ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

# 컬럼-지향 데이터베이스를 위한 컬럼-인지 트랜잭션 관리 기법☆

# Column-aware Transaction Management Scheme for Column-Oriented Databases

변 시 우[1*]
Si-woo Byun

**요 약**

컬럼-지향 데이터베이스 저장소는 우수한 입출력 성능으로 대용량 데이터 분석 시스템을 위한 매우 진보적인 모델이다. 전통적인 데이터 저장소는 빠른 쓰기 연산을 위하여 한 레코드의 속성들을 하드디스크에 연속적으로 배치되어 있는 가로-지향 저장 모델을 활용하였다. 하지만 검색이 대부분인 데이터웨어하우스 시스템을 위해서는 월등한 판독 성능 때문에 컬럼-지향 저장소가 더 적합한 모델이 되고 있다. 또한 최근에는 플래시 메모리를 사용한 SSD가 고속 데이터 분석 시스템을 위한 적합한 저장 매체로 인식되고 있다. 이제 플래시 메모리는 비휘발성, 낮은 전력소모, 빠른 데이터 접근 속도 등의 특징으로 최신 데이터베이스 서버의 핵심 저장 요소로 충분한 기반이 되었다. 하지만 컬럼 압축의 느린 특성과 일반 RAM 메모리에 비하여 상대적으로 느린 플래시 메모리 연산 특성을 고려하여 기존의 트랜잭션 처리 기법을 개선할 필요가 있다. 본 연구에서는 효율적인 트랜잭션 처리를 위하여 컬럼-인지 다중 버전 로킹(CaMVL) 기법을 제안한다. CaMVL은 로크 관리 과정에서 플래시의 느린 쓰기 연산과 지우기 연산을 효과적으로 제어하기 위하여 멀티 버전 읽기를 허용하고 압축 로크를 허용하여 트랜잭션 처리 성능을 높인다. 또한 성능 검증을 위하여 시뮬레이션 모델을 제안하였으며 실험 결과 분석을 통하여 CaMVL이 기존의 트랜잭션 처리 기법보다 우수함을 확인하였다.

☞ 주제어 : 컬럼-지향 데이터베이스, 플래시 메모리, 데이터웨어하우스, 컬럼-인지 잠금, 시뮬레이션

**ABSTRACT**

The column-oriented database storage is a very advanced model for large-volume data analysis systems because of its superior I/O performance. Traditional data storages exploit row-oriented storage where the attributes of a record are placed contiguously in hard disk for fast write operations. However, for search-mostly datawarehouse systems, column-oriented storage has become a more proper model because of its superior read performance. Recently, solid state drive using MLC flash memory is largely recognized as the preferred storage media for high-speed data analysis systems. The features of non-volatility, low power consumption, and fast access time for read operations are sufficient grounds to support flash memory as major storage components of modern database servers. However, we need to improve traditional transaction management scheme due to the relatively slow characteristics of column compression and flash operation as compared to RAM memory. In this research, we propose a new scheme called Column-aware Multi-Version Locking (CaMVL) scheme for efficient transaction processing. CaMVL improves transaction performance by using compression lock and multi version reads for efficiently handling slow flash write/erase operation in lock management process. We also propose a simulation model to show the performance of CaMVL. Based on the results of the performance evaluation, we conclude that CaMVL scheme outperforms the traditional scheme.

☞ keyword : column-oriented database, flash memory, datawarehouse, column-aware locking, simulation

## 1. 서 론

데이터웨어하우스 분야에서 빠르게 부상하고 있는 컬럼-지향(Column-Oriented) 데이터베이스[1,2]는 기존의 한 레코드 단위로 연속 저장하는 가로방향-저장 데이터베이스와는 상반되어 세로의 필드 단위로 분리 저장 후 검색하는 새로운 데이터베이스이다. 즉 컬럼-지향 데이터베이스는 세로로 저장하므로 유사성이 높은 데이터들이 서로 군집되어 압축, 저장, 검색에 매우 효과적인 구조이다.

특히 컬럼-지향 데이터베이스는 읽기가 대부분인 고객 관리, 전자 라이브러리, 전자 카탈로그용 데이터베이스 분야에도 빠르게 확산중이다. 이러한 판독 위주의 데이터

[1] Dept. of Digital Media, Anyang University, Anyang, 430-714, Korea
* Corresponding author (swbyun@anyang.ac.kr)
[Received 06 January 2014, Reviewed 10 January 2014, Accepted 02 April 2014]
☆ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행되었습니다(No.20120002767)

베이스 환경에서는 특정 컬럼에 데이터가 모여 군집되어 있는 컬럼 저장 구조가 당연히 효율적이며 C-Store[3,4]가 잘 알려져 있다.

전통적으로 하드디스크가 많이 사용되었으나 최근에는 기업의 중대형 데이터 서버나 노트북 등 에서 전통적인 하드디스크 저장 시스템에서 고속 플래시 메모리 기반의 SSD(Solid State Drive) 저장 시스템으로의 대체이동이 빠르게 진행되고 있다[5,6].

기본적으로 컬럼-지향 데이터베이스는 테이블 저장시 세로로 분리 저장한다. 회사 테이블을 예로 들어 논리적으로 행과 열의 2차원으로 구성되어 있을 경우 실제 물리적인 저장은 해당 저장 장치에서 연속적인 일차원 바이트 열로 구성되어 운영체제나 버퍼와 연동된다[7].

컬럼-지향 데이터베이스는 세로로 저장하므로 유사성이 높은 데이터들이 서로 군집되어 압축, 저장, 검색에 매우 효과적인 구조이다. 이러한 물리적 구조는 저장, 검색 외에도 파티셔닝, 인덱싱, 캐싱에도 성능상 큰 영향을 준다. 과거에는 하드디스크가 대세였고 스토리지도 이에 적응하였으므로 하드디스크의 접근 특성을 고려하였고 당연히 저장 연산 처리에 가장 효율적인 방식인 가로-지향 저장 방식으로 설계되었다. 그러나 이제 고성능 SSD가 속도와 더불어 가격 경쟁력을 갖춤에 따라서 기존의 하드디스크를 넘어서 차세대 컬럼-지향 데이터베이스의 저장 시스템으로서 SSD을 활용한 신속한 데이터 처리 및 관리 기법이 필요하게 되었다[8,9].

본 논문은 이러한 컬럼-지향 데이터베이스 시스템에서의 고속 대용량 플래시 메모리를 기반으로 데이터 처리 성능을 향상시킬 수 있는 새로운 관리 기법을 제안한다.

## 2. 관련 연구

먼저 본 연구의 제안에 앞서 기초가 되는 새로운 컬럼-지향 데이터베이스의 특성을 일반 데이터베이스와 비교하면 다음과 같다.

(1) 컬럼-지향 데이터베이스는 데이터가 메모리에 올라가면 같은 타입의 데이터가 순차적으로 군집되어 나열되어 있으므로 당연히 높은 압축 효율과 대용량의 고속 압축이 가능하다. 또한 압축후 접근할 데이터의 크기가 대폭 줄어든 효과로서 다량의 집합 연산 수행시 일반 스토리지가 불필요하게 전체 레코드를 읽어야 하는데 비하여 연관된 컬럼만 읽어도 되므로 매우 I/O 효율적이다[7].

또한 컬럼-지향 데이터베이스는 실제 SQL 명령 수행시 압축되거나 분리 저장된 데이터가 통합되어 접근이 용이한 레코드 포맷으로 변환 되어야 하는데 이 변환과정을 실체화[10]라고 한다. 또한 컬럼별로 분리되어 저장된 데이터를 같은 줄 단위로 서로 연결하여 일반 데이터베이스의 한 투플처럼 수평으로 실로 꿰매듯이 연결하는 작업이므로 “tuple stitching”이라고도 한다. 실체화를 수행하는 이유는 기존 범용 데이터베이스와 같은 쿼리 인터페이스 및 범용 접속 드라이버를 제공하기 위함이며 일반적인 가로-지향 데이터베이스에는 없는 기능이다. 실체화는 변환 시간이 많이 소요되므로 실체화의 효율과 성능이 좋아야 한다.

(2) 컬럼-지향 데이터베이스는 다량의 레코드에 한 컬럼의 수치를 수정 반영할 때에 더 효율적이다. 이런 업데이트 연산은 일상적으로 자주 발생하는데 일반 데이터베이스는 업데이트와 관련 없는 다른 컬럼들도 포함하여 스토리지로부터 읽고 메모리상에도 할당시키느라 많은 시간과 공간을 소모한다. 반면에 컬럼-지향 데이터베이스는 연산과 무관한 컬럼은 아예 터치하지 않고 관련된 수정연산만 수행하므로 공간 효율적이며 속도도 더 빠르다.

(3) 컬럼-지향 데이터베이스는 컬럼별로 반복적 파이프라인을 적용할 수 있는데 이는 캐시용량이 크고 하이퍼쓰레딩이나 멀티코어 CPU와 같은 최신 컴퓨터 환경에 잘 부합되므로 병렬처리 효과를 극대화시킬 수 있다.

(4) 컬럼-지향 데이터베이스의 또 다른 중요한 특징은 CPU 및 저장 부하의 변동성이다. 특히, 일반 데이터베이스에 비하여 컬럼-지향 데이터베이스의 데이터웨어하우스 환경은 소수의 인원이 집중적인 피크 로드를 발생했다가 유휴 시는 CPU 로드가 급격히 감소하므로 변동성이 훨씬 더 크다. 또한 컬럼-지향 데이터베이스의 특성상 CPU 기반의 대용량 분석 작업과 RAM과의 CPU간의 데이터 처리 등으로 고부하 작업시 CPU는 압축연산에 필요한 CPU 자원 제공이 어려워질 수도 있다. 이러한 CPU 부하 특성을 고려하면 효과적인 시스템을 설계할 수 있다.

(5) 업무 활용 분야 측면에서 정리해 보면, 쓰기 트랜잭션이 매우 빈번히 발생하는 OLTP(OnLine Transaction Processing)형 분야에서는 기존의 가로-지향형인 일반 데이터베이스가 더 적합하다. 반면 대용량 데이터의 복잡한 쿼리가 신속히 수행되어야 하는 데이터웨어하우스와 같은 OLAP(OnLine Analytical Processing)형 분야에서는 컬럼-지향 데이터베이스가 더 우수한 성능을 제공한다. 즉 일반 데이터베이스의 장점은 한 레코드의 많은 컬럼이 수정되는 경우나 새로운 데이터가 빈번히 추가되는 경우에 효과적이다.

지금까지 사용되어 온 하드디스크는 대용량 저장의 장점은 있으나 입출력 성능이 시스템 속도에 비하여 느렸다. 최근 대응책으로 SSD를 사용하는 추세인데 쓰기 횟수가 제한되는 고유한 특성이 있다. 또한 쓰기 연산 이전에 소거 연산을 수행하여야 하는 독특한 구조이다[5]. 컬럼-지향 데이터베이스도 시스템 운영을 위하여 반드시 저장 장치가 필요하며 전통적인 하드디스크를 그대로 사용할 수도 있다. 하지만 SSD도 이제 충분한 가격경쟁력을 갖추고 있고 빠른 추세로 대용량화 되고 있으며 속도가 빠르므로 고속 읽기용으로 더욱 적합하다. 따라서 본 연구에서처럼 새로운 저장기술인 플래시 메모리를 최신 컬럼-지향 데이터베이스에 접목시키면 더 효율적이다.

또한 고속의 판독작업이 많은 컬럼-지향 데이터베이스의 읽기 성능의 향상을 위하여 다중 버전 기법[11,12]을 활용하면 효과적이다. 이 방법은 한 데이터 항목에 대하여 여러 개의 버전을 유지하고 한 트랜잭션이 그 항목에 대하여 접근을 요구할 때 현재 수행중인 스케줄의 직렬가능성을 유지하기 위해 적절한 버전을 선택되도록 한다.

다중 버전 기법은 원래 다른 기법에서는 거부될 일부의 읽기 연산들이 이전 버전을 읽도록 허용하여 접근의 효율성을 높이는 반면에 여러 버전을 유지하기 위한 좀 더 많은 디스크 공간이 필요한 단점이 있다. 이러한 데이터 접근 기법을 플래시 메모리 기반 데이터베이스에 적용하면 느린 쓰기 연산으로 인한 접근 효율의 저하를 방지하고 읽기 연산의 접근 속도를 높일 수 있다. 또한 컬럼-지향 데이터베이스의 특성상 발생하는 다량의 압축 저장에 의한 시간 지연 시에도 이전 버전을 사용할 수 있으므로 그 효과는 더욱 배가된다.

## 3. 본 트랜잭션 처리 기법의 제안

### 3.1 제안 배경

컬럼-지향 데이터베이스는 컬럼 단위의 고효율 압축 특성을 이용하여 읽기 속도가 매우 빠르며 플래시 메모리를 활용하는 경우 그 효과는 배가된다. 하지만 일반 데이터베이스와는 달리 쓰기 과정 전의 압축 연산에 매우 많은 시간이 소요되며 플래시 메모리를 활용하는 경우 쓰기 연산에 앞서 저속도의 지우기 연산이 먼저 수행되어야 하는 플래시 메모리의 특성상의 단점이 있다. 이러한 컬럼-지향 데이터베이스의 압축 특성과 플래시 메모리의 고유한 단점을 고려하여 기존의 트랜잭션 처리 기법을 개선하지 않으면 플래시 메모리를 주 저장장치로 사용하는 컬럼-지향 데이터베이스에서 충분한 성능을 기대하기는 어렵다.

이러한 배경에서 본 연구에서 플래시 메모리기반의 컬럼-인지 트랜잭션 관리 기법을 제안하며 컬럼-지향 데이터베이스의 성능 개선과 더불어 컬럼-데이터베이스에 접근하는 트랜잭션의 직렬가능성을 보장하고자 한다.

### 3.2 컬럼-인지 다중 버전 잠금 기법의 제안: CaMVL

일반적으로 트랜잭션 처리와 관련된 동시성 제어 기법에서 데이터 일관성 유지를 위하여 직렬가능성(Serializability)을 보장하게 된다. 보통 트랜잭션이 입출력을 위하여 기다리는 동안 중앙처리장치는 대기상태로 되므로 이용률이 저하된다. 이때 일관성을 유지하면서 다른 트랜잭션을 동시에 수행하게 되면 처리 성능은 자연히 높아진다. 즉 트랜잭션의 직렬화가 가능하게 되면 정확성과 동시 실행성의 장점을 모두 얻을 수 있다. 직렬 가능성을 보장하는 보편적인 방법이 로킹 기법이며 동시에 실행되는 트랜잭션의 상호 간섭을 방지하기 위해 데이터 항목에 록(lock)을 걸게 된다. 현실적으로 데이터베이스 관리 시스템을 구현할 때 여러 가지 2단계 로킹의 변형 기법들 중에서 많이 사용되는 것이 엄격한 2단계 로킹(strict two-phase locking: S2PL)기법[11,12]이다.

표 1과 같이 S2PL은 간단히 읽기 로크와 쓰기 로크를 가지며 읽기 로크간에만 호환성이 유지된다. 또한 S2PL은 트랜잭션 T가 완료되거나 철회될 때까지 T가 보유한 로크들 중 어떠한 로크도 해제하지 않는다. 따라서 S2PL은 트랜잭션 T가 완료되지 않았으면 어떠한 다른 트랜잭션도 T가 쓴 항목을 읽거나 쓸 수 없다. 반면 S2PL과 비교되는 보수적 2PL의 경우에는 트랜잭션이 수행을 시작하기 전에 필요로 하는 모든 로크에 대하여 로크를 획득한다는 차이가 있다.

(표 1) S2PL의 록 호환성 테이블
(Table 1) Lock Compatibility Table of S2PL

| 록 모드 | 읽기(R) | 쓰기(W) |
|---|---|---|
| 읽기(R) | Y | N |
| 쓰기(W) | N | N |

이러한 S2PL 기법은 플래시 메모리 데이터베이스 환경에 적용할 경우 직렬가능성 조건을 만족한다. 그러나 성능 향상을 위해서는 컬럼-지향 데이터베이스의 압축

부하와 플래시 메모리 특성을 고려하여 더 개선할 필요가 있다.

먼저 컬럼-지향 데이터베이스 관점에서 살펴보면 읽기 트랜잭션의 경우에는 이미 압축 저장된 데이터를 복원하는 속도가 빠르므로 별 문제가 없다. 하지만 새롭게 압축하여 쓰기 연산의 경우에는 압축 전략과 판단 과정 압축인코딩 과정으로 인하여 속도가 상대적으로 매우 느리다. 즉 플래시 메모리에 저장 전에 반드시 수행해야 하므로 압축 연산의 부담이 상당히 큰 것이 문제이다.

다음으로 플래시 메모리 관점에서 살펴보면 역시 읽기 트랜잭션의 경우에는 플래시 메모리의 연산 속도가 일반 RAM 메모리에 비하여 크게 뒤지지 않으므로 별 문제가 없다. 하지만 플래시 메모리의 쓰기 연산의 경우에는 속도가 상대적으로 매우 느리며 더욱이 쓰기 연산 전에 상당히 느린 소거 연산을 반드시 수행해야 하므로 연산의 부담이 상당히 큰 것이 문제이다. 즉 플래시 메모리에 쓰기 트랜잭션을 수행시킬 경우 실제 특정 번지에 쓰기 연산을 수행하기 전에 매우 느린 속도로 세그먼트 단위의 소거 연산이 먼저 수행되어야한다. 또한 소거 연산 수행 후의 쓰기 연산도 한 바이트 단위가 아닌 더 큰 블록 단위로 수행해야 하는 부담도 있다.

따라서 쓰기 연산의 비중이 높을수록 빈번한 트랜잭션 충돌이 심각하게 발생하여 트랜잭션 처리 성능을 크게 저하시키게 된다. 이 때 접근하고자 하는 자원을 미리 점유하지 않고 바로 쓰기 록을 진행하면 높은 충돌 특성에 의하여 다수의 트랜잭션이 자원의 점유 과정에서 중도에 철회되게 된다.

본 제안 기법은 쓰기 연산의 록을 압축 록과 쓰기 록으로 세부 분리하여 쓰기 연산 진행시 미리 압축 록을 획득하여 진행하게 제어하였다. 즉 이 기법은 이 두 가지의 록을 효율적으로 관리하여서 가장 심각한 쓰기 트랜잭션 관련 철회 율을 개선하게 된다. 또한 쓰기 트랜잭션은 장시간 록을 점유하기 때문에 이 점유 시간에 발생되는 다수의 읽기 트랜잭션은 모두 대기하거나 철회되어야 한다. 이로 인한 트랜잭션 처리 성능의 저하도 심각하므로 제안 기법은 읽기 트랜잭션에 한하여 다중 버전을 허용하였다.

다중 버전 기법은 읽기 로크, 쓰기 로크, 보증 로크의 3가지의 로크가 있다. 그러므로 데이터 항목의 상태는 여기에 로크 해제가 포함되어 4가지의 중 하나가 된다. 다중 버전 로킹 기법에서는 여러 읽기 연산들과 하나의 쓰기 연산이 동시에 수행될 수도 있다. 이에 비하여 표준적인 S2PL 기법에서는 원천적으로 로크 호환이 차단되어서 수행이 불가능한 단점이 있다.

본 연구에서는 컬럼-인지 다중버전 잠금 기법(Column-aware Multi-Version Locking: CaMVL)을 제안하는데 기초가 되는 다중 버전 로킹 기법[11]을 기반으로 확장한 기법이다. CaMVL 기법은 표 2와 같이 읽기 록(Read Lock), 압축 록(Compress Lock), 쓰기 록(Write Lock), 인증 록(Verify Lock)의 4가지 록 모드가 있으며 상호간의 호환성을 나타내고 있다. 일반적인 록 관리 기법에서는 한 트랜잭션이 특정 오브젝트에 대하여 쓰기 록을 점유하게 되면 추후에 발생되는 읽기 및 쓰기 트랜잭션은 이 특정 오브젝트에 대한 연산을 수행할 수 없다. CaMVL의 다중버전 허용 록 관리는 어떤 트랜잭션이 특정 오브젝트에 대하여 쓰기 록을 걸고 있을 때 이 오브젝트에 접근하고자 하는 다른 읽기 트랜잭션을 허용하는 것이다. 이 CaMVL 기법은 한 오브젝트에 대하여 두 가지 버전을 관리하여 제어한다. 한 버전 오브젝트는 완료된 트랜잭션이 쓰기를 수행한 것이고 다른 하나의 버전 오브젝트는 어떤 트랜잭션이 그 오브젝트에 대해 쓰기 록을 점유할 때 만들어진다. 따라서 어떤 트랜잭션이 쓰기 록을 점유하고 있어도 추후 발생된 읽기 트랜잭션은 완료된 버전의 오브젝트를 계속 읽을 수 있다. 쓰기 트랜잭션은 완료된 버전의 값에 영향을 주지 않고 원하는 대로 오브젝트의 값을 갱신할 수 있다. 그러나 일단 트랜잭션이 완료할 준비가 되면 완료하기 전에 현재 쓰기 록을 보유하고 있는 모든 항목들에 대하여 인증 록을 확보해야 한다. 인증 록은 읽기 록과 호환성이 없으므로 자신이 쓰기 록을 가지고 있는 오브젝트들을 읽고 있는 다른 트랜잭션이 없을 때까지 트랜잭션 완료가 연기된다.

(표 2) CaMVL의 록 호환성 테이블
(Table 2) Lock Compatibility Table of CaMVL

| 록 모드 | 읽기(R) | 압축(C) | 쓰기(W) | 인증(V) |
|---|---|---|---|---|
| 읽기(R) | Y | Y | Y | N |
| 압축(C) | Y | N | N | N |
| 쓰기(W) | Y | N | N | N |
| 인증(V) | N | N | N | N |

CaMVL 록 관리자는 트랜잭션 관리자와 데이터 관리자와 메시지 교환을 통하여 록 관리를 수행한다. 트랜잭션 관리자로부터 요청 메시지가 오면 이를 분석한 후 록 호환성 테이블에 의거하여 적절히 록을 설정한다. 그리고 데이터 관리자를 호출하여 읽기 연산, 압축 연산, 쓰기 연산, 완료 연산 등을 수행하는데 읽기 연산의 경우는 최근

에 완료된 데이터를 판독하게 되며 실제 쓰기 연산은 압축 록과 쓰기 록을 확보한 후 실제 플래시 메모리에 쓰기가 수행되며 이때 새 버전이 생성되게 된다. 쓰기 연산이 완료되면 인증 록으로 전환되어 읽기 록이 모두 해제될 때까지 기다린 후 종료되게 된다. 데이터 관리자로부터 완료나 철회 메시지를 받게 되면 CaMVL 록 관리자는 해당 트랜잭션에 설정된 록을 모두 해제하고 대기하는 다른 트랜잭션을 꺼내어 수행에 필요한 관련 록들을 설정한 후 다시 데이터 관리자에 연산 수행을 요청하게 된다.

간략한 CaMVL 기법의 록 관리 알고리즘은 다음과 같다. 먼저 트랜잭션 관리자로부터 데이터베이스 연산에 대한 요청을 받게 되면 해당 연산의 트랜잭션 아이디와 연산의 종류 등을 파악한다. 연산의 종류가 읽기 또는 쓰기 연산이면 현재의 데이터 항목에 록을 걸고 있는 모든 록 유닛을 검색한다. 만일 록이 전혀 설정되어 있지 않거나 록이 호환가능하다면 해당 록을 설정한 후 데이터 관리자에게 해당 연산 실행 메시지를 보낸다. 연산의 종류가 트랜잭션 완료이면 쓰기 록이 인증 록으로 전환된다.

다음으로 데이터 관리자로부터 연산 수행의 결과 메시지를 받게 되면 해당 연산의 수행 결과와 연산의 종류 등을 파악한다. 연산의 종류가 트랜잭션의 완료나 철회이면 트랜잭션을 종료하면서 대기하고 있는 다른 트랜잭션을 활성화 시켜주어야 한다. 즉 자신의 록을 풀어준 후 대기큐에서 맨 앞에 위치한 트랜잭션을 선택하여 이 트랜잭션과 호환가능한 모든 트랜잭션들을 검색한다. 검색된 트랜잭션 집합에 대하여 각각 해당 록을 설정하고 다시 데이터 관리자에게 실행 요청을 함으로써 계속해서 다른 트랜잭션들이 수행되도록 한다. 이 과정을 의사 코드로 표현된 록 관리 알고리즘은 아래와 같다.

**알고리즘:: CaMVL록관리자**

```
Input: Message msg; /* 트랜잭션 메니져로부터 전송된 메시지 */
do {
  wait(msg); /* 새로운 메시지를 기다린 후 도착하면 진행. */
  switch (msg.type) { /* 도착한 메시지를 유형별로 분류 */
    case DBOP:      /* 형식이 데이터베이스 오퍼레이션임 */
      Op = msg.operation; x = msg.data;T = msg.transactionId;
      switch (Op) {
        case T_BEGIN: /* 시작되면 데이터 관리자에게 보냄 */
          send operation to flash data manager(FDM);
          break;
        case T_READ or T_WRITE:
          find the lock unit lu such that x ⊆ lu;
                  /* 데이터 x와 관련된 록들을 찾음 */
          /* 현재 록이 풀려 있거나 모드가 서로 호환가능하다면 */
          if ((lock mode of lu is compatible ) or (no lock held)) {
                  /* 록 호환 테이블을 참조하여 적합한 록을 설정함 */
            set lock on lu in appropriate mode;
            /* READ인 경우는 R-Lock을 설정하고, WRITE인 경우는
               C-Lock을 설정하고 압축후 W-Lock을 설정함*/
            send operation to flash data manager;
            /* 플래시 데이터 관리자를 호출하면, READ인 경우는
               최근 커밋된 버전을 판독하고, WRITE인 경우는 쓰면서
               새 버전이 생성되게 됨 */
          } else
              wait in queue;
          break;
        case T_ABORT: send operation to flash data manager; break;
              /* 트랜잭션이 철회되는 경우 */
        case T_COMMIT: convert W-Lock to V-Lock; break;
              /* 설정된 록이 W-Lock 인 경우 V-Lock으로 전환됨. */
      } /* end_switch */
      break;
    case FDMRM: /* 형식이 데이터 관리자의 리턴 메시지임. */
      Op = msg.operation; res = msg.result; T = msg.transactionId;
      if (Op==T_ABORT or Op==T_COMMIT) { /* 종료가 되면 */
        for each lock unit lu locked by T do {
            /* 그 동안 사용했던 록을 각각 풀어주고 */
          release lock on lu held by T;
          SOP = first operation from the queue;
            /* 작업큐에서 대기중인 오퍼레이션을 빼내어 */
          SOP = SOP ∪ {O|O is an operation on queue that can lock
                        lu in compatible mode}
          set the locks on lu; /* 그 오퍼레이션이 필요로 하는 록
                        이 호환가능하다면 록을 허가해 준 후 */
          for each operation in SOP do /* 플래시 데이터 메니저를
                        통하여 데이터를 읽거나 쓰게 한다. */
            send each operation to flash data manager;
        }
      } /* end if */
      break;
  } /* end_switch */
} While (true); /* 새로운 메시지가 있으면 반복 수행한다. */
```

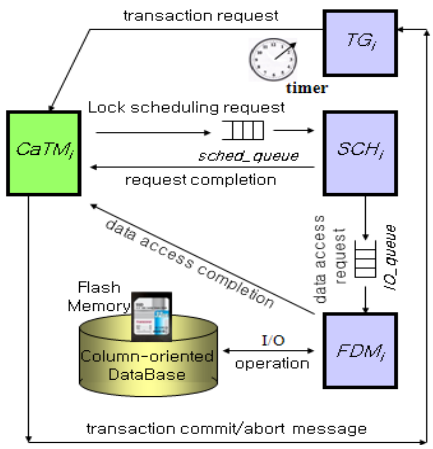
## 4. CaMVL 기법의 성능 분석

본 연구에서 제안된 CaMVL의 성능을 검증하기 위하여 컴퓨터 시뮬레이션을 수행하였으며 실험 결과를 분석해 보았다. 컴퓨터 시뮬레이션 실험에 사용된 비교 기법은 가장 보편적으로 사용하는 S2PL기법이며 실험이 수행된 하드웨어 환경은 인텔 i7 CPU에 메인 메모리 8G, 운영체제는 64bit 윈도우즈7을 사용하였고 개발언어는 비주얼 C++을 사용하였다. 또한 실험용 샘플인 대용량 데이터로서 일억 건 분량의 데이터베이스를 적용하였으며 부수적으로 작업 부하 생성모듈과 통계 자료 수집 및 성능 수치 분석용으로 CSim API[13]가 사용되었다.

### 4.1 시뮬레이션 모델

본 시뮬레이션은 그림1과 같이 CSIM에서 제공되는 폐쇄형 큐잉 모델을 사용하며 트랜잭션 생성 큐, 스케줄링 요청 큐, 입출력 큐, 메시지 큐 등이 사용되었다. 시뮬레이션은 트랜잭션 생성기(Transaction Generator:TG), 컬럼-인지 트랜잭션 관리자(Column-aware Transaction Manager: CaTM), 스케줄러(Scheduler: SCH), 플래시 데이터 관리자(Flash Data Manager:FDM)로 구성되었다.

TG는 트랜잭션을 가상적으로 생성하는 역할을 하며 CaTM은 TG에서 발생된 트랜잭션을 분석하여 스케줄링 요청 대기열로 전달한다. SCH는 대기열로 부터 선입선출 방식으로 꺼내어 알고리즘에 따라서 하나씩 처리한다. 들어온 읽기나 쓰기 연산이 실행가능하면 FDM의 I/O 대기열로 보내고 아니면 대기시킬 것인지 철회시킬 것인지를 결정한다. FDM은 각 연산 객체에 대하여 플래시 메모리 미디어와 관련된 입출력 작업을 수행하고 수행이 종료되면 완료 메시지를 리턴 한다. CaTM은 사용자 트랜잭션의 생성부터 종료까지의 수행을 관리한다. 또한 본 연구의 비교 실험을 위하여 S2PL과 CaMVL 알고리즘이 SCH에서 수행된다.




(그림 1) CaMVL 모의실험 모델
(Figure 1) CaMVL Simulation Model

시뮬레이션의 주요 성능 지표는 컬럼-세그먼트 단위의 트랜잭션 처리치(throughput)와 그 응답시간(response time)이다. 트랜잭션 처리치는 초당 몇 개의 트랜잭션이 처리되었는지를 의미하고 응답시간은 트랜잭션이 발생한 후 수행까지의 지연 시간을 의미한다. 전체 연산수에 대한 쓰기 연산의 비율인 갱신 비율은 기본적으로 7:3이며 시스템에 미치는 영향을 분석하기 위하여 5:5인 균등모드 9:1인 읽기집중모드로 변화시켜 보았다. 위의 설정된 주요 파라미터 외의 부수적인 파라미터의 수치는 고정된 값이지만 실험 진행 과정에 필요시 변화시킬 수 있다.

### 4.2 컬럼 데이터베이스의 구성 및 압축 특성

본 실험에 앞서서 실험용 샘플이 되는 대용량의 데이터가 필요한데 국내 한 기업의 데이터를 활용하여 총 1억건을 적용하였다. 각 컬럼을 압축하기 위하여 LZO 기법을 사용하였는데 컬럼-저장의 군집 특성상 압축률이 매우 좋은 편이다. 특히 종류가 한정된 컬럼 및 불리언 타입의 컬럼이나 NULL 값이 많은 경우 압축이 가장 잘되어 2%로서 거의 다 압축되기도 하였다.

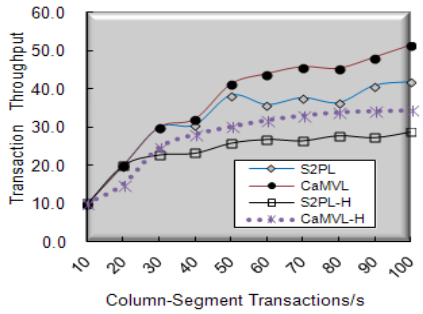
그 다음으로 일련번호, 사용자 번호가 5%대로 압축이 잘되며 날짜와 초단위로 표시되는 시간속성의 경우에는 10%정도로 압축이 되었다. 주소 속성과 같은 불규칙한 문자열이 많이 포함되는 경우는 27%로 압축되었다. 텍스트 메모성의 랜덤한 긴 스트링 데이터가 있는 경우는 압축효율은 더욱더 낮았다. 실제 테스트 해보니 약 53%로 압축되었다. 컬럼 데이터를 압축 저장시 대용량의 컬럼 데이터를 통째로 저장하면 압축 시간도 매우 느리고 복원 시간도 매우 느리게 되므로 적당한 양의 컬럼 세그먼트로 단위로 나누어서 저장하게 하였다.

### 4.3 실험 결과 및 분석

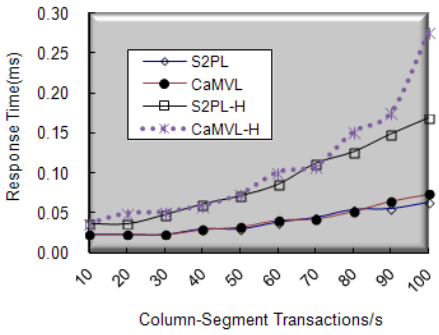
시뮬레이션은 보편적인 방법인 S2PL기법과 제안하는 CaMVL 기법을 대상으로 각 두 모드로 변형하여 총 4가지 방법으로 수행하였으며 성능에 민감한 영향을 주는 초당 트랜잭션 수와 갱신 비율 등의 주요 파라미터를 중심으로 하였다.

(1) 트랜잭션 수의 변화에 따른 성능 비교

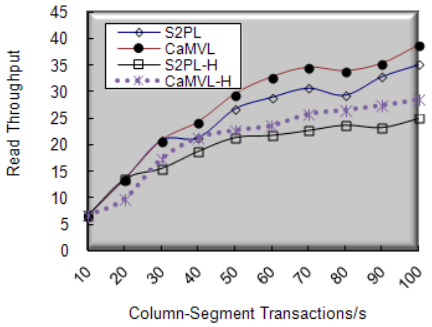
이 실험에서 그림2는 초당 발생된 트랜잭션의 수의 증가에 따른 트랜잭션 처리치를 표시한 그래프이다. 발생된 초당 트랜잭션의 수가 늘어날수록 점차로 트랜잭션 처리 결과치가 증가함을 알 수 있다. CaMVL과 S2PL 표식은 구성된 컬럼들의 평균적인 압축률을 적용한 경우의 성능 추이이며 CaMVL-H와 S2PL-H는 컬럼 중 가장 압축부담이 큰 경우 즉 압축 시간이 많이 걸리는 텍스트 메모성의 랜덤한 긴 스트링 데이터의 경우이다. 실험결과 전구간의 처리 성능을 종합한 결과 CaMVL, S2PL, CaMVL-H, S2PL-H순으로 우수하게 나타났다.




(그림 2) 전체 트랜잭션 처리치
(Figure 2) Throughput of Overall Transaction




(그림 3) 응답 시간
(Figure 3) Response Time




(그림 4) 읽기 트랜잭션의 처리치
(Figure 4) Throughput of Read Transaction

초당 트랜잭션의 수가 대략 50개를 넘으면서 S2PL 기법의 성능이 점점 낮아진다. 즉 이 이상으로 트랜잭션을 활성화시키는 것이 성능 향상에 도움이 되지 않음을 의미한다. 하지만 동일한 조건에서도 제안한 CaMVL 기법이 S2PL 기법에 비하여 성능이 상대적으로 더 높다. 또한 그림3에서 알 수 있듯이 응답시간도 CaMVL 기법이 S2PL 기법에 비하여 더 우수하다. 그 이유는 그림4에서 알 수 있듯이 CaMVL 기법이 S2PL 기법에 비하여 다중 버전 읽기를 허용하여 읽기 트랜잭션 처리치가 더 향상되었기 때문이다. 즉 철회되는 읽기 트랜잭션이 줄고 올드 버전을 판독하여 정상적으로 완료되는 읽기 트랜잭션의 수가 상대적으로 늘어남으로써 향상 효과가 있었다.

좀 더 설명하면 초기의 30이하 구간에서는 대부분 원래 버전으로 읽기 트랜잭션이 처리되지만 시스템의 부하가 30이상으로 증가할수록 다중 버전을 활용한 CaMVL의 읽기 트랜잭션이 S2PL 보다 더 많음을 알 수 있다. 또한 부하가 증가하면서 쓰기 트랜잭션의 수도 증가하게 되는데 이 경우 읽기 트랜잭션은 과도한 쓰기 트랜잭션의 부하를 피하여 CaMVL의 구 버전 읽기를 효과적으로 활용함을 알 수 있다. 이때 S2PL은 쓰기 트랜잭션 발생시 읽기 트랜잭션은 대기하여야만 하므로 처리 성능이 나쁘게 나타난다.

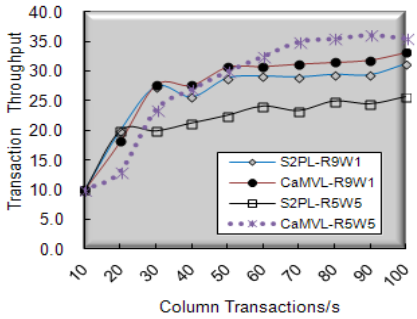
이 다중 버전 활용효과로 인하여 CaMVL은 S2PL에 비하여 약 11% 더 신속한 응답성을 보였으며 약 20% 더 좋은 트랜잭션 처리치를 보였다. 또한 쓰기 연산 전에 컬럼 세그먼트에 미리 압축의도 표시인 Compression 록을 확보하도록 제어하여 압축후 쓰기 연산이 중도에 철회되지 않고 원활하게 진행되도록 유도함으로써 결과적으로 전체적인 트랜잭션 성능 향상을 유도하였다.

(2) 업데이트 비율에 따른 성능 비교

이 실험은 갱신 비율이 위의 두 기법들의 성능에 어떤 영향을 미치는 지를 분석하기 위한 실험이다. 이 영향을 분석하기 위하여 갱신 비율을 50% 및 10%로 설정하였다. 그림5는 해당 갱신 비율에서 발생된 트랜잭션 증가에 따른 각 기법의 트랜잭션 처리 성능을 표시한 그래프이며 그림6은 그 응답 시간을 표시한 그래프이다. 그래프에서 R9W1은 읽기:쓰기 비율이 9:1이란 의미이며 R5W5는 5:5로서 갱신비율이 현실적으로 상당히 높은 경우이다.

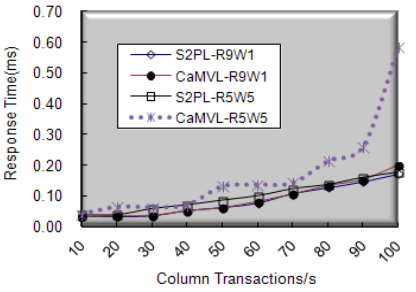
그림5와 그림6에서 보면 두 기법 모두 갱신 비율이 높으면 트랜잭션 처리 성능도 저하되고 응답 속도도 매우 느려짐을 알 수 있다. 이유는 갱신 연산이 많을수록 시간이 많이 소모되는 압축 부담이 커지며 저속의 쓰기 연산의 비율이 점차로 증가하여 읽기-쓰기 및 쓰기-쓰기 트랜잭션간의 충돌이 빈번해면서 시스템의 처리 성능과 응답 속도를 크게 저하시키기 때문이다. 이 그래프를 살펴보면 전반적으로 CaMVL이 S2PL에 비하여 트랜잭션 처리 성능이 우수하게 나타났다. 이는 전술한 다중 버전 허용 전략과 압축 록 관리의 효과 때문이다.

전반적인 갱신 비율 변화 구간에서 평가해보면 R9W1 모드에서 CaMVL은 S2PL에 비하여 6% 더 빠른 응답 성능을 보였으며 7% 더 높은 트랜잭션 처리 성능을 보였다. 반면 R5W5 모드에서 CaMVL은 S2PL에 비하여 54% 더 빠른 응답 성능을 보였으며 39% 더 높은 트랜잭션 처리 성능을 보여 갱신 비율이 높은 경우에 더욱더 효과가 있음을 확인할 수 있었다.




(그림 5) 전체 트랜잭션 처리치

(Figure 5) Throughput of Overall Transaction




(그림 6) 응답 시간

(Figure 6) Response Time

## 5. 결론 및 향후 과제

본 논문에서는 최근 데이터분석용으로 활용되는 컬럼-지향 데이터베이스와 고속 데이터 저장 장치로 많이 사용되는 플래시 메모리와 융합하여 트랜잭션 처리 성능과 응답 성능을 개선할 수 있는 CaMVL 기법을 제안하였다. CaMVL 기법은 컬럼-지향 데이터베이스의 느린 압축 특성을 고려하고 쓰기와 소거 연산이 상대적으로 매우 느린 플래시 메모리 연산의 단점을 고려하여 록 관리 기법에 반영하였다. 또한 다중버전 읽기를 허용하여 트랜잭션이 밀집된 상황에서 구 버전 읽기를 활용하여 전반적인 트랜잭션 철회를 줄였다. 이로 인하여 전체적인 트랜잭션 응답 성능과 처리 성능을 높일 수 있었다.

시뮬레이션 분석 결과 작업 부하가 적은 시작 구간에서는 기존 처리 기법과 비슷하게 나타났지만 트랜잭션이 집중되어 시스템의 부하가 증가하더라도 처리 성능이 저하되지 않고 기존 기법에 비하여 더 우수하게 나타났다. 전체적인 실험 구간에서 CaMVL은 기존 기법에 비하여 11% 더 개선된 응답 성능을 보였으며 20% 더 높은 트랜잭션 처리 성능을 보였다.

향후에는 플래시 메모리 스토리지와 컬럼-지향 데이터베이스의 효율적인 압축 부하 감소 방법을 연구하고자 한다. 또한 컬럼-지향 데이터베이스의 안정성 개선과 장애시 데이터베이스 회복에 대한 연구도 수행할 계획이며 추가적인 실험 데이터 구축한 후 성능평가를 수행할 예정이다.

## 참 고 문 헌(Reference)

[1] S. Ahn, K. Kim. "A Join Technique to Improve the Performance of Star Schema Queries in Column-Oriented Databases," Journal of Korean Institute of Information Scientist and Engineers, Vol. 40, no.3, pp. 209-218, 2013.6.

[2] S. Byun. "Column-aware Polarization Scheme for High-Speed Database Systems," Journal of Korean Society Internet Information, Vol. 13, no.3, pp. 83 - 91, 2012.

[3] D. Abadi, A. Boncz, and S. Harizopoulos, "Column-oriented Database Systems," Proc. of the VLDB, Lyon, France, August 24-28 2009.

[4] S. Harizopoulos, V. Liang, D. J. Abadi, and S. Madden, "Performance tradeoffs in read-optimized databases," Proc. of VLDB, pp. 487 - 498, 2006.

[5] S. Byun, M. Hur, "Flash memory Lock management for portable information systems," International Journal of Cooperative Information Systems, Vol. 15, no. 3, pp. 461-479, Aug. 2006

[6] Lucas Mearian, "Analysis: SSD performance -- is a slowdown inevitable?," Available From: http://www.computerworld.com/s/article/9132668/Analysis_SSD_performance_is_a_slowdown_inevitable_?taxonomyId=19& pageNumber=3, 2013.

[7] D. Abadi, S. Madden, and M. Ferreira. “Integrating compression and execution in column-oriented database systems,” Proc. of SIGMOD, pp. 671 – 682, 2006.
[8] Jeffrey Bausch, ″MLC vs. SLC NAND flash memory,″ http://www2.electronicproducts.com/MLC_vs_SLC_NAND_flash_memory-article-nand_flash_jul2011-html.aspx, 2013.
[9] Samsung, Samsung, “what is NAND Flash based SSD?,“ http://www.samsung.com/global/business/semiconductorproducts/flash/Products_FlashSSD.html, 2013.
[10] D. Abadi, D. Myers, D. DeWitt, and S. Madden. “Materialization strategies in a column-oriented dbms,” MIT CSAIL Technical Report. MIT-CSAIL- TR-2006-078, 2006.
[11] Tamer Ozsu, and Patrick Valduriez, “Principles of Distributed Database Systems,” Springer New York, 2011.
[12] Ramez Elmasri, Shamkant B. Navathe, “Fundamentals of Database Systems,“ Addison Wesley, 2007.
[13] Mesquite, “CSIM2.0 Development Toolkit for Simulation and Modeling,” http://www.Mesquite.com/documentation/documents/CSIM20_User_ Guide-C.pdf, 2013.

## ◐ 저 자 소 개 ◑

**변 시 우**
1989년 연세대학교 전산과학과 졸업(학사)
1991년 한국과학기술원 전산학과 졸업(석사)
1999년 한국과학기술원 정보및통신학과 졸업(박사)
2000년~현재 안양대학교 디지털미디어학과 교수
관심분야 : 데이터베이스, 저장장치, 스마트 임베디드 시스템
E-mail : swbyun@anyang.ac.kr