# 페이지 실행시간 동기화를 이용한 다중 파이프라인 해쉬 결합

## (Multiple Pipelined Hash Joins using Synchronization of Page Execution Time)

이 규 옥 † 원 영 선 †† 홍 만 표 †††
(Kyu-Ock Lee) (Young-Sun Weon) (Man-Pyo Hong)

**요 약** 관계형 데이타베이스 시스템에서 결합 연산자는 데이타베이스 질의를 구성하는 연산자들 중 가장 많은 처리시간을 요구한다. 따라서 이러한 결합 연산자를 효율적으로 처리하기 위해 많은 병렬 알고리즘들이 소개되었다. 그 중 다중 해쉬 결합 질의의 처리를 위해 할당 트리를 이용한 방법이 가장 우수한 것으로 알려져 왔다. 그러나 이 방법은 할당 트리의 각 노드에서 필연적인 지연이 발생되는 데 이는 튜플-시험 단계에서 외부 릴레이션을 디스크로부터 페이지 단위로 읽는 비용과 이미 읽는 페이지에 대한 해쉬 결합 비용간의 차이에 의해 발생하게 된다. 본 논문에서는 이 비용 차이로 인해 발생되는 전체 시스템의 성능 저하를 방지하기 위해 페이지 실행시간 동기화 기법을 제안하였고 이 기법을 통해 각 노드에서의 처리시간을 줄이고 나아가 전체 시스템의 성능을 향상시켰다. 또한 분석적 비용 모형을 세우고 기존 방식과의 다양한 성능 분석을 통해 비용 모형의 타당성을 입증하였다.

***Abstract*** In the relational database systems, the join operation is one of the most time-consuming query operations. Many parallel join algorithms have been developed to reduce the execution time. Multiple hash join algorithm using allocation tree is one of most efficient ones. However, it may have some delay on the processing each node of allocation tree, which is occurred in tuple-probing phase by the difference between one page reading time of outer relation and the processing time of already read one. In this paper, to solve the performance degrading problem by the delay, we develop a join algorithm using the concept of 'synchronization of page execution time' for multiple hash joins. We reduce the processing time of each nodes in the allocation tree and improve the total system performance. In addition, we analyze the performance by building the analytical cost model and verify the validity of it by various performance comparison with previous method.

## 1. 서 론

병렬 데이타베이스에 대한 연구가 학술분야에서 뿐만 아니라 산업분야에서도 많은 관심을 가지고 이루어지고 있다. 특히 최근 데이타베이스의 규모가 대용량화되고 데이타베이스 연산도 매우 복잡해짐에 따라 이 분야에 대한 연구가 더욱 활발히 이루어지고 있다.

이 논문은 1999년도 두뇌한국 21 사업에 의하여 지원되었음
† 비 회 원 : 한국기계연구원 연구원
kolee@kimm.re.kr
†† 비 회 원 : 아주대학교 정보및컴퓨터공학부
ysweon@madang.ajou.ac.kr
††† 종신회원 : 아주대학교 정보및컴퓨터공학부 교수
mphong@madang.ajou.ac.kr
논문접수 : 2000년 1월 18일
심사완료 : 2000년 5월 18일

관계형 데이타베이스 연산 중에서 결합 연산은 다른 연산에 비해 월등히 많은 비용을 요구하는 연산으로서 데이타베이스의 규모가 커지고 복잡도가 높아짐에 따라 그 비용은 더욱 증가하게 된다. 미래의 데이타베이스 시스템은 이처럼 복잡한 다중 결합 질의를 어떻게 효율적으로 처리해 줄 수 있느냐가 중요한 관건이며, 그에 대한 해결책은 병렬성(parallelism)을 최대한 이용하는 것이라고 할 수 있다. 이와 같은 복잡한 다중 결합 질의를 효율적으로 처리하기 위하여 병렬성을 이용한 많은 연구가 수행되었다. 병렬성은 하나의 결합 연산에 대해서 다수의 프로세서들이 병렬로 작업을 수행하는 연산자내 병렬성(intra-operator parallelism)과 다중 질의의 병렬 수행을 위한 연산자간 병렬성(inter-operator

parallelism)으로 나눌 수 있다. 일반적으로 질의 계획(query plan)은 결합 순차 트리(join sequence tree)라 불리는 연산 트리로 변환된다. 여기서 말단 노드는 입력 릴레이션을, 중간 노드는 두 자식 노드에 할당된 두 릴레이션의 결합으로부터 생성된 결과 릴레이션을 나타낸다. 연산 트리는 그 형태에 따라 좌향(left-deep) 트리, 우향(right-deep) 트리 그리고 bushy 트리로 분류되며, 좌향 트리와 우향 트리를 선형 실행 트리(linear execution trees) 또는 순차 결합 순서(sequential join sequences) 라고 부른다.

다양한 결합 방법 중에서 해쉬 결합은 다른 결합 방법에 비해 우수한 성능을 보이며 특히, 다수의 해쉬 결합이 우향 트리 형태로 구성된 다중 해쉬 결합일 경우, 파이프라인으로 처리할 수 있다[2,5,6,7]. 하나의 파이프라인으로 구성된 다중 해쉬 결합은 여러 단계로 이루어져 있으며, 각각의 단계는 여러 개의 프로세서에 의해서 병렬로 실행될 수 있는 하나의 결합 연산으로 이루어져 있다.

최근까지는 주로 선형 실행 트리에 대한 연구가 진행된 반면, 하드웨어의 급속한 발전과 점차 복잡해지는 질의로 인해 현재에는 bushy 트리에 대한 연구가 활발히 이루어지고 있다. 또한 연산자간 병렬화와 연산자내 병렬화의 통합적 접근이라든지 결합 순서 스케줄링 및 프로세서 할당 등을 다루는 연구들이 제안되었다[4]. 최근의 연구결과들 중에서 다중 해쉬 결합 질의 처리를 위한 효율적인 방법으로 할당 트리를 이용한 정적 프로세서 할당 방법이 제안되었는데, 이 방법은 파이프라인이 가능한 bushy 트리의 노드들을 그룹핑하여 할당 트리를 생성하고 기본 릴레이션의 초기 정보, 즉 릴레이션의 카디널리티와 속성 값의 도메인 카디널리티를 이용하여 상향식(bottom-up) 방법으로 누적 실행 비용을 계산 한 후 다시 하향식(top-down) 방법에 의해 프로세서를 할당하는 방법이다[1]. 특히, 이 방법은 동기실행시간(synchronous execution time) 개념을 이용해 할당 트리 내에서 동일한 수준의 노드들이 가급적 동시에 실행이 완료 되도록 함으로서 전체적으로 지연시간을 줄이고자 하였다[4]. 그러나 이 방법에서는 할당 트리의 각 노드에 할당된 프로세서들에 대해서는 각 노드 내에서의 해쉬 결합들을 수행하는 비용 모형에 대한 충분한 분석 및 평가가 고려되지 않음으로서 각 노드 내에서 필연적으로 발생될 수 있는 지연시간에 대한 처리를 할 수 없었다. 따라서, 본 연구에서는 할당 트리 내의 한 노드, 즉 하나의 우향 트리에서의 다중 해쉬 결합에 대한 보다 정확한 비용 모형을 설정하고 다양한 성능 분석을 수행하였다. 또한, 이 비용 모형과 성능 분석 결과를 근거로 하여 한 노드 내에서의 지연시간을 최소화하고, 더 나아가 전체 질의 실행시간을 줄일 수 있는 페이지 실행시간 동기화를 이용한 파이프라인 해쉬 결합 알고리즘을 제안하였다. 또한 이에 대한 비용 모형 설계 및 성능 분석을 수행하여 그 결과를 기존의 방법과 비교하였다.

논문의 구성은, 2장에서 관련 연구로서 할당 트리를 이용한 정적 프로세서 할당 방법에 대해 살펴보고, 3장에서는 본 연구에서 제안한 페이지 실행시간 동기화를 이용한 해쉬 결합알고리즘에 대해 살펴본다. 4장에서는 제안한 방법에 대한 성능 평가를 위한 비용 모형을 제시한다. 5장에서는 제시한 비용 모형을 통해 다양한 방법으로 성능을 평가를 하며 기존의 할당 방법과 비교한다. 마지막으로 6장에서는 결론으로서 본 논문의 결과를 정리한다.

## 2. 할당 트리를 이용한 프로세서 할당 기법

해쉬 결합은 표-구축(table building) 단계와 튜플-시험(tuple probing) 단계로 구성된다. 표-구축 단계에서는 결합 속성값에 해쉬 함수를 사용하여 내부 릴레이션(inner relation)에 대한 해쉬-표(hash table)들을 구축하며, 튜플-시험 단계에서는 외부 릴레이션(outer relation)의 튜플들을 동일한 해쉬 함수를 이용하여 해쉬-표에 있는 엔트리들에 대해 하나씩 시험한다. 다중 해쉬 결합이 우향 트리의 형태를 가진다면, 먼저 표-구축 단계를 통해서 트리 내의 모든 내부 릴레이션(R)들에 대한 해쉬-표를 구축한 다음, 튜플-시험 단계를 통해 외부 릴레이션(S)의 튜플들을 차례로 읽어 결합을 수행하고 여기서 결합된 튜플들을 상위 단계로 계속적으로 흘려 보냄으로서 파이프라인 방식으로 결합을 수행하게 된다. 따라서 전체적인 질의 실행시간을 줄일 수 있는 효과를 가질 수 있다.

이와 같이 파이프라인 방식으로 다중 해쉬 결합 질의를 처리하기 위하여 bushy 트리에서 파이프라인이 가능한 노드들을 그룹핑하여 할당 트리로 변환한 후 질의를 처리하는 방법이 제안되었다. 다음은 bushy 트리 형태로 표현된 다중 해쉬 결합 질의를 할당 트리의 형태로 변환하고 정적 프로세서 할당 기법에 의해 프로세서를 할당하는 방법에 대해 살펴본다.

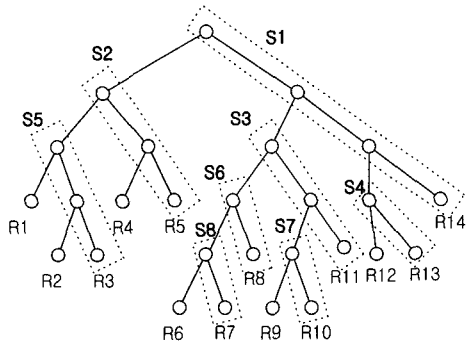
### 2.1 할당 트리

할당 트리를 생성하는 방법은 먼저 bushy 트리로 표현된 질의에서 파이프라인이 가능한 노드들을 찾는다. 그리고 파이프라인이 가능한 노드들을 그룹핑하여 할당

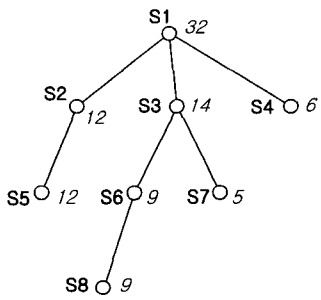
트리의 한 노드로 변환한다. 그림 1은 bushy 트리에서 파이프라인이 가능한 노드들을 식별하여 할당 트리로 변환하는 과정을 보여준다. (a)의 각각의 파이프라인의 그룹들은 (b)의 한 노드로 변환됨을 알 수 있다.

### 2.2 정적 프로세서 할당 기법

이 방법은 bushy 트리를 각 노드가 하나의 파이프라인으로 표현되는 할당 트리로 변환한 후, 상향식(bottom-up) 방법에 의해 누적실행비용(cumulative execution cost)을 계산하고 다시 하향식(top-down) 방법에 의해 각 노드에 프로세서를 할당하는 방법이다.




(a)




(b)

그림 1 Bushy 트리의 할당트리 변환

또한, 이 방법은 하나의 부(parent) 노드의 실행 전까지 모든 자식(child) 노드들이 거의 동시에 실행을 마칠 수 있도록 동기실행시간 개념을 이용하며, 결합 연산을 수행하기 전에 단지 기본 릴레이션들의 카디널리티와 속성값이 가질 수 있는 도메인의 카디널리티만을 고려하여 누적 실행 비용을 계산하고, 그 값에 의해 프로세서를 할당하는 정적 프로세서 할당 기법을 이용한다.

이러한 정적 프로세서 할당 방법의 실행 과정은 다음과 같다.

step 1 : 하나의 결합 순서 휴리스틱을 이용하여 bushy 실행 트리를 결정한다.

step 2 : Bushy 트리로부터 파이프라인이 가능한 릴레이션들을 그룹핑하여 할당 트리로 변환한다.

step 3 : 상향식 방법에 의해 할당 트리 각 노드의 누적 실행 비용을 계산한다.

step 4 : 하향식 방법에 의해 할당 트리 각 노드에 프로세서를 할당한다.

정적 프로세서 할당 기법의 문제점은 기본 릴레이션의 카디널리티와 속성값이 가질 수 있는 도메인의 카디널리티 만을 고려하여 누적 실행 비용을 계산하고 그 값에 따라 프로세서들을 할당하기 때문에 각각의 결합 결과로 생성되는 중간 릴레이션(intermediate relations)들의 크기를 전혀 예측할 수 없고 그로 인해 각 노드에 프로세서를 할당을 하기 위한 보다 정확한 workload를 예측하기 어렵다는데 있다. 따라서, 이러한 문제점을 해결하기 위한 연구들도 활발히 진행되고 있으며, 동적 프로세서 할당 방법 등과 같은 연구가 그 한 예라고 할 수 있다[9]. 하지만 이 부분은 본 연구의 주 관심분야가 아니므로 더 이상의 언급은 생략하기로 한다.

### 2.3 한 노드에서의 파이프라인 해쉬 결합

할당 트리의 한 노드는 그림 2와 같은 하나 이상의 해쉬 결합 질의를 파이프라인 방식으로 처리할 수 있는 우향 트리로 구성되어 있다. 여기서 내부 릴레이션들(R)에 대한 해쉬-표의 구축은 동시에 이루어질 수 있으며 모든 해쉬-표의 구축이 완료되면 외부 릴레이션(S)은 디스크로부터 페이지 단위로 입력되어 해쉬-표를 구축할 때 사용했던 동일한 해쉬 함수를 적용하여 한 튜플씩 해쉬-표에서 비교한다. 비교한 결과, 일치하는 튜

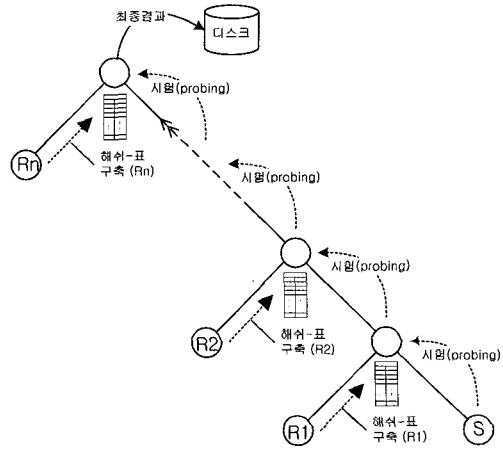



그림 2 할당 트리의 한 노드

플은 결합되어 상위-수준의 결합으로 흘러가며 일치하지 않는 튜플은 무시하게 된다. 일치하여 상위-수준의 결합으로 흘러온 튜플들은 같은 방식으로 처리되어 최상위-수준의 결합에 도달하게 되며 여기서 결합에 성공한 튜플들은 최종적으로 페이지 단위로 디스크에 저장하게 된다.

할당 트리의 한 노드를 구성하고 있는 각 해쉬 결합에는 다수의 프로세서들이 할당될 수 있으며 이 프로세서들의 합은 그 노드에 할당된 전체 프로세서 수와 같다. 각 프로세서들은 할당 트리를 구성하고 있는 해쉬 결합들 중에 어느 결합에 할당되느냐 따라 기능과 비용이 달라질 수 있으며 특정 해쉬 결합에 할당된 프로세서들 또한 기능과 그에 따른 비용이 달라질 수 있다. 예를 들면, 할당 트리의 한 노드를 구성하고 있는 해쉬 결합 중에서 최초의 해쉬 결합은 결합 연산 이외에 내부 릴레이션(R)과 외부 릴레이션(S)을 디스크로부터 페이지 단위로 읽어야 하므로 I/O 비용이 매우 커지며, 중간의 해쉬 결합들은 이미 읽은 튜플들에 대한 결합 연산만을 수행하므로 상대적으로 적은 비용을 요구한다. 마지막으로 최종 해쉬 결합에서는 결합 연산과 최종적으로 결합된 튜플들을 디스크에 다시 페이지 단위로 저장하여야 하는 I/O 비용으로 중간 해쉬 결합 보다 상대적으로 많은 비용을 요구하게 된다. 이와 같은 할당 트리 한 노드에서의 해쉬 결합의 위치 및 각 해쉬 결합 내에서의 기능에 따라 각 프로세서들이 처리해야할 비용은 많은 차이를 갖게 되며 이는 프로세서들의 불가피한 지연을 야기할 수 있으며 나아가 전체 시스템의 성능을 저하시킬 수 있다. 이 같은 문제점을 방지하기 위해서는 정확하고도 보다 정밀한 비용 모형을 세우고 다양한 성능 분석을 통해 효과적인 프로세서의 할당 방법을 마련하는 것이 필수적이다.

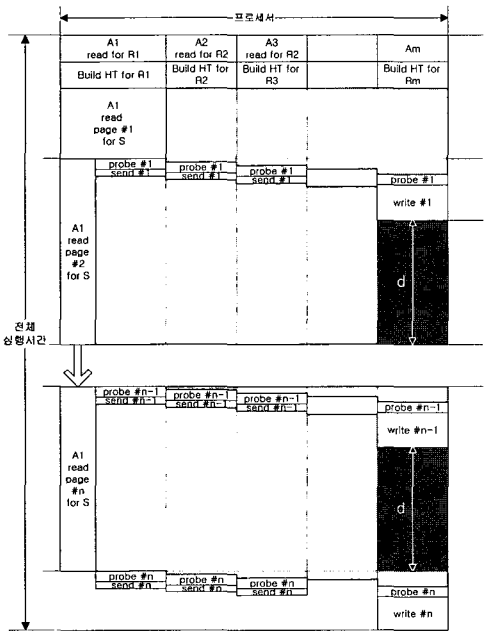



그림 3 할당 트리의 한 노드의 해쉬 결합 실행 절차

할당 트리의 한 노드, 즉 우향 트리의 해쉬 결합 실행 절차를 할당된 프로세서들의 측면에서 나타내면 그림 3과 같다.

여기서는 한 노드에 할당된 프로세서의 수가 그 노드 내의 내부 릴레이션(R)의 수 보다 많거나 같고, 각 프로세서는 I/O 전담 프로세서가 있으며, 프로세서들 사이의 통신지연시간(communication latency)은 없다고 가정한다. 또한 프로세서 구조는 분산 메모리와 공유 디스크를 갖는 다중 프로세서 구조라고 가정한다.

먼저, 할당 트리의 한 노드에 할당된 프로세서(A1, . .,Am) 전체는 그 노드에 속한 모든 내부 릴레이션들(R)을 디스크로부터 읽어서 각각의 해쉬-표를 구축한다. 구축이 완료되면, 할당된 프로세서들 중 하나의 프로세서(A1)는 외부 릴레이션(S)을 디스크로부터 페이지 단위로 읽어서 해쉬 결합을 진행하여 그 결과를 상위 해쉬 결합에 할당된 프로세서들에 전송한다. 이 같은 해쉬 결합은 상향식으로 이루어져 그 결과가 최상위 해쉬 결합에 할당된 프로세서들에 전송되며 최상위 해쉬 결합에 할당된 프로세서들은 최종적으로 결합을 수행한 후 그 결과를 디스크에 저장하게 된다. 이 작업은 외부 릴레이션(S)의 크기를 페이지 크기로 나눈 수 즉, $n$번 파이프라인 방식으로 실행되며 마지막으로 최종적으로 입력된 페이지에 대한 해쉬 결합을 상기의 절차로 수행하게 되면 할당 트리 한 노드의 처리가 완료된다.

그러나 그림 3에서 보는 바와 같이 외부 릴레이션(S)의 한 페이지를 디스크로부터 읽는 시간에 비해 이미 읽은 한 페이지에 대한 해쉬 결합 실행시간이 현저히 차이(그림 3의 d)가 나는 것을 볼 수 있다. 이 차이는 트리의 깊이(depth)가 클수록 더욱 커지며 따라서, 결합에 참여하는 프로세서들의 지연시간이 길어지고 그 영향이 전체 질의 실행시간에 미치게 된다. 또한 일반적으로 질의 최적화 과정을 통해 내부 릴레이션의 크기보다 외부 릴레이션의 크기가 상대적으로 크기 때문에 외부 릴레이션(S)의 크기가 커질수록 전체 질의 실행시간에 미치는 영향은 더욱 커지게 된다.

따라서 본 연구에서는 외부 릴레이션(S)의 한 페이지

를 디스크로부터 읽는 시간과 이미 읽은 한 페이지에 대한 해쉬 결합 실행시간과의 차이를 줄임으로서 할당 트리 내의 한 노드에서의 실행시간을 최소화하고 나아가 전체 질의 실행시간을 줄일 수 있는 페이지 실행시간 동기화 기법을 제안하고자 한다.

그림 4는 한 페이지에 대한 해쉬 결합의 정확한 실행 비용을 계산하기 위하여 그림 3에서의 한 페이지를 상세화한 그림이다. I/O 전담 프로세서가 외부 릴레이션(S)의 $i$+1번째 페이지를 디스크로부터 읽는 동안 이미 읽은 $i$번째 페이지를 A1부터 Am 프로세서에서 파이프라인 방식으로 결합하게 된다. A1 프로세서에서는 $i$번째 페이지의 튜플들을 시험(probe)하여 결합된 튜플들을 A2 프로세서에 전송한다. A2 프로세서는 A1 프로세서에서 결합되어 전송된 튜플들을 시험하여 결합된 튜플들을 A3 프로세서로 전송한다. Am 프로세서는 Am-1 프로세서에서 결합되어 전송된 튜플들을 시험하여 결합된 튜플들을 최종적으로 디스크에 페이지 단위로 저장한다. 자세한 비용 모형은 4장에 기술되어 있다.

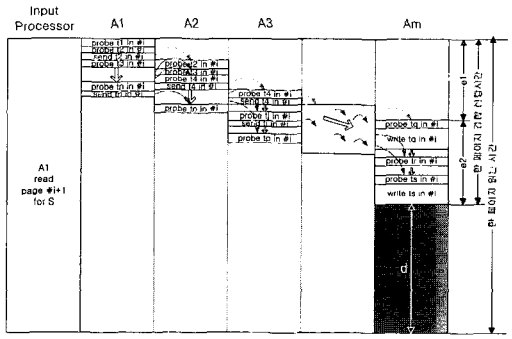



그림 4 한 페이지에 대한 해쉬 결합 실행 모형

## 3. 페이지 실행시간 동기화를 이용한 파이프라인 해쉬 결합

다중 해쉬 결합 질의의 실행시간을 최소화하기 위해서는 할당 트리의 각 노드의 실행시간을 줄임으로서 가능할 수 있다. 할당 트리의 각 노드에서는 외부 릴레이션(S)의 한 페이지를 읽는 시간과 이미 읽은 한 페이지에 대한 해쉬 결합 실행시간 사이의 차이로 인해 프로세서들의 지연시간이 발생하게 되고 결국 질의의 전체 실행시간에 영향을 미치게 된다. 따라서 본 연구에서는 페이지 실행시간 동기화를 이용하여 각 노드에서의 지연시간을 최소화하고 나아가 전체 실행시간을 줄였다.

### 3.1 프로세서 기능

할당 트리 한 노드 즉, 하나의 우향 트리를 파이프라인 방식으로 처리함에 있어서 할당된 프로세서들을 기능 위주로 나타내면 그림 5와 같이 표현할 수 있다.

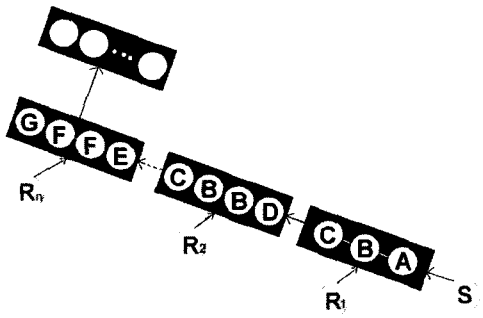



그림 5 할당 트리의 한 노드

그림 5는 깊이가 3 즉, 3개의 해쉬 결합으로 구성된 할당 트리의 한 노드를 표현한 예이다. A, B, C, D, E, F, 그리고 G 7개 타입의 프로세서들이 존재할 수 있으며 이것은 트리의 깊이가 커진다 해도 변하지 않으며, 깊이가 2일 경우는 D타입의 프로세서만 제외되며 1일 경우는 A, B, 그리고 C 타입의 프로세서에 디스크로 저장하는 기능만 추가되면 된다. 각 프로세서 타입별 실행 절차는 다음과 같다. 여기서는 유일-키에 의한 결합을 전제로 한다.

A-type processor

1. Disk read for R1;
2. Create Hash Table (HT) for R1;
3. Disk read for S;
4. Probe;
5. If match, Send to D;
   otherwise, Send to B;

B-type processor

1. Disk read for R1;
2. Create Hash Table (HT) for R1;
3. Receiving from A;
4. Probe;
5. If match, Send to D;
   otherwise, Send to C;

C-type processor

1. Disk read for R1;
2. Create Hash Table (HT) for R1;
3. Receiving from B;

4. Probe;
5. If match, Send to D or E;
   otherwise, discard;

D-type processor

1. Disk read for R2;
2. Create Hash Table (HT) for R2;
3. Receiving from A, B, or C;
4. Probe;
5. If match, Send to E;
   otherwise, Send to B;

E-type processor

1. Disk read for Rn;
2. Create Hash Table (HT) for Rn;
3. Receiving from B, C, or D;
4. Probe;
5. If match, Next step;
   otherwise, Send to F; exit;
6. Disk write for the result;

F-type processor

1. Disk read for Rn;
2. Create Hash Table (HT) for Rn;
3. Receiving from E;
4. Probe;
5. If match, Next step;
   otherwise, Send to G;
6. Disk write for the result;

G-type processor

1. Disk read for Rn;
2. Create Hash Table (HT) for Rn;
3. Receiving from F;
4. Probe;
5. If match, Next step;
   otherwise, discard;
6. Disk write for the result;

할당 트리의 한 노드 즉, 우향 트리 형태의 다중 해쉬 결합을 처리하기 위해서는 그 노드에 할당된 다수의 프로세서들이 유기적인 관계를 가지고 실행된다. 따라서, 특정 기능을 실행하는 프로세서의 실행 비용이 다른 타입의 프로세서의 실행 비용에 비해 상대적으로 많은 비용을 요구할 수 있으며 이로 인한 프로세서들의 불가피한 지연은 발생되는 것이 당연하다. A 타입의 프로세서는 다른 타입의 프로세서에 비해 월등한 I/O 비용을 요구하게 되며 이로 인한 다른 타입의 프로세서들의 지연이 발생된다. 또한, E, F, 그리고 G 타입의 프로세서들도 결합의 결과를 디스크에 저장하기 위한 I/O 비용이 추가로 요구되어 A 타입 보다는 적지만 D 타입의 프로세서 보다는 많은 비용이 요구된다. 이와 같은 프로세서의 타입별 비용 차이에서 오는 불가피한 지연을 최소화하기 위해 본 연구에서는 한 노드 즉, 다중 해쉬 결합을 실행하는 일련의 실행 과정을 정밀히 분석하였다. A 타입의 프로세서가 튜플-시험 단계에서 외부 릴레이션(S)을 디스크로부터 한 페이지 읽는 비용과 이미 읽은 페이지에 대한 처리시간 사이에 현격한 차이를 보였으며 이로 인한 프로세서들의 지연이 질의 처리시간에 가장 큰 영향을 미치며 이 지연은 외부 릴레이션(S)의 크기가 커질수록 더욱 커지게 된다. 따라서, 이 차이를 줄이는 것이 질의 처리시간을 줄이고 전체 시스템의 성능을 향상시키는 중요한 관건이 된다.

### 3.2 페이지 실행시간의 동기화

할당 트리의 한 노드 즉, 우향 트리로 표현된 다중 해쉬 결합을 파이프라인 형식으로 실행함에 있어 프로세서들의 지연시간을 최소화하고 나아가 전체 시스템의 성능을 향상시키기 위해서는 A 타입의 프로세서가 튜플-시험 단계에서 외부 릴레이션(S)을 디스크로부터 한 페이지 읽는 비용과 이미 읽은 페이지에 대한 처리시간 사이의 차이를 최대한 줄여야 한다. 이러한 차이를 줄이기 위해 본 연구에서는 외부 릴레이션(S)의 한 페이지를 읽는 시간에 다수의 페이지를 다수의 프로세서에서 동시에 읽도록 하고 그 시간 내에 이미 읽은 다수의 페이지들을 처리하도록 함으로서 페이지들을 동시에 읽는 시간과 이미 읽은 페이지들에 대한 처리가 가급적 동시에 완료될 수 있도록 하였다. 이는 상기에서 언급한 차이에 인한 프로세서들의 지연을 최소화하고 나아가 전체 시스템의 성능을 향상시킬 수 있게 되는데 본 연구에서는 이 개념을 '페이지 실행시간 동기화' 라고 하였다.

페이지 실행시간 동기화는 외부 릴레이션(S)의 한 페이지를 읽는데 소요되는 시간을 이미 읽은 외부 릴레이션(S)의 한 페이지에 대한 해쉬 결합 실행시간으로 나누고, 그 값을 $k$라고 할 때, 가능하다면 $k$개의 프로세서를 외부 릴레이션(S)의 페이지를 디스크로부터 읽는 작업에 할당하는 방법이다. 따라서, $k$개의 프로세서들은 외부 릴레이션(S)에서 $k$개의 페이지를 동시에 읽을 수 있으며, 읽혀진 $k$개의 페이지들에 대한 해쉬 결합 실행은 나머지 프로세서들에서 파이프라인 방식으로 실행하도록 한다. 결국, $k$개 단위의 페이지 입력과 이미 읽은 $k$개 페이지에 대한 다중 해쉬 결합 작업이 파이프라인

방식으로 처리될 수 있다. 따라서, 전체 실행시간에 많은 영향을 미치고 있는 외부 릴레이션(S)의 페이지 단위 입력 비용을 $k$개의 프로세서에서 동시에 실행할 수 있도록 함으로서 질의에 대한 전체 실행시간을 줄일 수 있다. 단, 여기서의 전제는 할당 트리 한 노드에 할당된 프로세서의 수는 그 노드의 내부 릴레이션(R)의 수보다는 커야 하고, 하나의 해쉬 결합에는 적어도 하나 이상의 프로세서를 할당하여야 한다는 것을 전제로 한다. 또한 프로세서 구조는 분산 메모리와 공유 디스크를 갖는 다중 프로세서 구조라고 가정한다.

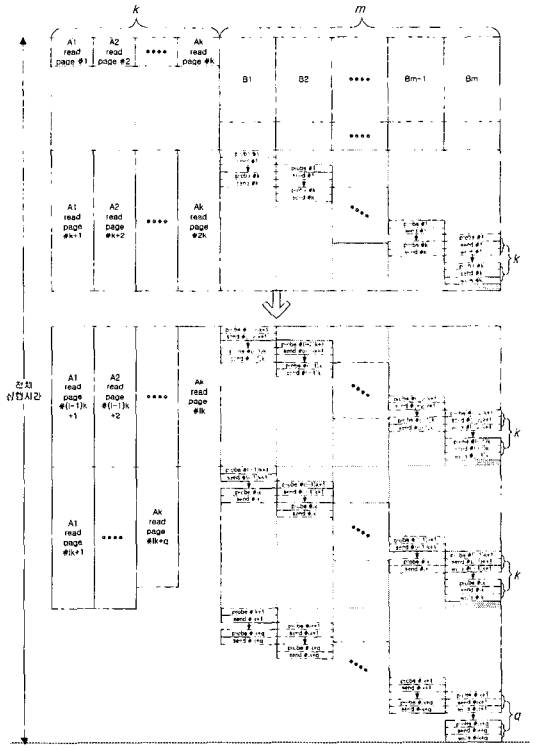



그림 6 페이지 실행시간 동기화를 이용한 다중 해쉬 결합 실행 과정

그림 6은 할당 트리의 한 노드에서 페이지 실행시간 동기화를 이용한 다중 해쉬 결합 의 실행 과정을 보여준다. 여기서, $k$는 외부 릴레이션(S)의 페이지를 읽는 작업에 할당된 프로세서의 수이며, $m$은 해쉬 결합을 수행하는 프로세서의 수로서, 그 노드에 포함된 내부 릴레이션(R)의 수 즉, 해쉬 결합의 수보다는 크거나 같아야 한다. 따라서, 그 노드에 할당된 전체 프로세서의 수를 $N$이라고 할 때, $N=k+m$ 이 된다. $k$값은 그림 4에서의 $d$ 값을 줄일 수 있는 최적의 값이며 프로세서 수가 충분하다고 더 큰 값을 $k$값으로 부여한다고 해도 해쉬 결합 쪽에서 그 시간 안에 $k$개의 페이지를 처리할 수 없으므로 페이지를 읽는 프로세서들이 오히려 지연될 수 있다. 그러므로 충분한 프로세서들이 할당되었을 경우에는 $k$개의 프로세서를 외부 릴레이션(S)을 페이지 단위로 읽는데 할당하고, 나머지 프로세서들은 모두 해쉬 결합을 수행하는 데 할당함으로서 내부 릴레이션들을 읽어서 해쉬-표를 생성하는데 소요되는 시간을 줄이도록 한다.

### 3.3 트리 깊이에 따른 효율적인 프로세서 할당

최적의 $k$값은 할당 트리의 각 노드에 해당하는 우향 트리의 깊이 즉, 다중 해쉬 결합의 수에 따라 달라진다. 왜냐하면, 한 노드 안에 해쉬 결합의 수가 많아지면 즉, 우향 트리의 깊이가 커질수록 결합률(join selectivity)에 의한 결과 릴레이션(result relation)의 크기가 감소하게 되므로 시험(probe)이나 상위 결합으로의 전송(send)에 비해 상대적으로 비용이 큰 디스크로의 저장 비용이 현저히 감소한다. 즉, $d$값이 커지게 되어 외부 릴레이션(S)의 한 페이지를 읽는 시간과 한 페이지를 해쉬 결합하는 시간과의 격차가 더욱 커지게 된다. 따라서, 할당 트리의 한 노드를 나타내는 우향 트리의 깊이 즉, 한 노드 내에 해쉬 결합의 수가 많으면 많을수록 $k$값은 커지게 된다.

참고로 4장에서 기술된 비용 모형에 따라 성능 분석을 수행한 결과 트리의 깊이에 따른 $k$값의 변화 추이는 다음의 표 1과 같다.

표 1 해쉬 결합 수(i.e. 우향트리 깊이)에 대한 $k$값의 변화 추이

| 해쉬 결합 수 i.e. 우향 트리 깊이 | 한 페이지 읽는 시간 (ms) | *한 페이지 해쉬 결합 시간(ms) | $k$ 값 |
|---|---|---|---|
| 1 | 20.166667 | 9.958879 | 2 |
| 2 | 20.166667 | 5.322595 | 3 |
| 3 | 20.166667 | 2.412640 | 8 |
| 4 | 20.166667 | 1.290138 | 15 |
| 5 | 20.166667 | 0.685678 | 29 |
| 6 | 20.166667 | 0.369691 | 55 |

*)그림 4의 e1+e2

## 4. 비용 모형

본 장에서는 새롭게 제안된 페이지 실행시간 동기화

를 이용한 다중 해쉬 결합 알고리즘에 대한 비용의 성능 평가를 위해 분석적 비용 모형을 제시한다. 비용 모형을 위한 기본 가정은 다음과 같다.

(1) 컴퓨터 구조는 분산 메모리와 공유 디스크를 갖는 다중 프로세서 구조이다.

(2) 각각의 프로세서는 하나의 내부 릴레이션에 대한 해쉬-표를 저장하기에 충분한 크기를 메모리를 갖는다.

(3) 프로세서들 사이에 통신지연시간(communication latency)은 없다.

(4) Equi-join을 근간으로 하며 유일키(primary key)를 이용한 결합이다.

본 연구의 비용 모형을 위한 매개변수 및 환경 특성 값은 표 2와 같다.

먼저, 기존 방식의 해쉬 결합에 대한 비용 모형을 세우고, 다음 본 연구에서 제안한 페이지 실행시간 동기화를 이용한 해쉬 결합 알고리즘에 대한 비용 모형을 세운다.

표 2 비용 모형과 성능 분석을 위한 매개변수와 특성 값

| 매개변수 | 의 미 | 특성값 |
|---|---|---|
| $N_P$ | 한 노드에 할당된 전체 프로세서 수 | |
| $\|\|R\|\|$ | 내부 릴레이션(R)의 카디널리티 | |
| $\|\|S\|\|$ | 외부 릴레이션(S)의 카디널리티 | |
| $P_{speed}$ | 프로세서 처리 속도 | 3 MIPS |
| $t_{pio}$ | 페이지 당 디스크 서비스 시간 | 20 ms |
| $I_{read}$ | 디스크로부터 한 페이지 읽는데 필요한 명령어 수 | 5,000 |
| $I_{write}$ | 디스크에 한 페이지 쓰는데 필요한 명령어 수 | 5,000 |
| $I_{build}$ | 해쉬-표 구축 단계에서 한 튜플 처리를 위한 명령어 수 | 100 |
| $I_{probe}$ | 튜플-시험 단계에서 한 튜플 처리를 위한 명령어 수 | 200 |
| $I_{send}$ | 한 튜플 전송에 필요한 명령어 수 | 100 |
| $P_{size}$ | 페이지 당 튜플 수 | 40 tuples |
| $\rho_i$ | 한 노드 내의 $i$-번째 결합 단계에서의 결합률 | |
| $d$ | 외부 릴레이션(S) 한페이지 읽는 시간과 한 페이지 결합 시간과의 차이 | |
| $k$ | $d$값을 줄이기 위해 할당한 외부 릴레이션(S) 입력 전담 프로세서 수 | |
| $B_P$ | $N_P$ - ($k$ + 한 노드 내의 해쉬결합 수) | |

### 4.1 기존 방식의 해쉬 결합

기존 방식의 다중 해쉬 결합에서는 외부 릴레이션(S)의 한 페이지를 디스크로부터 읽는 시간과 이미 읽은 한 페이지에 대한 해쉬 결합 처리 시간과의 차이가 발생하며 그 차이를 $d$라고 할 때, $d$는 다음과 같다.

$$d=\left(\frac{I_{read}}{P_{speed}}+t_{pio}\right) - \sum_{i=1}^{m-1}\frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1}\cdot\rho_i}\cdot I_{probe}+I_{send}}{P_{speed}} - \left(P_{size}^{m-1}\cdot I_{probe}+\frac{\frac{P_{size}^{m}}{P_{size}}\cdot I_{write}}{P_{speed}}+\frac{P_{size}^{m}}{P_{size}}\cdot t_{pio}\right) \quad (1)$$

여기서, $\rho_i(i=1,..,m)$는 할당 트리의 한 노드에 포함되어 있는 해쉬 결합들 중에서 $i$-번째 해쉬 결합의 결합률을 나타내고, $m$은 할당 트리의 한 노드에 포함되어 있는 우향 트리의 깊이 즉, 해쉬 결합의 수를 나타낸다.

또한, $P_{size}^{i}$는 $i$-1번째 해쉬 결합의 결합률에 의해 감소된 한 페이지 내에 존재하는 튜플 수를 나타내는데, $P_{size}^{i}=P_{size}^{i-1}\cdot\rho_i,(i=1,..,m)$이고 $P_{size}^{0}=P_{size}$이다. 그리고 이 $d$값은 $||S||$의 페이지 수 만큼 존재하며 그 만큼의 프로세서 지연시간이 존재하게 된다.

할당 트리 한 노드의 전체 실행시간, $T_S$는 다음과 같다.

$$T_S=\frac{\sum_{i=1}^{m}\frac{||R_i||}{P_{size}}\cdot\left(\frac{I_{read}+I_{build}}{P_{speed}}+t_{pio}\right)}{N_P} + \frac{||S||}{P_{size}}\cdot\left(\frac{I_{read}}{P_{speed}}+t_{pio}\right) + \sum_{i=1}^{m-1}\frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1}\cdot\rho_i}\cdot I_{probe}+I_{send}}{P_{speed}} + \left(P_{size}^{m-1}\cdot\frac{I_{probe}}{P_{speed}}+\frac{\frac{P_{size}^{m}}{P_{size}}\cdot I_{write}}{P_{speed}}+\frac{P_{size}^{m}}{P_{size}}\cdot t_{pio}\right) \quad (2)$$

$T_S$는 3개 항목의 합으로 구성되는데, 첫 번째 항목은 한 노드 내의 모든 내부 릴레이션들을 디스크로부터 읽어 들여 각각의 해쉬-표를 구축하는 비용이고, 둘째는, 외부 릴레이션을 디스크로부터 페이지 단위로 읽어 들이는 시간으로서 이 시간에는 읽어 들인 페이지들에 대한 해쉬 결합 실행시간이 포함된다. 셋째는, 마지막으

로 읽어 들인 페이지에 대한 해쉬 결합 실행 비용으로서 그림 4에서의 e1과 e2처럼 비용을 계산하며 이 2가지 항목에 대한 합으로 구성된다. e1은 각각의 해쉬 결합 단계에서의 결합률에 따라 결합이 성공되어 최초로 상위 수준의 해쉬 결합으로 전송(send)되는 튜플이 발생되는 데 걸리는 비용들의 합으로 구성되며, e2는 그 노드에서의 마지막 해쉬 결합을 위해 필요한 시험(probe) 비용과 결합에 성공한 튜플들에 대한 디스크로의 저장(write) 비용으로 구성된다.

### 4.2 페이지 실행시간 동기화를 이용한 해쉬 결합

기존 방식의 다중 해쉬 결합에서 발생되는 지연시간을 줄이기 위해 먼저, (1)식을 근간으로 하여 외부 릴레이션(S)의 한 페이지를 읽는 비용과 이미 읽은 한 페이지에 대한 해쉬 결합 비용을 분석하여 외부 릴레이션(S)의 페이지들을 동시에 입력하기 위한 전담 프로세서의 개수를 결정하였다. 외부 릴레이션(S)의 페이지들을 동시에 입력하기 위해 할당할 전담 프로세서의 개수를 $k$라고 할 때, $k$는 다음과 같이 결정한다.

$$k = \lceil \frac{\text{외부릴레이션}(S)\text{의한페이지입력비용}(\beta)}{\text{한페이지해쉬결합비용}(\alpha)} \rceil \quad (3)$$

여기서, 한 페이지 해쉬 결합 비용은

$$\alpha = \sum_{i=1}^{m-1} \frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1} \cdot \rho_i} \cdot I_{probe} + I_{send}}{P_{speed}} + \left( P_{size}^{m-1} \cdot \frac{I_{probe}}{P_{speed}} + \frac{\frac{P_{size}^{m}}{P_{size}} \cdot I_{write}}{P_{speed}} + \frac{P_{size}^{m}}{P_{size}} \cdot t_{pio} \right) \quad (4)$$

이고,

외부 릴레이션(S)의 한 페이지 입력 비용은

$$\beta = \frac{I_{read}}{P_{speed}} + t_{pio} \quad (5)$$

이다.

결정된 $k$값에 따라 페이지 실행시간 동기화를 적용한 다중 해쉬 결합에 대한 한 노드에서의 전체 실행시간, $T_S'$은 다음과 같다.

$$T_S' = \frac{\sum_{i=1}^{m} \frac{\| R_i \|}{P_{size}} \cdot \left( \frac{I_{read} + I_{build}}{P_{speed}} + t_{pio} \right)}{B_P} + \frac{\left( \frac{\| S \|}{P_{size}} - 1 \right)}{k} \cdot \left( \frac{I_{read}}{P_{speed}} + t_{pio} \right) + \sum_{i=1}^{m-1} \frac{\frac{P_{size}^{i-1}}{P_{size}^{i-1} \cdot \rho_i} \cdot I_{probe} + I_{send}}{P_{speed}} + \sum_{i=1}^{k} \left( P_{size}^{m-1} \cdot \frac{I_{probe}}{P_{speed}} + \frac{\frac{P_{size}^{m}}{P_{size}} \cdot I_{write}}{P_{speed}} + \frac{P_{size}^{m}}{P_{size}} \cdot t_{pio} \right) \quad (6)$$

여기서, $B_P$는 다중 해쉬 결합 초기에 내부 릴레이션(R)들을 디스크로부터 읽어서 해쉬-표를 구축하고 그 이후로는 페이지 단위로 해쉬 결합의 실행을 위해 할당되는 프로세서의 수를 말하며, 그 수는 $N_P-(k+depth)$가 된다. 만약, $B_P$가 0보다 작거나 같다면, $B_P$는 바로 트리의 $depth$가 되고, 만약 0보다 크다면, $B_P$는 $B_P+depth$가 된다.

이 의미는 $N_P-(k+depth)$가 0보다 작을 경우는 각각의 해쉬 결합에 하나씩의 프로세서들을 할당하고 $k$보다 작은 나머지 프로세서를 외부 릴레이션(S)을 페이지 단위로 읽는 전담 프로세서로 할당한다는 것이고, $N_P-(k+depth)$가 0과 같을 경우는 각각의 해쉬 결합에 하나씩의 프로세서들을 할당하고 $k$개의 프로세서를 외부 릴레이션(S)을 페이지 단위로 읽는 전담 프로세서에 할당한다는 의미이다. 마지막으로, $N_P-(k+depth)$가 0보다 클 경우는 $k$개의 프로세서를 외부 릴레이션(S)을 페이지 단위로 읽는 전담 프로세서에 할당하고 나머지 프로세서 전체를 내부 릴레이션(R)들을 디스크로부터 읽어서 해쉬-표를 구축하고 페이지 단위로 해쉬 결합의 실행하는 데에 할당한다는 의미가 된다.

## 5. 성능 분석

본 장에서는 4장에서 설정한 비용 모형을 근거로 다중 해쉬 결합 알고리즘에 대하여 기존 방식과 그리고 본 연구에서 제시한 페이지 실행시간 동기화를 이용한 방식을 다양한 방법으로 비교 분석하였다. 성능 분석을 위한 매개 변수는 표 2와 같다. 분석을 위해 내부 릴레이션들(R)과 외부 릴레이션(S)의 카디널리티는 1,000,000에서 5,000,000 튜플 정도를 사용하였고, 할당 트리의 각 노드에 포함된 각각의 해쉬 결합에서의 결합률은 난수(random number)를 생성하여 결합에 이용하였다. 결과의 신뢰성을 유지하기 위하여 1,000번 이상을 반복하여 그 결과의 평균값을 비용으로 간주하였다.

한 노드에 포함된 해쉬 결합 수에 따라서 기존 방식과 페이지 실행시간 동기화를 이용한 방식의 실행시간을 비교한 결과가 그림 7에 나와 있다.

그림 7에서 보는 바와 같이 한 노드에 포함된 해쉬 결합 수와 관계없이 그 노드에 할당된 프로세서의 수가 증가하면 두 방식 모두 노드의 전체 실행시간은 점점

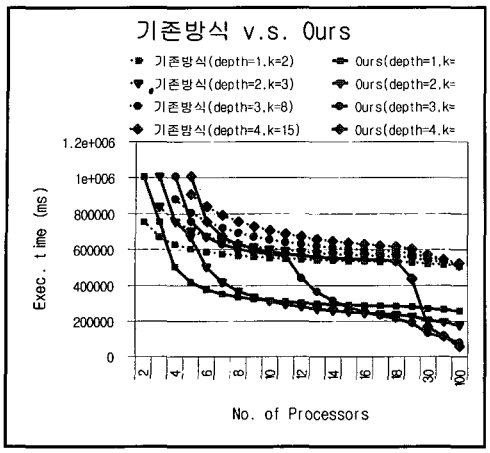



그림 7 기존 방식과 페이지 실행시간 동기화를 이용한 방식의 성능 비교

줄어든다. 하지만, 증가하는 프로세서 수에 대하여 실행시간이 줄어드는 정도의 차이는 페이지 실행시간 동기화를 이용한 방식이 기존 방식에 비해 크다는 것을 알 수 있으며 전체 실행시간도 적게 소요되는 것을 볼 수 있다.

또한, 한 노드에 할당된 전체 프로세서의 개수가 한 노드에 포함된 입력 릴레이션(내부 릴레이션(R)과 외부 릴레이션(S))의 수와 같을 경우에는 동기화를 이용한 방식이 오히려 기존 방식보다 성능이 떨어지지만, 할당되는 프로세서 수가 입력 릴레이션의 수 보다 많아질수록 페이지 실행시간 동기화를 이용한 방식의 성능이 기존 방식에 비해 월등히 우수함을 알 수 있다.

또 한가지 주목할 점은 페이지 실행시간 동기화를 이용한 다중 해쉬 결합의 경우에는 (6)식에서의 $B_P$값의 변화에 따라 실행시간의 감소추이가 급격히 변하는 경향을 보인다. 다시 말하면, (3)식에서 결정된 $k$값과, 노드의 깊이(*depth*) 즉, 노드에 포함된 해쉬 결합의 개수의 합보다 할당된 프로세서의 개수가 더 많아지기 시작하는 부분에서부터 급격한 감소 추이를 보인다. 그 이유는 $B_P$값 즉, $N_P-(k+depth)$의 값이 0 보다 큰 값으로 계속 증가할 경우, $k$개의 프로세서는 외부 릴레이션(S)를 디스크로부터 페이지 단위로 읽는 전담 프로세서로 할당하고, 나머지 프로세서 전체는 내부 릴레이션(R)을 디스크로부터 읽어서 메모리에 해쉬-표를 구축하고 이미 $k$개의 전담 프로세서들이 읽어들인 외부 릴레이션(S)들을 실제 페이지 단위로 해쉬 결합을 수행하는데 할당함으로서 그 동안 깊이(depth) 만큼의 프로세서들이 수행했던 내부 릴레이션(R)들에 대한 해쉬-표 구축 비용을 추가로 할당된 프로세서들과 분담하여 수행하게 되므로 그 비용이 점점 감소하게 되기 때문이다.

결론적으로, 페이지 실행시간 동기화를 이용한 다중 해쉬 결합의 경우에 할당 트리의 한 노드에 할당되는 프로세서의 수가 증가하면 할수록 기존 방식에 비해 그 노드의 전체 실행시간을 깊이가 1인 경우 약 50%, 2인 경우 약 33%, 3인 경우 약 13%, 4인 경우 약 7% 그리고 깊이가 5인 경우에는 약 4%까지 줄일 수 있다.

그림 8은 노드의 깊이가 3인 경우 즉, 한 노드에 포함된 해쉬 결합의 수가 3이고 또 내부 릴레이션(R)의 카디널리티가 1,000,000 튜플일 경우에 (3)식에 의해 결정되 최적의 $k$값($k$=8)에서 외부 릴레이션(S)의 카디널리티를 1,000,000 튜플에서 5,000,000 튜플로 증가시킬 경우에 두 방식의 노드 전체 실행시간 변화 추이를 분석한 결과이다.

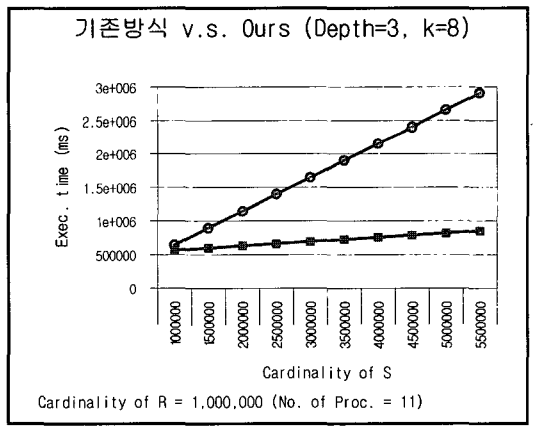



그림 8 외부 릴레이션(S) 크기 변화에 따른 성능 비교

그림 8에서 보는 바와 같이 외부 릴레이션(S)의 카디널리티가 커질수록 기존 방식의 경우 노드 전체 실행시간이 비례적으로 증가하는 반면, 페이지 실행시간 동기화를 이용한 방식은 미세한 증가 추이를 보임을 알 수 있다. 이것은 페이지 실행시간 동기화를 이용한 방식이 외부 릴레이션(S)의 크기에 거의 독립적이라는 점이며 외부 릴레이션(S)의 크기가 커질수록 기존 방식에 비해 더욱 우수한 성능을 보인다는 것을 의미한다.

## 6. 결 론

복잡한 다중 해쉬 결합 질의에 대한 처리를 할당 트리를 이용하여 처리하는 것은 가장 이상적인 방법이라

고 할 수 있다. 물론 기존의 방식처럼 프로세서 할당 문제에 있어서 프로세서를 정적으로 할당함으로서 발생되는 문제점은 있지만 이 또한 동적 프로세서 할당 방법을 통해 해결할 수 있다. 하지만 다중 파이프라인 해쉬 결합을 실행하기 위해 할당 트리의 한 노드에 할당된 프로세서들이 기능별로 상대적인 비용의 차이를 보이게 되고, 이로 인한 불가피한 프로세서들의 지연이 전체적인 질의 성능을 저하시켰다.

본 연구에서는 이 프로세서들의 지연 시간을 최대한 줄이기 위해, 할당 트리 내의 한 노드 즉, 하나의 우향 트리로 표현된 다중 해쉬 결합들에 대한 보다 정확한 비용 모형을 설정하고 다양한 성능 분석을 수행하였으며, 이를 근거로 하여 한 노드 내의 지연시간을 최소화하고 더 나아가 할당 트리로 표현된 다중 해쉬 결합의 전체 실행시간을 줄일 수 있는 페이지 실행시간 동기화를 이용한 파이프라인 해쉬 결합 알고리즘을 제안하였다.

또한, 제안한 알고리즘과 기존 방식과의 성능을 비교 분석한 결과, 한 노드에 포함된 해쉬 결합의 수에 관계없이 전체적인 질의 처리 성능이 기존 방식에 비해 우수하였고, 특히 페이지 실행시간 동기화 기법을 이용한 방식은 외부 릴레이션(S)의 크기에 독립적이면서 외부 릴레이션(S)의 크기가 커질수록 그 성능이 기존 방식에 비해 더욱 좋아짐을 보였다.

향후 연구과제로서는 한정된 프로세서 하에서 실행시간을 최소로 할 수 있는 최적의 $k$값을 찾는 것도 하나의 과제가 될 수 있으며, 또한 해쉬 필터(hash filter)와의 연계(interleave)를 통한 다중 파이프라인 해쉬 결합 알고리즘의 확장도 과제로서 의미가 있을 것이다.

## 참 고 문 헌


[1] Hui-I Hsiao, Ming-Syan Chen, "Parallel Execution of Hash Joins in Parallel Databases," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 8, No. 8, pp.872-883, Aug. 1997.

[2] Hui-I Hsiao, Ming-Syan Chen, and Philip S. Yu, "On Parallel Execution of Multiple Pipelined Hash Jois," *Proc. ACM SIGMOD*, pp.185-196, May 1994.

[3] Ming-Syan Chen, Ming-Ling Lo, Philip S. Yu, and Honesty C. Young, "Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins," *18th International Conference on VLDB*, pp.15-26, August 1992.

[4] Ming-Syan Chen, P.S. Yu, and K.L. Wu, "Scheduling and Processor Allocation for Parallel Execution of Multi-Join Queries," *Proc. 8th International Conf. Data Engineering*, pp.58-67, Feb. 1992.

[5] Ming-Syan Chen, Mingling Lo, Philip S. Yu, and Honesty C. Young, "Applying Segmented Right-Deep Trees to Pipelining Hash Joins," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 7, No. 4, August 1995.

[6] Donovan A. Schneider and D.J. DeWitt, "Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines," *Proceedings of the 16th VLDB Conference*, pp.469-480, August 1990.

[7] Mingling Lo, Ming-Syan Chen, C. V. Ravishankar, and Philip S. Yu, "On Optimal Processor Allocation to Support Pipelined Hash Joins," *Proc. ACM SIGMOD*, pp.69-78, May 1993.

[8] D.J.DeWitt, and J. Gray, "Parallel Database System: The Future of High Performance Database System," *Comm. of ACM*, pp.85-98, June 1992.



이 규 옥

1985년 충남대학교 계산통계학과 졸업. 1987년 충남대학교 계산통계학과 석사. 1987년 ~ 1989년 한국동력자원연구소 위촉연구원. 1989년 ~ 현재 한국기계연구원 선임연구원. 1995년 3월 ~ 현재 아주대학교 컴퓨터공학과 박사과정. 관심분야는 병렬데이타베이스, 컴퓨터그래픽스, 시스템 통합임



원 영 선

1993년 경기대학교 전자계산학과 이학사. 1995년 경기대학교 전자계산학과 이학석사. 1996년 ~ 현재 아주대학교 컴퓨터공학과 박사과정. 관심분야는 병렬처리, 병렬 데이타베이스, 병렬 알고리즘.

홍 만 표

정보과학회논문지 : 시스템 및 이론
제 27 권 제 1 호 참조