데이타 공유 시스템에서 동적 부하분산을 지원하는 해쉬 기반 병렬 조인 처리 기법

°문애경, 조행래
영남대학교 컴퓨터공학과

Hash-based Parallel Join Schemes Supporting Dynamic Load Balancing in Data Sharing Systems

°AeKyung Moon, HaengRae Cho
Department of Computer Engineering, Yeungnam University

요 약

해싱 함수를 이용하여 작업을 여러 노드에 분할해서 실행하는 해쉬 기반 병렬 조인 처리 기법에서 Data Skew는 특정 노드에 부하를 집중시키므로 시스템의 성능을 떨어뜨린다. 본 논문에서는 기본적인 해쉬 기반 조인 기법을 데이타 공유 시스템에 적용하고, Data Skew를 해결하기 위하여 동적 작업 할당과 부하가 집중된 노드의 작업을 다른 노드로 재할당하는 작업 재배치 방법을 제안한다 제안된 기법들의 성능을 분석하기 위하여 모의 실험을 수행하였으며, 모든 노드에서 데이타베이스가 저장된 디스크를 공유하는 데이타 공유 시스템의 경우 동적 작업 할당과 작업 재배치 방법이 효과적임을 알 수 있었다.

1. 서론

병렬 데이타베이스 시스템에서는 조인 질의의 응답 시간을 줄이기 위하여 해쉬 기반 조인 기법들을 이용하여 병렬 실행한다. 해쉬 기반 조인 기법은 해싱 함수를 이용하여 릴레이션을 분할하기 때문에 특정 노드에 더 많은 튜플들이 분배되는 Data Skew[12] 현상이 발생할 수 있다. Data Skew가 발생할 경우, 부하가 집중된 노드의 작업 지연으로 전체적인 질의 처리 시간이 길어질 수 있다

본 논문은 데이타 공유 시스템(Data Sharing System: DSS)[7,9]에서 Data Skew 문제를 해결하기 위해 동적 부하분산을 지원하는 해쉬 기반 병렬 조인 처리 기법을 제안한다. 현재까지 제안된 대부분의 병렬 조인 처리 기법은 데이타 분할 시스템(Data Partition System: DPS)을 가정하였다[3,8]. 그러나, DPS는 데이타베이스를 분할해서 저장하고 데이타를 저장하고 있는 노드에 작업을 의뢰하기 때문에 데이타 재배치 기법과 실행시 작업의 동적 할당이 어렵다 DSS를 가정한 병렬 조인 처리 기법으로는 Lu와 Tan의 연구[6]를 들 수 있다. 이 기법에서는 공유 메모리를 이용하여 작업을 동적 할당한다.

본 논문에서도 Data Skew 해결을 위해 동적 할당 기법을 이용하는데, [6]과는 달리 공유 메모리 없이 작업을 동적 할당하고 다른 노드에 할당된 작업들을 재배치 방법을 이용하여 분산한다 DSS는 모든 노드에서 데이타베이스가 저장된 디스크를 공유하는 구조이기 때문에 작업 재배치 방법이 효과적이다

본 논문의 구성은 다음과 같다. 2절에서는 DSS에서 해쉬 기반 병렬 조인 처리 기법을 설명하고, 3절에서는 성능 평가 모형 및 입력 매개 변수를 설명한다. 4절에서는 성능 평가 결과를 분석하고, 끝으로 5절에서 결론을 맺는다.

2. DSS에서의 병렬 조인 처리 기법

2.1 동적 부하분산 방법

해쉬 기반 조인 처리 기법은 조인 연산을 병렬화하기 위해서 분할 단계와 조인 단계로 나누어진다 분할 단계에서는 해싱 함수를 이용하여 두 개의 릴레이션 R과 S를 여러 개의 버킷으로 분할하여 각 처리 노드에 할당한다 조인 단계는 릴레이션 R 버킷의 튜플들로 해쉬 테이블을 만들고, 릴레이션 S의 버킷 튜플들로는 해쉬 테이블을 검색하여 조인의 결과를 산출한다 각 버킷에는 동일한 해싱 값을 가지는 튜플들만 들어있기 때문에 노드들은 독립적으로 병렬 처리가 가능하지만, 각 노드에 분배되는 튜플의 수가 균일하지 않은 경우(Data Skew)에는 특정 노드의 작업 지연으로 전체적인 질의 처리 시간이 길어질 수 도 있다 .

본 논문은 이러한 Data Skew 문제를 해결하기 위하여 동적 작업 할당과 작업 재배치라는 동적 부하분산 방법을 제안한다.

작업의 동적 할당을 위해 각 노드들은 태스크 관리자에게 작업을 요청하고 태스크 관리자는 작업을 요청한 노드에게 처리해야 할 버킷 정보를 준다. 작업 재배치 방법은 실행시 Data Skew가 발생한 노드의 버킷 오버플로우 부분을 모아서 재배치하는 것이다. 즉, 릴레이션 R의 버킷 오버플로우 부분들을 메모리 크기에 알맞게 나누면 처리 노드수가 결정되고, 태스크 관리자에 의해 필요한 처리 노드 수만큼 분산된다. 이 때, 릴레이션 S의 버킷은 릴레이션 R의 버킷이 재배치되어 실행되는 모든 노드에서 읽혀져야 하기 때문에 DPS에 비해 데이타베이스를 공유하는 DSS에 구현이 용이하다

2.2 해쉬 기반 병렬 조인 처리 기법

본 논문은 기존의 해쉬 기반 조인 기법(Simple Hash Join(SHJ)[2], Hybrid Hash Join(HHJ)[2], Grace Hash Join(GHJ)[4])들을 동적 부하 분산 방법을 고려하면서 DSS 환경으로 확장하였다.

2.2.1 Parallel Simple Hash Join(PSHJ)

SHJ를 DSS에 적용한 PSHJ에서 버킷 수는 조인 연산에 참여하는 노드 수와 동일한 것으로 가정한다. 각 노드는 릴레이션 R을 버킷 수만큼 분할하고 해딩 처리 노드로 분할된 버킷을 전송한다 버킷 번호와 처리 노드를 동일하게 정하였기 때문에 예를 들면, 노드 N_1에서 생성된 버킷 중, 버킷 B_1을 제외한 나머지는 해당 처리 노드로 전송된다.

2.2.2 Parallel Hybrid Hash Join(PHHJ)

PHHJ는 분할 단계에서 릴레이션을 처리 노드의 메모리에 알맞은 크기의 버킷으로 분할한다 HHJ에서는 첫 번째 버킷만 해쉬 테이블로 바로 만들어지게 되지만 PHHJ는 메모리 이용 효율을 증대시키기 위하여 처리 노드 수만큼의 버킷들이 해쉬 테이블로 바로 만들어진다.

예를 들면, 노드 수가 3이고 분할된 버킷 수는 10이라 가정하자. 각 노드에서는 버킷을 분할하는데 이중 버킷 B_1은 노드 N_1, B_2는 노드 N_2, B_3는 노드 N_3로 전송되어 바로 해쉬 테이블로 만들어진다. 나머지 버킷 B_4 ~ B_{10}은 디스크에 저장되고 태스크 관리자에 의해 동적 할당된다.

2.2.3 Parallel Grace Hash Join(PGHJ)

PGHJ는 PHHJ와 같이 릴레이션을 처리 노드의 메모리 크기에 알맞게 분할한다. PGHJ는 분할된 버킷을 디스크에 저장하기 때문에 데이타가 균일하게 분포된 경우, 버킷 오버플로우가 전혀 발생하지 않는다. 반면에, 릴레이션의 튜플이 비균일 분포인 경우, 버킷 오버플로우가 발생한다.

3. 성능 평가 모형

2절에서 설명한 병렬 조인 처리 기법들의 모의 실험을 위해서 개발된 DSS 환경은 그림 1에 나타난다 모의 실험은 미국의 MCC에서 개발한 CSIM 언어[11]를 이용하여 수행되었다 모든 노드는 분할 단계가 끝나면 질의 관리자에게 분할이 끝났

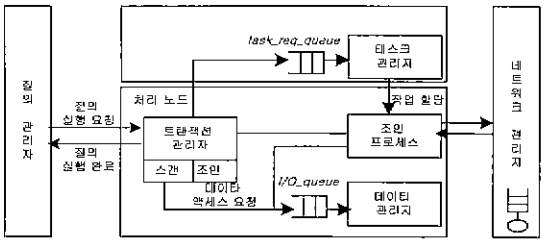

그림 1 성능 평가 모형

음을 알리고, 질의 관리자는 태스크 관리자에게 알린다. 태스크 관리자는 각 노드의 버킷 정보를 수집하여 태스크 테이블을 만들고, 그 정보를 바탕으로 작업을 동적 할당한다 조인 프로세스는 다른 노드에서 전송된 페이지들을 받아서 해쉬 테이블을 만드는 작업을 한다 본 실험에서는 질의 관리자와 태스크 관리자는 특정 노드에 하나만 존재하고 조인 프로세스는 모든 노드에 존재하는 것으로 가정한다 모의 실험을 위하여 사용한 입력 매개변수 표 1과 같다. 각 매개 변수의 구체적인 값은 [1,3,6]을 참조하였다

표 1 입력 매개 변수

시스템 구성 변수		
CPUSpeed	노드의 CPU의 처리 속도	10 MIPS
NetBandwidth	네트워크의 데이터 전송 속도	10 Mbps
NumDBMS	DBMS의 수	4 - 40
DiskTime	디스크 액세스 시간	10 ~ 30 millisec
DBSIZE	릴레이션의 크기	1000 Kbyte
오버헤드 변수		
MsgInst	메시지를 처리하기위한 명령수	1,000
PerIOInst	디스크 I/O를 위한 명령수	300
PerDataInst	데이타 처리 위한 명령수	5000
ReadTuple	튜플을 읽기 위한 명령수	500
WriteTuple	튜플을 버퍼에 쓰기 명령수	500
ProbeHashTable	해쉬 테이블을 검색 명령수	200
InsertHashTable	해쉬 테이블 삽입 명령수	100
HashTuple	튜플을 분할하는 명령수	100

4. 실험 결과

본 논문에서는 각 해쉬 기반 조인 처리 기법을 동적 부하 분산 방법을 지원하는 경우와 지원하지 않는 경우의 두 가지 형태로 구현하였는데, 전자는 동적 기법(PSHJ_D, PHHJ_D, PGHJ_D)으로 후자는 정적 기법(PSHJ_F, PHHJ_F, PGHJ_F)으로 표현한다.

Data Skew 정도를 구현하기 위하여 조인 애트리뷰트의 생성에 Zipf 분포[5]를 이용하였다. $||D_i|| = ||R|| / (i^{\theta} \sum_{j=1} \frac{1}{j^{\theta}})$에서 ($\theta$=0)인 경우는 uniform 분포에 해당되고, (θ=1)인 경우는 non-uniform 분포로 Data Skew 정도가 높다

4.1 실험 1: Uniform 분포

Uniform 분포인 경우는 그림 2에 나타나듯이 각 해쉬 기반 조인 기법의 동적 기법과 정적 기법들 사이의 성능이 거의 동일하다 그 이유는 uniform 분포로 각 노드에서 작업해야 하는 버킷

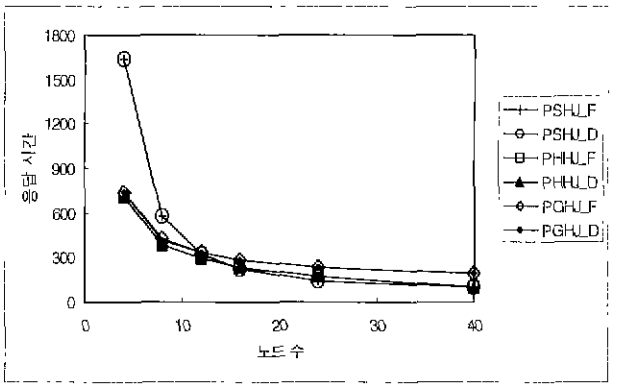

그림 2 Uniform 분포(θ=0)인 경우의 성능 비교

의 양이 같기 때문이다. 따라서, uniform 분포에서는 버킷 오버플로우 영역들을 다른 노드로 재배치하는 동적 기법의 사용이 큰 효과가 없었다

PHHJ는 처리 노드 수에 관계없이 PGHJ와 PSHJ에 비해 우수하다 처리 노드 수가 4인 경우, PSHJ와 2배 정도의 성능 차이를 보인다. PGHJ는 처리 노드 수에 대하여 성능 향상 정도가 다른 기법에 비해 낮다. 그 이유는 PGHJ는 분할 단계에서 분할된 버킷을 모두 디스크에 저장하므로 메모리 크기의 영향을 적게 받기 때문이다. PSHJ는 노드 수가 증가할수록 성능이 급속도로 좋아진다. 그 이유는 PSHJ는 분할되자마자 메모리에서 해쉬 테이블로 만들어지 메모리 크기에 영향을 많이 받기 때문이다. 즉, 노드 수가 증가하면 버킷의 수는 많아지고 버킷의 양은 적어지므로, 메모리 크기가 증가하는 효과를 보인다.

4.2 실험 2: Non-Uniform 분포

그림 3은 Non-Uniform 분포인 경우의 각 기법들의 성능 비교 결과이다. 동적 기법의 경우는 정적 기법의 경우에 비해 1.1배에서 2배 정도의 성능이 향상이 된다 그 이유는 정적 기법은 조인 연산의 실행시 한번 할당된 노드에서 작업을 마쳐야 하는 반면에 동적 기법은 부하가 많은 노드의 작업을 다른 노드에서 실행할 수 있기 때문이다.

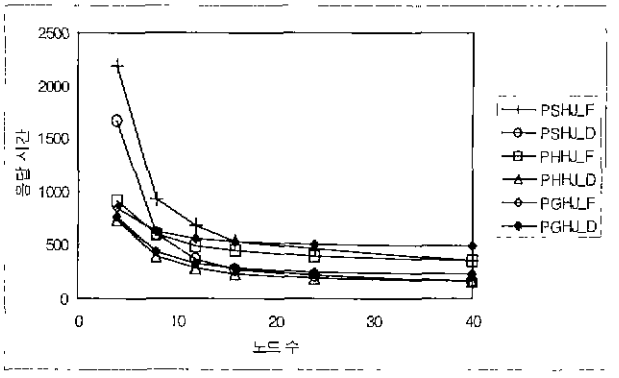

그림 3 Non-Uniform 분포(θ=1)인 경우의 성능 비교

동적 기법은 정적 기법에 비해 노드 수가 증가할수록 성능 향상의 폭이 커진다 즉, 처리 노드수가 4인 경우는 1.1배, 노드수가 40인 경우는 2배로 성능 향상된다 그 이유는 처리 노드 수가 증가할수록 각각의 노드에서 처리해야 하는 버킷 수가 줄어들어 처리 노드간 응답 시간의 편차가 커지기 때문이다.

5. 결론

본 논문은 DSS에서 Data Skew 문제를 해결하기 위해 동적 작업 할당과 작업 재배치라는 동적 부하분산 방법을 지원하는 해쉬 기반 병렬 조인 처리 기법을 구현하고, 모의 실험을 수행하였다. 본 논문에서 제안한 작업 재배치 방법은 버킷 오버플로우 부분을 메모리 크기에 알맞게 분할하여, 다른 노드로 분산하므로, 메모리 크기에 따라 유동적이다 DPS에서 해쉬 기반 조인 기법들의 성능을 비교한 DeWitt의 연구[10]에 따르면, 메모리가 큰 경우는 Data Skew에 영향을 적게 받았다. 그러므로, 본 논문에서 제안한 동적 부하분산 방법은 메모리 이용률의 변화가 심한 다중 사용자용 트랜잭션 환경으로 확장이 가능하다.

주요한 연구 결과로는 (1) 처리 노드수가 적은 경우에는 PHHJ와 PGHJ의 성능이 좋고, 처리 노드수 수가 많은 경우에는 PSHJ와 PHHJ의 성능이 좋다. (2) Non-Uniform 분포에서 동적 부하분산을 적용한 경우, 최대 2배 정도 성능이 향상되었다.

참고문헌

[1] M Carey, M Franklin and M Zaharioudakis, "Fine-Grained Sharing in a Page Server DBMS," *Proc. ACM SIGMOD* pp 359-370, 1994

[2] D DeWitt, et al., "Implementation Techniques for Main Memory Database Systems," *Proc. ACM SIGMOD*, pp 1-8, 1984.

[3] A Heal D Yuan and H Rewni, "Dynamic Data Reallocation for Skew Management in Shared-Nothing Parallel Databases," *Distributed and Parallel Databases*, 3(4) pp 271-288, 1997

[4] M Kitsuregawa, H. Tanaka, and T Moloka, "Application of Hash to Data Base Machine and Its Architecture," *New Generation Computing*, 1(1), pp 63-74, 1983

[5] D E Knuth, *The Art of Programming, Vol3' Sorting and Searching*, Addison Wesley, 1973

[6] H Lu and K Tan, "Dynamic and Load-Balanced Task-Oriented Database Query Processing in Parallel Systems," *Proc 3th Int'l Conf Extending Database Technology*, pp 357-372, 1992

[7] L. Miller, A Hurson, and S Pakzad, *Parallel Architectures for Data/Knowledge-Based Systems*, IEEE Computer Society Press, 1995.

[8] V. Poosala and Y Ioannidis, 'Estimation of Query-Result Distribution and its Application in Parallel-Join Load Balancing," *Proc 22nd Int'l Conf. VLDB*, pp 447-459, 1996

[9] E Rahm and T Sthr, "Analysis of Parallel Scan Processing in Shared Disk Database Systems," *Proc EURO-PAR Conf*, 1995

[10] D Schneider and D DeWitt, "A Performance Evaluation of Four Parallel Join Algorithms in a Shared Nothing Multiprocessor Environment," *Proc ACM SIGMOD*, pp 110-121, 1989

[11] H Schwetman, *CSIM Users Guide for use with CSIM Revision 16*, MCC, 1992

[12] C Walton, A Dale, and R. Jenevein, "A Taxonomy and Performance Model of Data Skew Effects in Parallel Joins," *Proc 17th Int'l Conf VLDB*, pp 536-548, 1993