韓國 컴퓨터情報學會 論文誌
第8卷 第3號, 2003. 9.

2003-8-3-1-6

ROLAP 환경에서 집단함수 질의처리를 위한 효율적인 알고리즘

김 인 식*, 김 종 겸**, 정 순 기***

Efficient Algorithm for Query Processing of Aggregate functions in ROLAP Environment

In-Sik Kim*, Jong-Kyum Kim**, Soon-Key Jung***

요 약

최근 하드웨어 기술의 발달로 다양하고 복잡한 기능들을 가지는 고성능 프로세서들이 일반화되어 사용되고 있다. 데이터베이스 시스템의 질의 처리 과정에서도 이러한 하드웨어적 특성들을 고려한 알고리즘들이 중요한 연구대상으로 부각되고 있다. 최근 연구 결과에 의하면 캐시 미스의 증가로 발생하는 미스 패널티가 메모리와 CPU간에 새로운 병목 현상이 되고 있으며, 분기 오 예측으로 인한 프로세서 자원 손실도 상당한 비중을 갖는다는 것을 보여준다. 본 논문에서는 이러한 하드웨어적 특성들을 효율적으로 사용할 수 있는 기법들에 대한 연구를 통해 질의처리 알고리즘 가운데 집단함수를 최적으로 구현할 수 있는 알고리즘을 제안하였다.

Abstract

The high-performance processors have recently employed sophisticated techniques to overlap and simultaneously execute multiple computation and memory operations. For the query processing of database management systems, those hardware characteristics are the important research issue. The latest works show that the cache miss penalty between main memory and CPU becomes new bottlenecks and the branch misprediction causes serious resource-waste. An efficient algorithm for query processing of aggregate functions considering these hardware characteristics was proposed in this dissertation.

▸ Keywords : 집단함수, 질의처리, 캐시메모리(Cache Memory)

* 경북전문대학 컴퓨터정보과 부교수
** 인천기능대학 자동화시스템과 부교수
*** 충북대학교 컴퓨터공학과 교수

I. 서 론

1. 연구 배경 및 목적

최근 인터넷 기술과 컴퓨터 하드웨어 기술의 발전으로 대용량 데이터베이스로부터 사용자에게 유용한 정보를 선별적으로 제공할 수 있는 데이터 웨어하우스징과 OLAP(On-Line Analytic Processing) 기술이 의사결정지원시스템에서 중요한 연구과제로 등장하고 있다[1].

본 논문에서는 ROLAP(Relational OLAP) 기술에 대한 연구의 일환으로 밀결합(tightly coupled) 다중 프로세서 환경에서 캐시 메모리의 효과적인 운영을 통해 캐시 미스의 최소화와 명령어의 분기예측 최적화를 통해 집단함수 질의를 효율적으로 처리할 수 있는 방법을 제안하고자 한다.

최근 하드웨어 기술의 발전으로 계층적으로 관리할 수 있는 캐시 메모리의 출현, 명령어의 분기를 예측하여 사전에 실행코드를 생성할 수 있는 분기명령의 예측기법, CPU와 메모리 간의 명령어 파이프라이닝 기법 및 TLB (Translation Look-aside Buffer)기법 등이 출현되고 있다[2].

CPU는 일반적으로 명령어의 분기예측 기능을 수행하는 루틴을 가지고 있다[3]. 그러나 분기예측 루틴을 너무 빈번하게 사용하면 오히려 분기 오 예측으로 인해 오버헤드를 초래할 수 있다. 특히, 집단함수는 알고리즘의 특성상 연산의 대상이 되는 테이블 집합들로부터 검색 조건에 맞는 레코드들의 추출 시 매우 빈번한 분기예측 루틴의 사용이 필요하다. 〈그림 1〉은 4종류의 상용 RDBMS에서 질의처리 시 각각에 대한 계산 시간, 캐시 미스, 분기 오 예측 및 자원낭비의 비율을 나타내고 있다[4].

2. 연구 범위 및 내용

RDBMS에서는 질의문 처리 시 실행계획을 수립하여 최적의 실행문을 선택하여 처리한다. 이러한 방법은 하드웨어 특성을 고려하지 않고 단지 질의문에 포함된 연산자들의 처리 순서를 계산 시간과 소요 공간을 고려하여 최적으로 배치하는 것에 불과하다. 본 논문에서는 캐시 메모리의 계층구조를 효율적으로 활용하여 캐시 미스를 최소화 할 수 있는 방법과 명령어의 분기 오 예측 발생 빈도를 줄일 수 있는 방법을 제안하고자 한다.

II. 관련 연구

1. 집단함수 질의처리 과정

대용량의 데이터를 처리하는 ROLAP 환경에서 집단함수 질의처리는 매우 중요하다. 질의 최적기에서 집단함수 연산자의 위치를 적절하게 선택하면 입력 레코드의 수를 줄여 전체적인 질의 수행 성능을 높일 수가 있다[5].

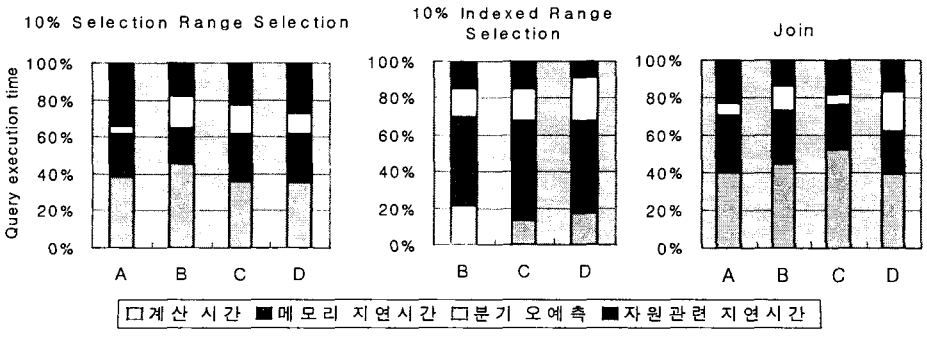

그림 1. 상용 DBMS에서 질의처리 성능 분석
Fig. 1 Performance Analysis of Query Processing in Commerce DBMS

또한, 질의처리 시 같은 질의를 반복하는 경우에 미리 집단함수를 실행하여 뷰(view)의 형태로 저장하는 선-계산(pre-computation) 방법이 제안되어 사용되고 있다. 이 방법은 저장 공간의 오버헤드와 주기적인 갱신에 따른 연산의 오버헤드가 크다는 단점이 있고, 모든 연산들이 함수에 따라 처리할 연산 영역이 정해져 있는 것은 아니기 때문에 사용자의 요구를 만족시키기는 어렵다. 이러한 연구들은 집단함수 알고리즘 자체의 효율성을 고려한 것이라기보다는 전체 질의 수행시간에 있어서 집단함수의 비중을 줄이기 위한 시도였다고 볼 수 있다.

2. 질의처리 시간의 분석

최근 하드웨어 기술의 발달로 다중 레벨 캐시 메모리, 명령어 파이프 라이닝, 분기예측을 통한 예상 수행과 같은 다양하고 복잡한 기능들을 가지는 고성능 프로세서들이 일반화되어 사용되고 있다. <그림 2>는 일반적인 프로세서의 간략화 된 구조를 나타낸다. 여기서 계산 시간을 TC, 메모리 지연으로 인한 시간TM, 분기 오 예측으로 인한 시간을 TB, 자원 관련 지연 시간을 TR, 오버랩 시간을 TOVL이라고 할 때 전체 질의 수행 시간 TQ는 아래식과 같이 나타낼 수 있다[4]. 자원 관련 지연 시간에는 TLB 미스로 인한 지연시간도 포함된다.

$$TQ = TC + TM + TB + TR - TOVL \quad \cdots\cdots (1)$$

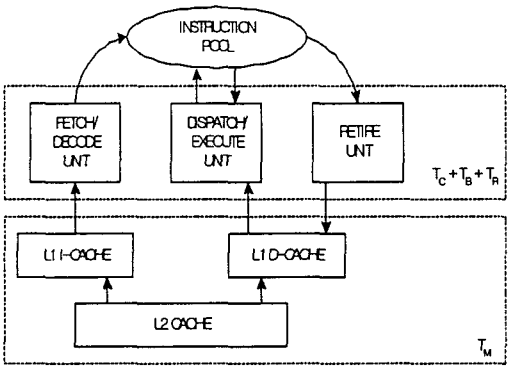

그림 2. 일반적인 프로세서의 구조
Fig. 2 Architecture of General Processor

3. 캐시 미스의 최소화 방법

최근 데이터베이스 시스템 분야에서도 캐시 메모리의 효율적인 활용방법에 대한 많은 연구들이 진행되고 있다. 이러한 노력들을 분류하면 크게 질의처리 과정[4], 인덱스 구조[6], 데이터 저장구조[7] 등에 대한 연구로 구분할 수 있다. 캐시 미스의 최소화 방법으로는 컴파일러 이용, 테이블 분해 기법, 키-포인터 추출 기법 및 데이터 군집화기법 등이 소개되고 있다.

3.1 데이터 군집화

상호 관련된 데이터를 군집화(clustering)시켜 공간 밀집도를 높일 경우 캐시 효과를 높일 수 있다. 이 방법은 해싱함수의 특성상 한 번의 테이블 스캐닝을 통해 신속한 군집화가 가능하다는 장점이 있다. 캐시 메모리의 활용 측면에서 고찰할 때, 해싱함수의 특성상 군집 수에 따라 메모리의 여러 장소에 쓰기 연산을 해야 하는 임의 접근 패턴(random access pattern)을 가지고 있다. 따라서 군집 수가 캐시블록의 수를 초과할 경우 해시 값에 따라 캐시 공간이 남아 있음에도 불구하고 캐시 미스가 발생하는 캐시블록 대체(cache trashing) 현상이 발생한다. 따라서 군집 수를 가능한 한 캐시블록 수보다 적게 유지해야 한다. 래딕스 군집화 과정에서는 키 값의 하위 비트를 이용하며 <그림 3>은 그 과정을 보여준다.

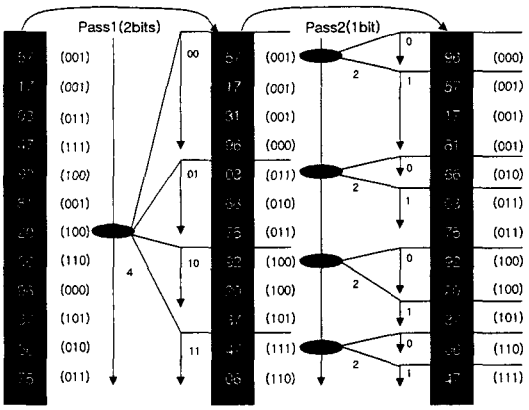

그림 3. 래딕스 군집화
Fig. 3 Radix Clustering

4. 분기예측 성능의 향상

프로그램의 실행 시 프로세서는 명령어의 분기를 예측하여 분기코드를 사전에 생성해 둔다. 질의처리 시 명령어의 분기가 많이 발생하므로 분기예측에 대한 최적화 과정이 필요하다. [8]에서는 분기 오 예측으로 인해 발생하는 손실을 최소화할 수 있는 기법으로 질의문에서 선택조건들을 비교 시 논리 연산자 AND(&&)로 처리하는 경우 보다 비트 논리 연산자 AND(&)를 사용하는 경우 두 배 이상의 성능

향상을 가져올 수 있음을 보여준다. 따라서 명령어의 분기 오 예측으로 인한 프로세서의 성능 저하가 현저히 감소하게 된다.

5. 기존 집단함수의 구현 방법

집단함수 질의처리 알고리즘은 일반적으로 정렬기반 방식과 해시기반 방식이 있다[9]. 정렬기반 방식은 먼저 그룹을 수행하는 필드에 대해서 정렬 한 뒤, 그 그룹핑 필드가 같은 것끼리 스캔해 나가면서 결과를 얻는다. 알고리즘이 간단하고 구현이 용이하나, 정렬에 사용되는 퀵 소트 알고리즘의 특성 상 랜덤 액세스를 수반하므로 해시테이블을 기반으로 하는 알고리즘 보다 캐시 성능이 저하된다는 단점이 있다.

해시기반 알고리즘은 테이블을 스캔해 나가면서 그룹핑 필드에 대하여 임시의 해시 테이블을 유지시켜 나가면서 해시테이블의 값을 갱신시켜 나가는 것으로, 일반적으로 하드웨어적인 특성인 캐시 활용과 분기예측 측면을 전혀 고려하지 않고 있다.

III. 집단함수의 효율적인 처리방법

1. 집단함수 질의처리의 특징

1.1 단항 연산자

조인이나 집합 연산에 비해 집단함수 연산은 단항 연산자로서 피연산자로 개별적인 테이블 만을 이용한다. 즉, 테이블을 한번 스캔 함으로써 질의처리가 종료된다. 테이블을 파티션하여 조인 연산 등에서 반복적인 메모리의 읽기 연산 시 캐시 효과를 제고시킬 수 있다. 래딕스-군집화 방법은 테이블의 분해과정에서 캐시 미스를 감소시키기 위한 방법에 해당된다[3]. 래딕스-군집화를 통해 한 파티션에 속하는 데이터가 다른 파티션에는 존재하지 않게 되므로 데이터 그룹 수를 감소시킬 수 있다.

1.2 갱신 집중적

조인 연산은 테이블에서 조건에 맞는 튜플을 반복적으로 검색해야 되기 때문에 읽기 집중적인 반면에 집단함수 연산은 매번 해시테이블을 갱신하는 갱신 집중적이다. 테이블의 튜플 수가 n개일 때, SUM이나 COUNT 연산의 경우 n번, MIN, MAX 연산의 경우는 평균 n/2 번의 갱신 연산이 발생한다. 캐시 메모리에 기록이 발생할 때마다 메인 메모리에도 기록하는 동시 기록(write through) 방법을 사용하는 시스템에서는 공유 메모리를 사용하는 멀티프로세서 환경에서 해시테이블의 갱신 횟수를 줄일 수 있으므로 일관성 캐시 미스를 감소시킬 수 있다.

2. 질의처리의 최적화 방법

질의처리의 최적화를 위한 방법으로 첫째, 캐싱효과의 고려로서, 데이터 그룹 수가 캐시블록 수보다 많아지면 캐시블록 대체현상이 발생하게 된다. 캐시 메모리에 기록 연산을 수행할 때마다 캐시 미스가 발생되므로 캐시 메모리의 활용성이 저하된다. 본 논문에서는 래딕스-군집화 방법을 사용하여 테이블을 여러 개로 분해한다. 분해 된 테이블에 대해 집단함수를 수행하면 테이블 당 데이터 그룹 수가 줄어들게 되므로 캐시 활용 효과를 높일 수 있다. 둘째로, 분기예측 확률을 향상시켜 분기 오 예측을 줄이는 방법으로, 다수의 테이블에서 특정한 조건을 만족하는 레코드들을 검색하는 과정에서 일반적인 논리 연산자 && 대신에 비트 논리 연산자 &를 사용하면 분기 오 예측으로 인한 오버헤드를 줄일 수 있게 된다. 컴파일러의 어셈블리 코드를 보면 일반적인 논리 연산자 &&는 피연산자 수만큼의 분기명령이 발생한다. 그러나 비트 논리 연산자 &는 피연산자 수에 관계없이 하나의 분기명령만 발생한다. 분기 오 예측 수는 분기명령 수에 비례하여 발생하므로 비트 논리 연산자 &를 사용하면 일반적인 논리 연산자 &&를 사용하는 경우에 비하여 분기 오 예측 수를 줄일 수 있게 된다.

3. 집단함수 처리 알고리즘

캐시효과를 향상시키고, 분기 오 예측을 감소시킬 수 있는 효율적인 집단함수 질의처리 알고리즘에서 전체그룹 수를 줄이기 위하여 래딕스-군집화 알고리즘을 사용하여 테이블을 분해하고 파티션을 생성한다. 생성된 파티션에서 레코드를 하나씩 스캔하면서 해시테이블을 갱신시킨다. 기존 알고리즘과의 차이점은 레코드를 스캔하고 해시테이블 갱신을 지연시켜 다음 레코드를 스캔하고 이전 값과 비교 후 테이블을 갱신시키는 방법으로 캐시쓰기를 줄이면서 테이블을 한번에 갱신시킨다. 이러한 하드웨어의 특성을 고려한 알고리즘은 <그림 4>와 같다.

```
Hardware-conscious Aggregation algorithm

procedure HW_CONSCIOUS(R, b, GF, F);
  begin
    Radix_clustering(R)  // radix-clustering
    for each partition i do begin
      tmp_aggre = 0
      for each record r in R(i) do begin
        K=Hash(r)  // Hash Key Extracting
        if (GF(prev_rec) == GF(r)) then
          tmp_aggre += grouping field of r
        else begin
          Insert/update the hashtable entry
                          for the group
          (hashtable_search())
       // Using Bit Logical AND Operator
          prev_rec = 0
        end
      end
    end
  end
  for each partition i do
    merge the result of hashtable(i)
```

그림 4. 하드웨어 특성을 고려한 집단함수 알고리즘
Fig. 4 Hardware Conscious Aggregation Algorithm

Ⅳ. 실험 및 결과분석

1. 실험 환경

실험은 Intel Xeon 프로세서에서 Perfmon을 사용하여 캐시블록 L1, L2 캐시 미스 수를 클럭 사이클 단위로 측정하였다. Calibrator를 사용하여 캐시블록 L2의 캐시 미스로 인해 발생하는 메모리 지연(memory latency) 시간을 측정하였다[10]. 2개의 Xeon 프로세서와 2GB RAM을 내장한 X4000 서버를 사용하였다. 질의문의 실행에는 좀 더 객관적인 결과를 위해 한 개의 쓰레드(thread)만 사용하였으며, 레코드의 크기가 84 바이트인 100만개의 레코드들로 구성된 데이터베이스를 실험 대상으로 하였다.

2. 결과 분석

2.1 캐싱 효과

〈그림 5〉는 전체그룹 수에 따라 캐시 미스 수의 발생을 나타낸다. 기존 집계함수 질의처리 알고리즘의 경우는 전체 그룹 수가 10개를 초과하게 되면 캐시 미스 수가 급격히 증가하는 것을 볼 수 있다. 즉, 해시테이블의 크기가 캐시 메모리의 크기보다 크기 때문에 캐시블록 대체 현상이 발생하기 때문이다. Xeon 프로세서는 캐시블록의 크기가 32B이고, 캐시 메모리의 크기가 2MB인 4-way 집합 연합 매핑(set associative mapping)방식을 사용하므로 캐시블록 대체가 발생하지 않도록 이용 가능한 캐시블록의 크기는 2MB/4 = 500B 정도가 된다.

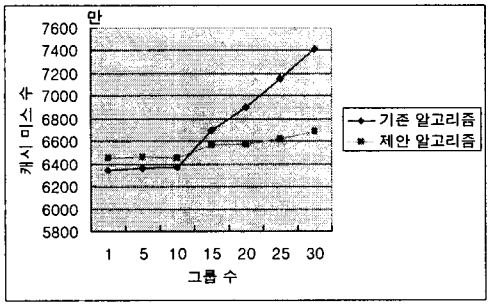

그림 5. 그룹 수에 따른 캐시 미스 수
Fig. 5 No. of Cache Miss by Group Number

〈그림 6〉은 데이터 참조 횟수를 100%로 하였을 때 캐시 미스 수가 차지하는 비율을 보여준다. 제안된 알고리즘은 캐시 성능을 효과적으로 이용할 수 있기 때문에 캐시 미스 비율이 낮게 나타나는 것으로 분석된다.

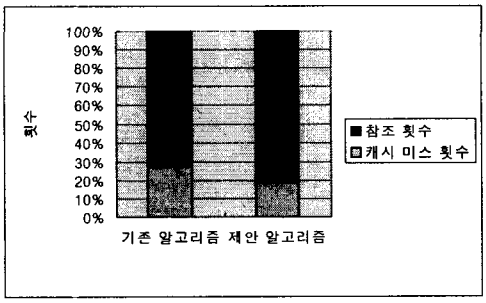

그림 6. 참조횟수 대 캐시 미스 수의 비율
Fig. 6 Rate of No. of Reference and Cache miss

반복되는 같은 그룹의 합계를 일단 레지스터에서 보관하고 그룹이 바뀔 때마다 캐시 메모리에 저장 후 캐시블록 교체 시 메모리에 저장하는 방법을 사용하였을 경우에는 성능향상이 거의 없었는데, 그 이유는 Xeon 프로세서가 캐시블록 대체 시 메모리에 저장하는 후 기록(write back) 방식의 캐시 메모리를 장착하고 있어서 캐시에 기록 횟수를 줄여도 성능향상을 가져올 수 없기 때문으로 분석된다.

2.2 분기예측 효과

동일한 해시 값을 갖는 전체그룹 수에 따라 분기 오 예측 수가 변하게 된다. 버킷의 크기에 따라 분기 오 예측 수를 실험한 결과는 〈그림 7〉과 같다. 버킷의 크기가 증가함에 따라 일반적인 논리 연산자(&&)를 사용했을 경우는 비트 논리 연산자(&)를 사용했을 경우보다 분기 오 예측 수가 급격히 증가하는 것을 확인할 수 있다.

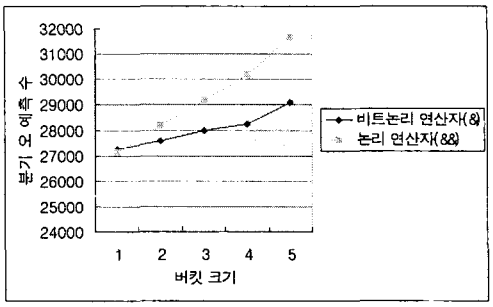

그림 7. 버킷 크기에 따른 분기 오 예측 수
Fig. 7 No. of Branch Misprediction by Bucket Size

2.3 질의처리 시간

〈그림 8〉에서는 하드웨어 특성을 고려하지 않는 기존 집단함수 처리 알고리즘과 본 논문에서 제안한 집단함수 처리 알고리즘의 수행시간을 실험을 통해 분석한 결과를 나타낸다. 본 논문에서 제안된 집단함수 처리 알고리즘이 캐시 활용과 분기예측 측면에서 기존 알고리즘보다 성능이 양호한 것으로 증명 되었다.

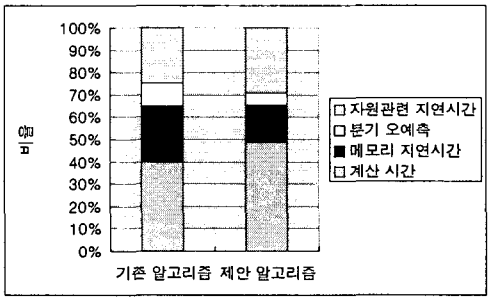

그림 8. 알고리즘의 성능 분석
Fig. 8 Performance Analysis of Algorithm

V. 결 론

대용량 데이터 웨어하우스에서 집단함수의 연산이 큰 비중을 차지하고, 처리 시 많은 시간이 소요되므로 하드웨어 특성을 고려한 질의처리 시간의 단축 방법에 대한 연구가 요구되고 있다. 캐시 미스로 인한 메모리 지연 시간을 단축시키기 위해 테이블을 분할하여 데이터 그룹수를 감소시키며, 집단함수 질의처리 시 생성되는 해시테이블의 크기를 캐시 라인의 수보다 작게 유지하며, 테이블의 분해는 캐시 효과를 향상시킬 수 있는 것으로 알려진 래딕스-파티션(radix partition) 기법을 사용하였다. 데이터의 군집화를 통한 캐시 효과를 테스트한 실험 결과는 캐시 미스 수 측면에서 약 10% 정도의 성능향상이 있었음을 보여주었다. 이는 테이블 분해 과정의 오버헤드도 포함된 것으로써 만일 데이터베이스 관리 시스템에서 테이블을 분해 된 형태로 저장한다면 그 이상의 성능 향상을 가져올 수 있음을 나타낸다. 분기 오 예측 수에 관한 실험결과는 비트 논리 연산자 &를 사용하여 약 20~30% 정도의 성능 향상이 있음을 보여주고 있다.

본 논문에서 제안한 집단함수 질의처리 알고리즘을 구현하여 기존 알고리즘과의 성능을 캐시 미스의 발생 수, 명령어의 분기예측 성능 및 전체 질의처리 시간 측면에서 비교, 분석한 결과 제안한 알고리즘이 모든 면에서 우수한 것으로 증명 되었다.

참고문헌

[1] S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP technology," Proceedings of ACM SIGMOD Conference, pp. 65-74, 1997.

[2] Intel Corporation, "IA-32 Intel Architecture Software Developers Manual Volume3 : System Programming Guide," Order Number 245472, 2001.

[3] P. A Boncz, S. Manegold, and M. L. Kersten, "Database Architecture Optimized for the New Bottleneck : Memory Access," Proceedings of the 25th VLDB Conference, pp. 54-56, 1999.

[4] J. L. Hennessy and D. A. Patterson, Computer Architecture : A Quantitative Approach, Morgan Kaufman Publishers, Inc., 2nd edition, pp. 251-280, 1996.

[5] W. P. Yan and P. Larson, "Eager Aggregation and Lazy Aggregation," Proceedings of the 21st VLDB Conference, pp. 350-354, 1995.

[6] J. Rao and K. Ross, "Making B+-trees Cache Conscious in Main Memory," Proceedings of ACM SIGMOD Conference, 2000.

[7] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving Relations for Cache Performance," Proceedings of the 27th VLDB Conference, pp. 169-180, 2001.

[8] K. A. Ross, "Conjunctive Selection Conditions in Main Memory," Proceedings of the PODS, pp. 109-120, 2002

[9] H. Garcia-Molina, J. D. Ullman, and J. Widom, Database System Implementation, Prentice Hall, pp. 237-328, 2000.

[10] S. Manegold, "The Calibrator, a Cache Memory and TLB Calibration Tool," (available from http://www.cwi.nl/manegold/Calibrator) 2000.

저 자 소 개

김 인 식

2003 충북대학교 컴퓨터공학과 공학박사

1990 ~ 현재 경북전문대학 컴퓨터정보과 부교수

김 종 겸

2002 충북대학교 컴퓨터공학과 박사수료

1994 ~ 현재 인천기능대학 자동화시스템과 부교수

정 순 기

1994 Rijksuniversiteit Groningen, Computing Science Ph.D

1986 ~ 현재 충북대학교 컴퓨터공학과 교수