
4.2 성능평가

본 논문에서 제안하는 기법의 우수성을 증명하기

위해 상용 디스크기반 DBMS인 Oracle과 MSSQL을 가지고 저장 비용과 저장 속도를 비교한다. SDSM은 저장하는 데이터의 압축이 가능하고 또한 시간 필드에 대해 항상인덱스를 생성한다. 이와 같은 SDSM의 특징을 반영한 삽입과 검색의 성능 비교를 위해 상용 DBMS의 인덱스의 생성 여부에 따른 삽입과 검색의 성능을 비교한다. 또한 SDSM가 압축을 사용하거나 사용하지 않을 때의 삽입 속도를 비교한다. 상용 DBMS의 삽입 속도 측정 시에는 SDSM과 유사한 저장 방식을 사용하기 위해 대량삽입(Bulk Insert) 기법을 사용한다. 대량삽입으로 비교하는 이유는 SDSM은 튜플 단위의 트랜잭션이 아닌 블록 단위로 대량 삽입연산을 수행하기 때문이다.

4.2.1 저장 비용

SDSM은 저장 시 압축기법을 사용하기 때문에 저장비용을크게 감소 시킬 수 있다. 이와 같은 압축 저장 시 감소되는 저장비용을 측정하기 위해 데이터 저장 후 저장된 데이터의 크기를 비교하였다. 저장하는 데이터의 크기는 1MB ~ 100MB로 변화시키면서 다양하게 저장비용을 측정하였다 오라클은 11g의 압축률을 최대로 높이는 옵션인 압축레벨 HIGH를 사용하여 압축 저장 결과를 비교하였다

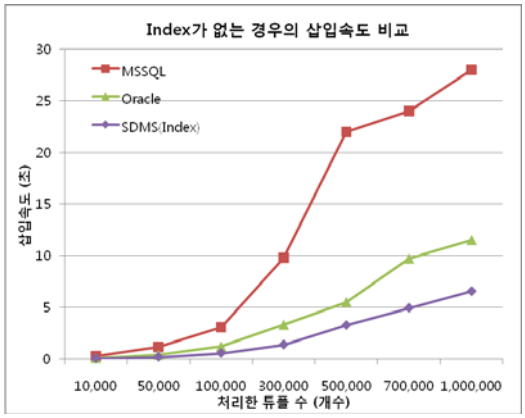


<그림 8> 상용DBMS와 SDSM의 저장비용 분석

<그림 8>은 위와 같은조건에서의 SDSM과 상용 DBMS의 저장비용을 측정한 결과이다. 오라클의 압축 데이터의 크기는 원본 데이터에 비해 약 75~80% 정도의 압축률을 보이고, 제안 기법은 약 85~90% 정도의 압축률로 다소 우수하다. 특히 입력되는 대부분의 데이터가 알파벳 문자열로 구성되어 있으므로 제안기법의 압축을 위한 데이터의 분포도가 낮게 되어 압축효율이 매우 높아 질 수 있다.

4.2.2 저장 속도

첫 번째 실험으로 상용 DBMS는 인덱스를 생성하지 않고 SDSM는 인덱스를 생성할 때의 삽입 속도의 비교이다. 삽입 튜플이 10,000개 에서 1,000,000개까지 다양하게 처리되는 경우의 속도를 비교하였다

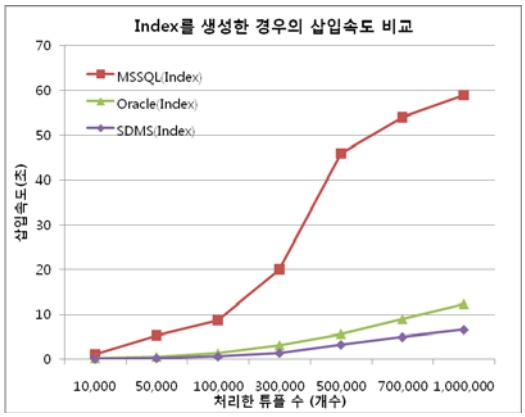


<그림 9> 인덱스를 생성하지 않은 경우 저장시간 분석

<그림 9>의 결과를 통해 SDSM의 삽입속도가 가장 우수함을 볼 수 있다. 측정된 삽입속도는 대량 삽입을 이용하는 MSSQL보다는 크게 향상 되었고 오라클 보다는 소폭 향상되었다. SDSM은 블록 단위로 데이터를 묶어 한번에 대량 삽입하기 때문에 트랜잭션 단위의 데이터 삽입에 필요한 일부 작업이 줄어들어 삽입속도가 다소 향상된다. 또한 SDSM은 데이터 압축을 위한 시간이 필요하지만, 압축을 하면 디스크 IO가 감소하기 때문에 전반적

으로 저장속도를 크게 감소시키지는 않는다.

그 다음 실험은 상용 DBMS가 인덱스를 생성하는 경우에 SDSM와의 삽입 속도의 비교이다. 위와 동일하게 삽입 트랜잭션 요청이 10,000개 에서 1,000,000개까지 다양하게 입력한 경우 처리되는 속도를 비교하였다



<그림 10> 인덱스를 생성하는 경우 저장시간 분석

<그림 10>의 결과를 통해 상용DBMS가 인덱스를 생성하면 제안기법(SDSM)의 삽입속도가 <그림 9>의 실험의 결과 보다 더욱 우수함을 볼 수 있다. 상용 DBMS는 인덱스로 B+ Tree를 생성하는데, B+ 트리는 대량적재 기법을 사용해도 트리의 노드 분할및 재구성의 시간이 필요하기 때문에 삽입속도가 다소 감소한다. 그러므로 이 경우 제안기법의 삽입 속도가 더욱 우수하게 된다

5. 결론 및 향후 연구

본 논문에서는 반도체 생산 장비에서 발생하는 대용량을 데이터를 저장공간을 줄이고 빠른 속도로 디스크에 기록하는 저장관리기 SDSM을 제안하였다. SDSM는 기존의 저장관리자들의 고속처리의 단점인 튜플 단위의 트랜잭션 기법을 사용하지 않고 블록단위의 트랜잭션기법을 사용하였다. 그래서 디스크 반영 시 압축을하여 로그 기록 수와 디스크 IO를 감소시켜 저장 속도를 향상시켰다. 또한 데이터를 그대로 디스크에 반영하면 디스크 공간이 매우 많이 소모되므로 데이터를 압축하여 저장을 하였고, 압축 시 시간 기반 인덱스를 생성하여 검색 성능이 저하되지 않도록 구성하였다.

향후 연구로는 압축기법의 성능을향상시키는 것이다. 현재 일반적인 압축기법은 압축효율은 보장하지만 압축속도가 느려 초당 10만 건 이상의 데이터 처리는 매우 어렵다. 또한 압축 시 튜플들의 정보가 유실되기 때문에 압축된 블록을 모두 해제해야 검색이 가능한 단점이 있다. 그러므로 압축 속도를 보다 향상시키고, 압축된 블록에서 즉시 검색이 가능한 기법에 대한 연구가 필요하다. 그 밖에 시스템의 안정성과속도를 더욱 향상시키기 위해 일반적인 DBMS에서 사용하는 동시성 제어 기법 및 회복 기법을 제안된 저장관리자 구조에 적합하도록 확장하는 연구가 필요하다.

참고문헌

1. 반도체장비기술교육센터(SETEC) 기술교육 시리즈(1-1) 반도체 장비 간 통신 표준, 아이티이노베이션, 김원태
2. 알티베이스, 반도체 생산 관리 솔루션 EES , 하이브리드 DBMS 성공사례, http://www.altibase.com
3. 박상근, 박순영, “GMS: 공간 데이터베이스 관리 시스템.” 공동춘계학술대회, 2003
4. Whang, K.-Y., Krishnamurthy, R., “Query Optimization in a Memory-Resident Domain Relational Calculus Database System. ACMTransactions on Database Systems.” March 1990
5. Boncz, P., Manegold, S., Kersten, M., “Database architecture optimized for the new bottleneck: memory access.” VLDB, 1999, pp. 54-65
8. Olston, C., Rosenstein, J. and Varma. R., “Query Processing, Resource Management, and Approximation in a Data Stream

Management System." CIDR, 2003 Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G.,

9. Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J., "Models and Issues in Data Stream Systems." PODS, 2002
10. Abadi, D. J, Carney, D., Centintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S., "Aurora: A New Model and Architecture for Data Stream Management." VLDB Journal, 2003.
11. Chandrasekharan S., and J. Franklin, M., "Streaming queries over streaming data." VLDB, 2002, pp. 203-214
12. Mohan, C., Haderle, DON., "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging." TODS. ACM. Vol.17 No. 1, 1992, pp. 94-162
13. Lory. D., Krithi, R., "Recovery Protocals for Shared Memory Database Systems." SIGMOD, 1995
14. Jochen van den B., Bernhard, S., "An Evaluation of Generic Bulk Loading techniques." VLDB, 2001
15. Arge, L., Hinrichs, K., et al. "Efficient Bulk Operations on Dynamic R-trees." ALENEX. 1999, pp. 328 -348

백성하
2005년 인하대학교 수학통계학부(이학사)
2007년 인하대학교 컴퓨터 정보공학과(공학석사)
2007~현재 인하대학교 대학원 컴퓨터 정보공학과 (박사과정)
관심분야 : 데이터 스트림 관리 시스템, 데이터베이스 클러스터, 위치기반서비스
e-mail : shbaek@dblab.inha.ac.kr

이동욱
2003년 상지대학교 전자계산공학과(공학사)
2005년 인하대학교 컴퓨터 정보공학과(공학석사)
2005~현재 인하대학교 대학원 컴퓨터 정보공학과 (박사과정)
관심분야 : 공간 데이터 웨어하우스, 공간 정보 관리, 유비쿼터스 환경을 위한 SDBMS
e-mail : dwlee@dblab.inha.ac.kr

어상훈
2003년 인하대학교 컴퓨터 정보공학과(공학사)
2003~현재 인하대학교 대학원 컴퓨터 정보공학과 (통합과정)
관심분야 : 공간DBMS, 센서 네트워크
e-mail : eosanghun@dblab.inha.ac.kr

정원일
1998년 인하대학교 전자계산공학과(공학사)
2004년 인하대학교 컴퓨터 정보공학과(공학박사)
2004~2006년 한국전자통신연구원 선임연구원
2007~현재 호서대학교 정보보호학과 전임강사
관심분야 : 데이터베이스, 데이터스트림, 이동객체, 시스템 보안
e-mail : wnchung@hoseo.edu

김경배
1992년 인하대학교 전자계산공학과(공학사)
1994년 인하대학교 대학원 전자계산공학과(공학 석사)
2000년 인하대학교 대학원 전자계산공학과(공학 박사)
2000년 ~ 2004년 학국전자통신 선임연구원
2004년 ~ 현재 서원대학교 컴퓨터교육학과 조교수
관심분야 : 이동실시간 데이터베이스, 스토리지 시스템, GIS, VOD
e-mail : gbkim@seowon.ac.kr

오영환
1993년 인하대학교 전자계산공학과(공학사)
1997년 인하대학교 전자계산공학과(공학석사)
2001년 인하대학교 전자계산공학과(공학박사)
2000~2001년 (주)케이지아이 시스템 개발부 부장
2002년~현재 나사렛대학교 컴퓨터공학부 조교수
관심분야 : 공간 데이터베이스, 데이터베이스 시스템의 보안, GIS
e-mail : yhoh@kornu.ac.kr

배해영
1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형 GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2004년~2006년 인하대학교 정보통신대학원 원장
2006년~현재 인하대학교 대학원장
관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등
e-mail : hybae@inha.ac.kr