Journal of Korea Multimedia Society Vol. 12, No. 9, September 2009(pp. 1191-1200)

그리드 데이터베이스 환경에서 동적 접근 빈도를 이용한 갱신 기법

신숭선[†], 백성하[††], 이연[†††], 이동욱[††††], 김경배[†††††], 정원일[††††††], 배해영[†††††††]

요 약

그리드 데이터베이스에서 복제본 데이터는 응용 서비스 제공을 위한 기반 정보로서 활용되며, 응용 서비스의 종류, 정보의 특성에 따라 각각의 복제본에 대한 접근 빈도에 차이가 발생한다. 특히 많은 노드에 복제본이 저장됨으로 인해 그리드 컴퓨팅 환경에서의 각각의 복제본을 관리하는 연구가 계속 되고 있다. 그의 일환으로 접근 빈도를 기반으로 한 데이터 갱신 기법이 있으며, 복제본 노드의 접근 빈도를 기반으로 데이터를 갱신한다. 하지만 동적인 접근에 대한 고려가 부족하여 그리드 환경에는 부적합하다. 따라서 본 논문에서는 갱신 정보를 관리하기 위한 동적 접근 빈도 트리를 이용한 갱신 기법을 제안한다. 동적 접근 빈도 트리는 접근 빈도를 통하여 미리 각 노드를 그룹단위로 묶어 트리를 구축하며, 또한 트리의 불균형으로 인하여 성능이 저하되는 것을 방지한다. 제안 기법은 성능평가를 통해 빠른 갱신을 보임으로써 기존의 기법에 비하여 향상된 성능을 보인다.

Update Method based on Dynamic Access-Frequency Tree in Grid Database System

Soong-Sun Shin[†], Sung-Ha Back[††], Yeon Lee[†††], Dong-Wook Lee[††††], Gyoung-Bae Kim[†††††], Worn-Il Chung[††††††], Hae-Young Bae[†††††††]

ABSTRACT

The replicas in the Grid database is utilized for a lot of application services. And for deferent services or for deferent information depends on location, the access frequency of each replica is dissimilar. When one replica is stored in many nodes, each replicas applies the week-consistency in the grid computing environment. Especially, when a node work load or operation capacity is varied from others, the replica management would cost expansive. Therefore, this paper proposed the Update Method based on Dynamic Access-Frequency Tree. The dynamic access-frequency tree is pre-constructed by grouping nodes based on each nodes access frequency to manage the replica efficiently and avoid unbalance replica tree. The performance evaluation shows the proposed methods support more quick update than current methods.

Key words: Grid(그리드), Update(갱신), AF-tree(AF트리), Replica(복제본)

※ 교신저자(Corresponding Author) : 정원일, 주소 : 충남 아산시 배방면 세출리 산 29-1(336-795) 전화 : 041)540-5698, FAX : 041)548-9667, E-mail : wnchung@hoseo.edu
접수일 : 2009년 1월 16일, 수정일 : 2009년 4월 10일
완료일 : 2009년 6월 1일
[†] 준회원, 인하대학교 정보공학과 박사과정
(E-mail : hermit@dblab.inha.ac.kr)
[††] 인하대학교 정보공학과 박사과정
(E-mail : bshzeratul@dblab.inha.ac.kr)
[†††] 준회원, 인하대학교 정보공학과 박사과정
(E-mail : leeyeon@dblab.inha.ac.kr)
[††††] 준회원, 인하대학교 정보공학과 박사과정
(E-mail : dwlee@dblab.inha.ac.kr)
[†††††] 정회원, 서원대학교 컴퓨터교육학과 조교수
(E-mail : gbkim@seowon.ac.kr)
[††††††] 준회원, 호서대학교 정보보호학과 전임강사
[†††††††] 정회원, 인하대학교 컴퓨터공학부 교수
(E-mail : hybae@inha.ac.kr)

1. 서 론

그리드 데이터베이스는 종래의 분산 데이터베이스 시스템의 기능을 기본적으로 포함하고, 대용량 자원을 공유하며, 고속 연산을 가능하게 한다. 그리드 데이터베이스를 구성하는 각 노드는 데이터의 처리 성능과 가용성 향상을 위해 서로 다른 위치에 데이터를 복제하여 저장하며, 데이터 연합, 데이터 변환, 데이터 배치, 데이터 통합 등의 기능을 제공한다[1,2]. 본 논문에서 노드는 데이터베이스 관리 시스템으로 구성되며, 또한 클러스터 시스템으로 구성된 데이터베이스 관리 시스템이다[3].

그리드 데이터베이스를 구성하는 각 노드는 수 많은 데이터들을 포함하고 있다. 모든 데이터는 어플리케이션에 따라 그 형태와 목적이 다르지만, 가용성과 성능의 향상을 위해 다른 노드에 복제본을 구성하게 된다. 하나의 데이터가 여러 복제본을 가지고, 최적의 성능을 낼 수 있는 노드에 각각 배치되는 것이다[4]. 이런 그리드 데이터베이스에서는 많은 복제본의 사용으로 그리드 데이터베이스에서 복제된 데이터들의 일관성을 유지하는 것이 중요하다. 일관성을 유지함으로써, 데이터들의 신뢰성과 합법성을 보장할 수 있다. 일관성을 유지하기 위한 기법 중에는 갱신 시의 일관성을 유지하는 기법들이 있으며, 모든 복제본에 대한 갱신 완료 시간이 짧을수록 효율적인 갱신 방법이다.

데이터의 일관성을 위한 기존의 연구로는 복제본 체인을 통한 갱신 전파방법인 UPTReC(Update Propagation through Replica Chain) 갱신 전파 방법이 있다[5]. 하지만 이 기법은 노드들의 갱신이 늦어질 수 있으며, 불필요한 전파가 많이 발생하는 문제점이 있다. 일관성을 위한 또 다른 연구로는 ARCS (Adaptable Replica Consistency Service)가 있다[6]. 이는 다수의 복제본 간에 일관성을 유지하면서도 빠른 전파가 가능하도록 다수의 마스터 노드를 사용한다. 하지만 동적인 접근에 대한 고려가 되지 않아 데이터의 갱신이 효율적이지 못하다.

본 논문에서는 접근 빈도가 동적으로 변화하는 환경에서 동시 다발적으로 발생하는 갱신 정보에 대한 관리 기법으로서 그리드 데이터베이스에서 갱신 정보 관리를 위한 동적 접근 빈도 트리를 이용한 갱신 기법을 제안한다. 동적 접근 빈도 트리는 접근 빈도가 많은 노드의 데이터부터 데이터의 갱신 연산을 수행함으로써 동일한 시간에 보다 많은 사용자에게 갱신된 데이터를 전파할 수 있다. 각각의 그룹은 하나씩의 루트 노드로서 대표 노드를 갖는다. 대표 노드를 기준으로 자식 노드들은 트리 형태의 네트워크 구조로 연결된다. 이때, 대표 노드는 그룹 내에서 가장 접근 빈도가 높은 노드가 선택되며, 접근 빈도의 우선 순위에 따라 부모 노드부터 자식 노드까지 트리 구조로 선택 된다. 대표 노드는 다시 대표 노드들끼리 트리 구조로서 연결될 수 있으며, 임의의 그룹에서 갱신 메시지가 발생하면 이는 해당 그룹에서의 갱신 메시지로서 대표 노드로 전송되는 동시에 대표 노드는 대표 노드들 간의 트리 구조에서 루트 노드로 다시 갱신 메시지를 전파하게 된다. 정보의 특성으로 인해 대표 노드는 대부분의 경우 갱신이 최초로 발생되고 저장되는 노드가 선택된다. 이러한 방법을 통하여 대표노드의 오류에 의한 데이터 갱신의 지연을 방지하며 또한 트리의 기본적인 문제점인 한 쪽으로 기울어지는 문제(skew)를 계층 구조를 이용하여 해결함으로써 성능을 향상 하였다.

제안 기법은 동적인 접근 빈도 트리를 사용함으로써 기존의 갱신 방법에 비하여 시스템 전체적인 갱신 성능을 향상 하였다. 성능 평가를 통하여 제안 기법이 기존의 기법 보다 질의 처리시에 향상된 성능을 보인다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로 기존의 분산 환경에서의 갱신 기법 대해서 소개하고, 문제점에 대하여 이야기 한다. 3장에서는 본 논문에서 갱신 정보 관리를 위한 동적 접근빈도 트리의 구조와 동작 과정 등에 대하여 설명한다. 4장에서는 제안 기법에 대한 성능 평가를 수행한다. 마지막으로 5장에서 결론 및 향후 연구를 기술한다.

2. 관련연구

2.1 그리드 데이터베이스

기하급수적으로 늘어가는 방대한 데이터들의 효율적 성능향상을 위한 노력은 지속적으로 이루어져 왔다. 이러한 노력의 일환으로 분산 데이터베이스와 데이터베이스 클러스터 그리고 그리드 데이터베이스에 대한 연구가 계속 되었다.

분산 데이터베이스는 컴퓨터 통신망을 이용하여

여러 개의 지역데이터베이스를 논리적으로 연관시킨 통합된 데이터베이스이다. 물리적으로는 분산되고 논리적으로는 집중되어 있는 형태의 구성으로 단순한 연결이 아닌 각 데이터베이스가 서로 관여를 하는 연결구조이다. 분산 데이터베이스의 장점은 데이터를 분산 배치하므로 장애에 대한 대비에 강하고 다수의 이용자가 대규모의 데이터베이스를 낮은 비용으로 공유할 수 있는 점이다. 분산 데이터베이스는 중앙 집중형 데이터베이스 보다 저비용으로 구성이 가능하며, 확장성 및 가용성에 장점이 있다.

분산 데이터베이스가 데이터의 공유, 유통 및 투명성에 초점이 맞추어 있다면, 데이터베이스 클러스터는 고속의 네트워크 연결하여 데이터 처리의 성능, 신뢰성, 가용성을 향상 시키는데 목적이 있다. 데이터베이스 클러스터에서는 작업 부하를 분산시키기 위하여 부하 분산 기법을 사용하여 데이터베이스의 작업량을 균형 있게 분배한다[7].

그리드 데이터베이스는 그리드 컴퓨팅 환경에서 분산된 데이터의 효율적 처리와 사용을 위한 데이터베이스 관리 시스템이다. 그리드 데이터베이스는 기존 분산 데이터베이스 시스템의 기능을 기본적으로 포함하는 동시에 통합, 자동화 및 가상화 등의 기능을 제공한다. 그리드 데이터베이스의 주요한 기능으로는 대용량 자원에 대한 관리와 고속 연산, 그리고 데이터 복제를 통한 가용성 향상에 있다. 그리드 데이터베이스를 구성하는 각 노드는 데이터를 처리 성능과 가용성 향상을 위해 서로 다른 위치에 복제하여 저장하며, 데이터 연합, 데이터 변환, 데이터 배치, 데이터 통합 등의 기능을 제공한다[8].

2.2 갱신 전파 기법

기존 분산 환경인 P2P에서 사용되는 갱신 전파 기법으로는 UPTReC 기법 있다[5]. 이것은 복제본 데이터를 상호 체인으로 연결하고 각각의 체인에 속한 복제본 데이터는 고유키를 갖는다. 모든 복제본 데이터는 k개의 가까운 노드들로부터 체인을 생성하고 다음노드를 탐색한다. 만약 데이터의 갱신이 발생되었다면 체인을 따라서 갱신이 진행되며 탐색 메시지는 다음 탐색 노드로 전달된다. 반복적으로 다음 노드로 메시지가 전파되며, 가장 마지막에 있는 노드에 메시지가 전달 되었다면, 모든 선택 노드는 연결이 해지되며, 모든 전달은 중지된다. 하지만 새로운 갱신이 발생하게 되면 또 다시 갱신을 위하여 노드들은 모든 탐색 노드를 찾아야 하는 문제점을 갖는다.

그리드 컴퓨팅 환경에서의 복제본 일관성 유지 서비스(GCS: Grid Consistency Service)는 복제본의 일관성을 유지 및 관리한다[9]. GCS는 데이터 그리드에서 복제본의 갱신 및 일관성 동기화를 위해 제안되었으며, 여러 단계로 분류되어 특정 레벨의 경우 약한 일관성을 지원하게 된다. 이를 위해 다양한 잠금 관리 정책이 사용된다.

갱신 전파는 복제본에 대한 일관성 유지에 있어서 중요한 부분이다. GCS의 잠금 관리 정책을 이용한 일관성 유지 기법으로서 ARCS(Adaptable Replica Consistency Service)가 있다[6]. 이는 다수의 복제본 간에 일관성을 유지하면서도 빠른 전파가 가능하도록 다수의 마스터 노드를 사용한다. ARCS는 다수의 마스터 노드 간의 갱신 전파 프로토콜과 하위 노드들 간의 접근 프로토콜을 제시하고 있다. ARCS는 데이터 그리드에서 사용되는 일관성 유지 서비스로서 빠른 갱신 전파를 위해 다수의 마스터 노드를 사용한다. 마스터 노드는 갱신이 처음 일어나는 노드로서 이러한 노드를 다수 두어 다른 노드들로 갱신 메시지를 전파함으로써 전파 과정의 병렬성을 향상시킨다.

또 다른 일관성 유지 기법으로서 빠른 일관성 유지 기법이 사용된다. 빠른 일관성 유지 기법은 보다 많은 사용자들이 최신의 정보를 얻어갈 수 있도록 접근 빈도에 따른 우선 순위를 기준으로 복제본 관리를 하는 기법이다[10]. 이 기법은 동적으로 데이터에 대한 접근 빈도가 변화하는 환경에 대한 고려가 없다. 또한 작업흐름 조정은 일종의 스케줄링 문제에 속하며, 현재 그리드 컴퓨팅 환경을 구성하는 각 노드의 자원 상태에 따라 작업을 재분배하고 병렬적으로 수행하여 처리 성능을 높이는 것에 관한 연구 기술이다[11]. 노드의 작업 부하 상태에 따른 작업 흐름 조정이 주요한 관건이 되지만, 일반적으로 그리드 컴퓨팅 환경에서는 실시간으로 자원 상태를 알아내는 것이 힘들다.

3. 동적 접근 빈도 트리를 이용한 갱신 기법

동적 접근 빈도 트리는 각각의 노드에 저장되는 동일 복제본 간의 갱신 전파를 위한 트리 형태의 네

트워크 연결 구조를 나타낸다. 일반적으로 트리 형태의 메시지 전파 구조를 사용하는 경우 우선 가장 루트에 속한 노드로 전송되고, 루트 노드로부터 자식 노드로 순차적으로 갱신 정보가 전송된다. 그러나, 이러한 전송 방식은 초기 전송 속도가 느린 동시에 루트 노드에 접속 단절이 발생할 경우 가용성 제공에 문제가 발생한다. 따라서 동적 접근 빈도 트리는 각각의 복제본 수를 일정하게 분리하여 여러 그룹으로 구성한다. 그룹 구성을 위한 기반 정보로는 지역적인 인접성 또는 네트워크 속도를 기준으로 임의의 구성이 가능하다.

3.1 동적 접근 빈도 트리의 구조

본 논문에서의 접근 빈도란 단일 노드에서의 특정 데이터에 대한 단위 시간 당 읽기 연산 요청 빈도로 정의한다. 따라서 각 노드는 저장되는 모든 데이터에 대해 접근 빈도를 기록하며, 이 정보는 주기적으로 갱신 된다.

대표 노드는 다시 대표 노드들끼리 트리 구조로서 연결될 수 있으며, 임의의 그룹에서 갱신 메시지가 발생하면 이는 해당 그룹에서의 갱신 메시지로서 대표 노드로 전송되는 동시에 대표 노드는 대표 노드들 간의 트리 구조에서 루트 노드로 다시 갱신 메시지를 전파하게 된다.

동적 접근 빈도 트리는 항상 완전 트리 형태를 유지한다. 모든 노드에서 트리의 높이가 항상 일정하고 한 쪽으로 기울어지는 쏠림 문제가(skew) 없어야 한다. 각각의 노드는 C개의 자식 노드를 가질 수 있다. 하나의 노드가 가질 수 있는 최대의 자식 노드 수 C는 임의로 정의할 수 있으며, 완전 트리로 구성함으로 인해 최하위 노드를 제외한 모든 부모 노드는 C개의 자식 노드를 포함한다.

그림 1는 접근 빈도 트리의 기본 구조를 나타낸다. 지역적으로 서로 떨어져 있는 노드들에 현재 복제본 A가 저장되어 있으며, 각 노드의 A에 대한 접근 빈도를 기준으로 순서대로 계층 구조로 연결되어 있다.

그림 1에서는 각 노드의 자식 노드의 개수를 3개로 설정한 상태이다. 현재 노드 N1의 복제본 A1에 대한 접근 빈도가 가장 높으며, 나머지 복제본들 또한 접근 빈도 크기 순서대로 정렬이 되어 있다. 또한 A1에 대한 접근 빈도가 가장 높기 때문에 현재 노드 N1이 대표 노드로 지정된 상태이다.

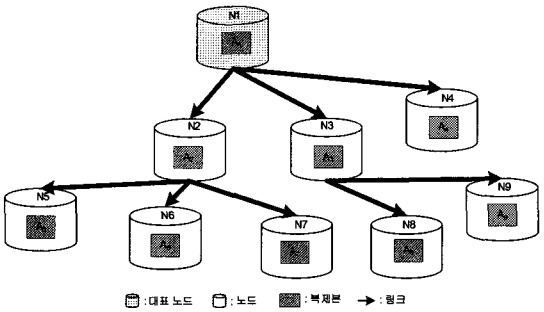


그림 1. 접근 빈도 트리의 기본 구조

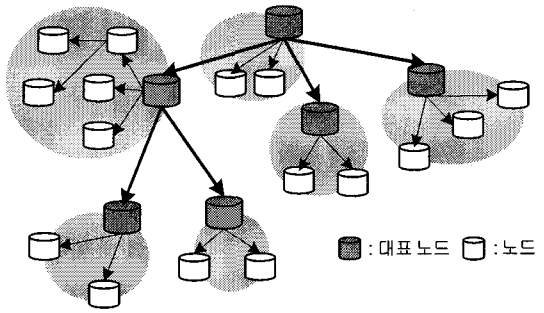


그림 2. 접근 빈도 트리의 다중 계층 구조

그림 2은 접근 빈도 트리의 대표 노드들로 구성되는 상위 단계의 그룹을 포함하는 다중 계층 구조를 나타낸다. 앞서 설명한대로 대표 노드들로 구성되는 상위 그룹 또한 각각의 접근 빈도 크기에 따라 다시 계층 구조로 구성이 된다. 따라서 갱신 메시지가 발생하였을 때 최종적으로는 여러 그룹 중에서 가장 접근 빈도 값이 높은 대표 노드가 속한 그룹에 먼저 갱신 메시지가 전송된다.

3.2 초기 접근 빈도 트리의 생성

초기 그룹 구성 시 그룹에 포함될 노드들이 선택되면, 임의의 한 노드에서 모든 노드에 대한 접근 빈도를 수집한다. 그리고 수집된 노드 정보를 바탕으로 해당 노드는 모든 노드의 접근 빈도를 최대값을 기준으로 정렬한다.

정렬된 노드들에 대한 정보는 다시 완전 트리의 형태로 구성된다. 이 때 정렬된 노드들을 기준으로 자식 노드의 개수인 C를 이용해 각각의 노드들에 대한 계층 구조를 구성한다. 예를 들어 i번째 순번에 위치한 노드의 자식 노드는 i*C, i*C+1, … i*C+(C-1)까지가 된다. 이러한 구조를 이용해 정렬된 노드들을

완전 트리 형태의 계층 구조로 구성할 수 있다. 이 경우 모든 트리의 높이를 균등하게 분배할 수 있는 장점이 있으며, 이후 노드의 삽입 & 삭제 시에도 트리의 균형을 유지한 상태에서의 동적 구성을 적용한다.

트리 구조를 생성한 후에는 모든 노드로 각 노드의 부모 노드 및 자식 노드에 대한 주소 정보와 접근 빈도에 대한 현재 정보를 전송한다. 부모와 자식 노드에 대한 주소는 갱신 메시지 전달 시 또는 접근 빈도 비교를 위해 저장되며, 접근 빈도 정보는 빈도수 비교를 위한 메시지 전송 횟수를 감소시키기 위해 저장된다. 또한 모든 노드는 루트 노드에 대한 주소 정보를 포함하고 있어, 향후 갱신 메시지 전송에 활용한다. 이에 대한 알고리즘은 다음과 같다.

알고리즘 1. 접근 빈도 트리 생성 알고리즘

```
Input:
NInfo[n]: n개의 노드 정보 // 노드의 주소 및 링크 정보를 포함
C: 자식 노드의 개수
Output:
Dtree: 접근 빈도 트리 자료 구조
Procedure CreateDtree(NInfo[n])
01: Begin
02: i = 1
03: while(i<=n)
04:         NInfo[i].demand=GetDemandFromNode(NInfo[i].address)
05:     i++
06: endwhile
07: SortNodeInfoByDemand(NInfo)
08: while(i<=n)
09:     j = 0
10:     while(j<C)
11:                 if(i*C+j>=n)then
12:                             break
13:                 endif
14:             NInfo[i].child[j+1]=NInfo[i*C+j].address
15:             NInfo[i*C+j].parent=NInfo[i].address
16:                 j++
17:     endwhile
18:     i++
19: endwhile
20: Dtree=NInfo
21: return DTree
22: End
```

알고리즘 1의 03~06 라인은 노드의 주소 정보를 통해 그룹에 포함되는 모든 노드의 접근 빈도 값을 얻어오는 함수를 사용한다. 각각의 노드에 대한 접근 빈도 값을 얻어온 후에는 SortNodeInfoByDemand 함수를 통해 노드 정보가 담긴 NInfo 배열을 접근 빈도 값을 기준으로 최대값 순으로 정렬한다(07라인). 이후에 자식 노드의 개수 C를 기준으로 정렬된 배열에 계층 구조에서의 부모와 자식 노드에 대한 주소 값을 각각의 노드의 자료 구조 안에 대입한다(08~19라인). 이때 만일 자식 노드의 계산 범위가 전체 노드의 개수를 벗어나면 루프를 종료한다. 마지막으로 계산된 결과 값을 Dtree 변수에 대입하여 이를 반환한다(20~21라인). 반환된 Dtree 정보 내에 포함되는 각 노드에 대한 새로운 정보는 각각 다시 모든 노드로 전달되어 각 노드 간에 계층 구조 구성에 사용된다.

3.3 접근 빈도 트리에서의 갱신 전파

접근 빈도 트리에서 임의의 노드에 갱신이 발생하는 경우 해당 노드는 갱신 전파에 사용할 갱신 메시지를 작성한다. 작성된 갱신 메시지는 먼저 해당 노드에서 처리될 수 있도록 질의 처리 대기열에 삽입되며, 다음으로 전달되는 노드는 해당 복제본이 속한 그룹의 루트 노드로 갱신 메시지를 전달한다. 마지막으로 전달되는 순서는 해당 복제본의 자식 복제본을 포함하는 노드들이다. 이러한 과정은 모든 노드에서 동일한 순서로 진행된다.

루트 노드는 상위 그룹이 존재하지 않는 경우 루트 노드에서의 갱신 연산 처리 후 자식 노드에게 갱신 메시지를 전파한다. 만일 해당 그룹의 상위 그룹이 존재한다면 루트 노드는 해당 노드에서의 갱신 연산 처리 후 자식 노드로 갱신 메시지를 전파하기 전에 상위 그룹의 루트 노드로 갱신 메시지를 전파하게 된다. 각각의 전파 과정에 대한 순서에 대한 예제는 아래 그림 3과 같다.

그림 3은 노드 N2에서 갱신 연산이 발생한 경우에 대한 갱신 전파 순서를 나타낸다. 갱신 연산이 발생하게 되면 N2는 먼저 해당 갱신 연산에 대한 갱신 메시지를 생성한다. 생성된 갱신 메시지는 우선 로컬 노드인 N2의 질의 처리 대기열에 삽입되며, 다음으로 루트 노드인 N1으로 전송된 후 자식 노드들인 N5, N6, N7로 순서대로 전송한다. 이때 자식 노드들에 대한 전송 순서는 자식 노드들 중 접근 빈도 값이 높은 순서를 적용한다.

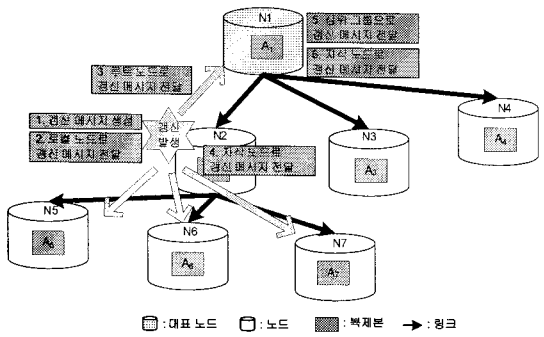


그림 3. 접근 빈도 트리에서의 갱신 전파 순서

루트 노드 N1은 갱신 메시지를 전송 받으면 N2에서와 마찬가지로 자신의 질의 처리 대기열에 갱신 메시지를 삽입한 후에 마찬가지로 상위 노드를 찾게 된다. 이 때 상위 노드는 상위 그룹에서의 루트 노드가 된다. 이후에는 루트 노드의 자식 노드들로 갱신 메시지가 전송된다. 또한 상위 노드로부터 갱신 메시지를 전달 받는 자식 노드들은 갱신 메시지를 처리한 후에 자기 자신의 자식 노드로 메시지를 전달하지만 루트 노드로는 갱신 메시지를 전달하지 않는다. 만일 전송 과정에서 이미 해당 갱신 메시지를 처리한 노드를 만나는 경우 해당 노드는 더 이상 갱신 메시지를 전달하지 않는다.

알고리즘 2는 갱신 메시지를 전달 받은 노드에서의 갱신 전파 처리 알고리즘이다. 해당 갱신 메시지를 전송 받은 노드는 먼저 갱신 메시지의 버전과 자신의 복제본의 버전을 비교한다. 만일 갱신 메시지의 내용이 이미 갱신된 내용이라면 해당 노드는 더 이상의 전파 과정을 수행하지 않고 전파 과정을 종료한다(02~04라인). 만일 갱신 메시지의 버전이 현재 복제본의 버전보다 높다면 해당 갱신 메시지를 자신의 질의 처리 대기열에 삽입 후에 자신의 자식 노드로 메시지를 전파한다(05~09라인).

알고리즘 2. 접근 빈도 트리에서의 갱신 메시지 전파 알고리즘

```
Input: Update Message(m)
Output: Update Message(m)
Procedure UpdateMessagePropagation(m)
01: Begin
02: if(m.version<=current_version)then
03:     return m
04: endif
05: InsertQueryQueue(m)
06: j = 1
07: while(j<=n)
08:     SendUpdateMessageToChildNode(m)
09: endwhile
10: End
```

3.4 동적 환경을 고려한 트리의 재구성

본 절에서는 각 복제본에 대한 접근 빈도가 동적으로 변화하는 환경에서 제안 기법을 구성하는 각 복제본의 위상 변화에 대해 설명한다.

또한 제안 기법 트리의 전체 높이가 항상 균등한 완전 트리여야 한다. 또한 접근 빈도의 변화에 따라 노드의 트리 내 위상이 변화할 수 있다. 그러나 트리의 구조가 항상 변화하기 때문에 새로운 노드가 삽입되거나 삭제될 때 완전 트리를 유지하기 위해서는 삽입 및 삭제 연산 시 항상 트리의 전체 구조를 재조정해야 하는 문제가 발생한다. 이에 대한 해결을 위해 본 절에서는 최상위 노드에 대한 주소와 트리에 포함되는 노드 수만 가지고 노드의 삽입 및 삭제 연산을 수행하면서도 완전 트리 형태를 유지할 수 있는 기법을 설명한다.

동적인 환경에서 각 노드의 접근 빈도는 항상 변화하기 때문에 부모 노드와 자식 노드의 접근 빈도의 크기가 바뀔 수 있으며 이로 인해 트리를 구성하는 각 노드의 동적 재구성 과정이 필요하다. 그러나, 주기적으로 모든 노드가 부모 노드 및 자식 노드와 접근 빈도 정보를 전송하게 되면 그에 따른 네트워크 사용량이 크게 증가하게 된다. 따라서 본 논문에서는 네트워크 사용량을 줄이기 위해 첫째, 갱신 메시지가 루트 노드로부터 전파되는 과정에서 갱신 메시지에 부모 노드의 접근 빈도를 포함하여 전송하며, 둘째, 갱신 메시지가 발생하지 않는 경우를 대비하여 각 노드에 부모 노드와 자식 노드의 접근 빈도를 저장한 후 이 정보를 기준으로 접근 빈도 기반 트리를 구축하는데 활용한다. 그러나, 두 번째 방법의 경우 시간이 지나면서 기억된 접근 빈도 값이 서로 틀려지기 때문에 두 번째 방법으로 인해 노드 교환 과정이 이루어지게 될 경우 먼저 현재의 접근 빈도 값에 대한 교환이 선행되어야 한다.

동적 노드의 재구성은 자식 노드가 부모 노드와의 접근 빈도 비교 과정을 통해 이루어진다. 자식 노드의 접근 빈도가 부모 노드의 접근 빈도보다 큰 경우 자식 노드는 자신의 위치를 부모 노드와 교환하게 된다. 이러한 과정은 트리 구조에 새로운 노드를 삽

입하는 경우에 부모 노드와의 접근 빈도 비교를 수행하는 과정과 마찬가지로 해당 노드의 접근 빈도가 더 이상 부모 노드보다 크지 않을 때까지 반복 수행된다. 단, 노드의 삽입, 삭제 과정에서는 노드의 동적 구성이 중지된다.

동적 재구성은 루트 노드가 교환되는 경우에는 다음과 같은 추가적인 연산이 필요하다.

- 루트 노드에 대한 주소 값을 그룹 내의 모든 노드로 전파

루트 노드가 변경된 경우 이에 대한 정보를 자식 노드로 반영하지 않을 경우 기존의 노드들이 이전의 루트 노드에 대한 주소 값을 가지게 되어 갱신 메시지를 전파하는 과정에서 잘못된 주소로 갱신 메시지를 전파하는 문제가 발생한다. 따라서 루트 노드가 변경되면 새롭게 바뀐 루트 노드에 대한 정보를 그룹 내의 모든 노드로 전파하여야 한다.

- 상위 그룹 또는 하위 그룹이 존재할 경우에 대한 동적 재구성

현재의 그룹보다 상위에 해당하는 또 다른 그룹이 존재하는 경우 현재 그룹의 루트 노드는 상위 그룹에서는 중간 노드로 구성될 수 있다. 따라서 상위 그룹에서의 트리 구조를 유지하기 위해서는 루트 노드의 재구성 시 연결된 또 다른 그룹에서의 재구성 과정도 함께 진행 되어야 한다.

그림 4는 접근 빈도 값의 비교를 통해 부모 노드와의 위치를 교환하게 되는 과정에 대한 예제를 나타낸다. 접근 빈도 값이 6인 자식 노드가 부모 노드에 위치 교환 요청을 하게 되면, 부모 노드는 자신의 자식 노드 중 위치 교환 신청을 한 노드보다 접근 빈도 값이 더 높은 노드가 있는지 찾아본 후 가장 높은 접근 빈도 값을 갖는 노드와 위치 교환을 하게 된다.

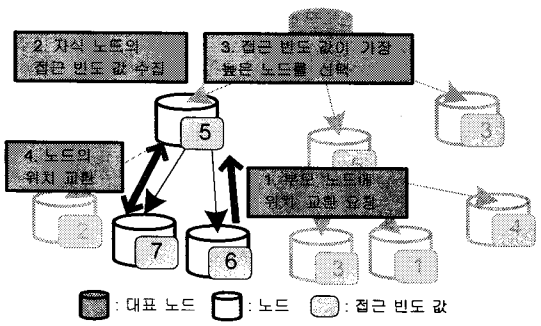


그림 4. 노드의 동적 재구성 예

4. 성능평가

4.1 실험 환경

본 논문에서는 제안 기법의 평가를 위하여 프로세서의 개수와 노드의 수에 제한을 받지 않는 C/C++ 언어 기반의 시뮬레이션 툴인 CSIM를 사용했다[12]. CSIM는 분산 환경에서의 컴퓨팅 모델, 알고리즘 평가 등 대부분의 시뮬레이션이 가능한 툴이다.

제안 기법의 구현을 위해 다수의 클라이언트와 서버를 구성하였으며, 모든 서버에서는 데이터의 접근 빈도를 기록 하였다. 실험 시간은 30000unit 동안 실험 하였으며, 실험을 위한 서버의 개수는 40개를 가지고 실험하였다. 또한 각 노드당 테이블의 개수는 20~50개 사이의 갖으며, 사이즈는 10kbyte부터 100kbyte의 크기를 갖는다. 또한 갱신 질의 입력의 간격은 1unit 이다. 시뮬레이션 구현을 위하여 MS Visual C++ 6.0을 사용하였다. 전체적인 구성은 그리드 환경에 적합하도록 복제본의 사용을 허용 하였으며, 이질적인 분산 환경에서의 성능 평가를 위하여 갱신 연산시의 처리 시간은 랜덤하게 4~7unit를 갖으며, 갱신 정보 전송 시간 또한 랜덤하게 2~4unit을 갖는다.

실험을 위해 제안 기법을 DF-tree 라 부르며 비교 대상으로서 기존의 복제본 체인과 통한 갱신 전파방법인 UPTReC(Update Propagation through Replica Chain) 갱신 전파 방법과 ARCS(Adaptive Replica Consistency Service)를 사용한다. ARCS는 데이터 그리드에서 사용되는 일관성 유지 서비스 이다.

실험 평가의 대상이 되는 인자로는 임의의 갱신 연산에 대해 모든 노드가 갱신 완료될 때 까지 걸리는 시간과 접근 빈도에 따른 데이터의 검색 비율 그리고 데이터가 처리되면서 발생되는 네트워크 처리량을 주요 평가 인자로 보고 이것을 집중 적으로 평가하였다.

4.2 실험 평가

본 실험 평가에서는 복제본 수에 따른 갱신 완료 시간, 접근 빈도에 따른 데이터의 검색 비율, 데이터 처리에 따른 네트워크 사용량을 평가하였다.

그림 5는 복제본 수에 따라 갱신 연산이 완료되는 시간을 측정한 결과 그래프이다. DF-tree의 경우

UPTReC와 ARCS 기법에 비해 빠른 완료 시간을 보인다. 이는 UPTReC와 ARCS 기법에 비해 DF-tree에서의 갱신 연산이 적은 메시지 교환 방식을 사용하기 때문이다. DF-tree에서는 이미 접근 빈도에 따른 우선 순위가 결정되어 있기 때문에 갱신 메시지가 별 다른 과정 없이 순차적으로 전달이 된다. 그러나 UPTReC는 갱신 데이터 마다 노드를 선택해야하며, ARCS 기법은 동적인 환경에서 데이터가 한쪽으로 쏠리는 현상이 발생되기 하기 때문에 다음과 같은 성능의 차이가 발생하게 된다.

그림 6는 갱신 변화에 따른 네트워크 사용량을 패킷 단위로 나타낸 그래프이다. 기존 기법이 제안 기법에 비하여 네트워크 사용량이 크게 증가함을 알 수 있다.

DF-tree를 구성하는 각 노드의 임계값(threshold), 즉 하위 노드의 개수를 2개에서 4개까지 늘려가며 평가를 하였으며, 그 외에 네트워크 속도와 각 노드의 처리 속도에 변화를 주었을 때 전체적인 성능의 변화는 있으나 UPTReC와 ARCS 기법에 비해 항상 좋은 성능을 나타내었다. 그러나, 최적의 성능을 위한 DF-tree의 임계값은 환경에 따라 다르게 적용될 수 있다.

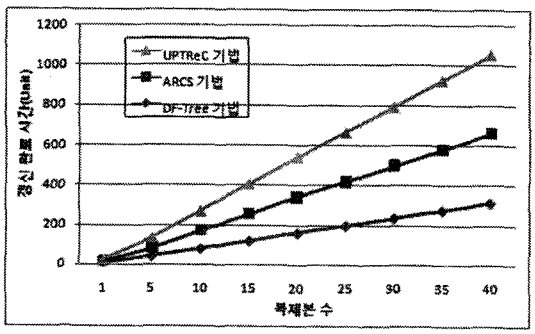


그림 5. 복제본 수에 따른 갱신 완료 시간

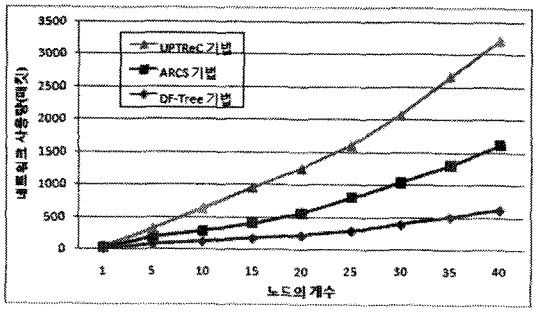


그림 6. 접근 빈도의 변화에 따른 네트워크 사용량

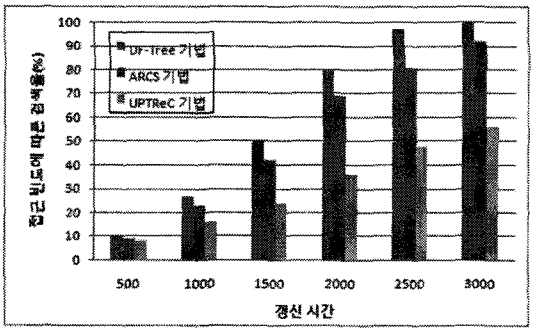


그림 7. 접근 빈도에 따른 데이터의 검색 비율

그림 7은 접근 빈도에 따른 데이터의 검색 비율을 나타낸 그림이다. 클라이언트는 랜덤으로 여러 노드에 접속하여 갱신된 정보를 검색할 때 동일한 시간에 얼마나 많은 노드에 데이터가 갱신되었나를 확인해 보기 위한 실험 평가이다.

데이터의 갱신을 한 후 여러 노드로 데이터의 갱신이 전파되는 중에 다수의 클라이언트를 통하여 갱신된 정보를 확인하였다. 이 성능평가를 통하여 DF-Tree가 성능이 가장 좋은 것을 볼 수 있었다. 제안 기법은 동일한 클라이언트의 개수 중에서 더 많이 접속 하는 노드의 데이터 갱신을 먼저 하여 줌으로써 동일한 시간에 보다 많은 사용자가 갱신된 데이터를 사용할 수 있기 때문에 발생되는 현상으로 보인다. 기존의 기법은 접근 빈도에 영향을 받지 않고 데이터의 갱신을 임의의 순서로 하기 때문에 DF-tree에 비하여 성능이 적게 나온 것을 알 수 있었다.

5. 결 론

본 연구는 분산된 노드에서 복제본 데이터를 허용하는 그리드 데이터베이스에서 갱신 정보 관리를 위한 동적 접근 빈도 트리를 이용한 갱신 기법 제안하고, 제안 기법과 다른 기법과의 성능 평가를 수행하였다. 제안 기법은 각각의 복제본 수를 일정하게 분리하여 여러 그룹으로 구성한다. 그룹 구성을 위한 기반 정보로는 지역적인 인접성 또는 네트워크 속도를 기준으로 임의의 구성을 함으로써 대표노드의 오류에 의한 데이터 갱신의 지연을 방지하며 또한 트리의 기본적인 문제점인 한 쪽으로 기울어지는 쏠림 문제(skew)를 계층 구조를 이용하여 해결하였다.

성능 평가에서는 제안 기법이 기존의 기법 보다

10~25% 정도 향상됨을 보였다. 이것은 동적 접근 트리를 구성함으로써 기존의 기법에 비하여 향상됨을 보인다.

제안 기법은 그리드 컴퓨팅이나 분산 환경 같은 노드의 수가 많고, 복제본 데이터를 갖는 환경에서 유용하게 사용될 수 있다.

본 연구에서는 네트워크의 속도와 노드의 처리 속도를 통한 연구 결과로써 평가되었으나, 향후 연구로는 그리드 환경에 부합되는 유동적인 환경에서의 네트워크 와 처리 대기 시간 등의 구성을 통한 연구가 필요하다.

참 고 문 헌

[1] P. Watson, "Databases And The Grid," January 01 2002.

[2] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," Int. J. Supercomputer Applications, pp. 200-222, 2001.

[3] H. Wolfgang and M. Gavin, "Grid Enabled Relational Database Middleware," Informational document, Global Grid Forum, Oct., 2001.

[4] S. Vazhkudai, S. Tuecke, and I. Foster, "Replica selection in the Globus data grid," In International Workshop on Data Models and Databases on Clusters and the Grid (DataGrid 2001), pp. 106-113, 2001.

[5] A. Datta, M Hauswirth and K. Aberer, "Updates in Highly Unreliable Replicated Peer-to-Peer Systems," Proceedings of IEEE ICDCS'03, pp. 0093-0106, 2006.

[6] C. Ruay-Shiung and C. Jih-Sheng, "Adaptable Replica Consistency Service for Data Grids," Information Technology: New Generations 2006., pp. 646-651, 2006

[7] Ceri and G. Pelagatti, "Distributed Databases: Principles & Systems," McGraw- Hill Company, 1984.

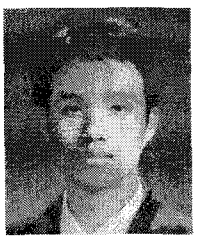
[8] M. A. N. Santisteban, J. Gray, A. S. Szalay, J. Annis, A. R. Thakar, and W. J. O'Mullane, "When database System Meet Grid," Proceedings of the 2005 CIDR Conference, pp. 154-161, 2005.

[9] D. Dillmann, W. Hoschek, J. Jean-Martinez, A. Samar, H. Stockinger, K. Stockinger, "Models for replica synchronization and consistency in a data grid," in: Tenth IEEE Symposium on High Performance and Distributed Computing (HPDC-10), San Francisco, CA, Aug. 2001.

[10] Jesis Acosta-Elias, Leandro Navarro Moldes, "A Demand Based Algorithm for Rapid Updating of Replicas," IEEE Workshop on Resource Sharing in Massively Distributed Systems (RESH'02), pp. 686-694, July. 2002.

[11] B. Hamidzadeh, Dymanic scheduling of real-time aperiodic tasks on multiprocessor architectures," System Sciences, Proceedings of the Twenty-Ninth Hawaii International Conference, Vol.1, pp. 469-478, Jan. 1996

[12] Mesquite Software. Inc. CSIM19 The Simulation Engine, 2005, http://www.mesquite.com.



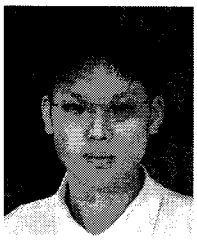
신 승 선

2006년 서원대학교 컴퓨터 교육학과 졸업(이학사)

2008년 인하대학교 컴퓨터공학부 (공학석사)

2008년~현재 인하대학교 정보공학과 박사과정

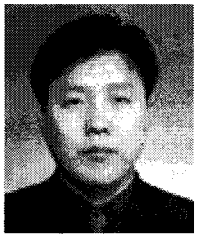
관심분야 : 공간 데이터베이스, 그리드 데이터베이스, 위치기반 서비스, u-GIS, 데이터 스트림 관리 시스템



백 성 하

2005년 인하대학교 수학과 (이학사)
2007년 인하대학교 컴퓨터공학부 (공학석사)
2007년~현재 인하대학교 정보공학과 박사과정

관심분야 : 데이터 스트림 관리 시스템, 데이터베이스 클러스터, 위치기반서비스



김 경 배

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

이 연

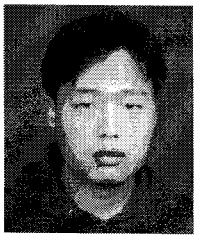
2006년 중국 중경우전대학교 지리정보공학과(이학사)
2006년~2008년 인하대학교 정보공학과(공학석사)
2008년~현재 인하대학교 정보공학과(박사과정)

관심분야 : 공간 데이터베이스, 공간 데이터웨어하우스, 지리정보 시스템, 스트림 데이터 시스템

정 원 일

1998년 인하대학교 전자계산공학과(공학사)
2004년 인하대학교 컴퓨터정보공학과(공학박사)
2004년~2006년 한국전자통신연구원 선임연구원
2007년~현재 호서대학교 정보보호학과 전임강사

관심분야 : 데이터베이스, 데이터스트림, 이동객체, 시스템 보안



이 동 욱

1996년~2003년 상지대학교 전자계산공학과(이학사)
2003년~2005년 인하대학교 컴퓨터정보공학과(공학석사)
2005년~현재 인하대학교 정보공학과 박사과정

관심분야 : 공간 데이터웨어하우스, 데이터 스트림 관리 시스템, Ubiquitous 환경을 위한 SDBMS

배 해 영

1974년 인하대학교 응용물리학과(공학사)
1978년 연세대학교 대학원 전자계산학과(공학석사)
1989년 숭실대학교 대학원 전자계산학과(공학박사)
1985년 Univ. of Houston 객원교수
1992년~1994년 인하대학교 전자계산소 소장
2004년~2006년 인하대학교 정보통신대학원 원장
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2006년~현재 인하대학교 일반대학원 원장

관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스 등