Journal of Korea Multimedia Society Vol. 11, No. 7, July 2008(pp. 928-939)

그리드 데이터베이스에서 일관성 유지를 위한 버전 관리 기법

신숭선[†], 장용일[††], 정원일[†††], 이동욱[††††], 어상훈[†††††], 배해영[††††††]

요 약

그리드 컴퓨팅 환경에서 대용량 데이터의 처리와 데이터 통합을 위하여 그리드 데이터베이스 관리 시스템이 사용되며, 각 노드의 처리 성능과 가용성 향상을 위하여 서로 다른 위치에 복제본 데이터를 갖는다. 그리드 데이터베이스 환경에서 동시 다발적으로 저장된 복제본 데이터에 대한 갱신이 발생하는 경우 그 우선순위가 모호해져 갱신에 대한 전파 순서가 섞이게 되고 결국 데이터에 대한 일관성이 없어지는 문제가 발생할 수 있다. 이러한 문제를 해결하기 위하여 본 논문에서는 그리드 데이터베이스에서 일관성 유지를 위한 버전 관리 기법을 제안한다. 제안 기법은 버전 관리자라는 버전 관리 프로세서를 사용하여 각 복제본 데이터들의 버전을 관리한다. 버전 관리자는 초과용량 대기열과 우선갱신 대기열 갖고, 노드에서는 우선순위 대기열을 통하여 각 노드들에서 발생하는 갱신 연산의 일관성을 유지함으로써 기존의 기법보다 안정적이고 빠른 갱신을 나타낸다. 제안 기법은 성능평가를 통해 안정적이고 빠른 갱신을 보임으로써 기존의 기법에 비하여 향상된 성능을 보인다.

Version Management Method for Consistency in the Grid Database

Soong-Sun Shin[†], Yong-Il Jang[††], Weon-Il Chung[†††], Dong-Wook Lee[††††], Sang-Hun Eo[†††††], Hae-Young Bae[††††††]

ABSTRACT

The grid database management system is used for large data processing, high availability and data integration by grid environment. It has a replica for processing performance and high availability in each node. The grid database has a problem of inconsistency, when the update is occurred with a co-incidently frequent. To solve this problem, in this paper proposed a version management method for consistency in the grid database. Proposed version manager manages a version of each replica. The version manager keeps a consistency of update operation when is occurred at each node by using a pending queue and waiting queue. Also the node keeps a consistency using a priority queue. So, proposed method has stable and fast update propagation. The proposed method shows stable and faster update propagation than traditional method by performance evaluation.

Key words: Grid database(그리드 데이터베이스), Version management(버전 관리자), Data replication (복제본)

※ 교신저자(Corresponding Author) : 정원일, 주소 : 아산시 배방면 세출리 165(336-795), 전화 : 041)540-5984, FAX : 041)548-9667, E-mail : wnchung@hoseo.edu

접수일 : 2008년 1월 17일, 완료일 : 2008년 4월 30일

[†] 준회원, 인하대학교 정보공학과 박사과정
(E-mail : hermit@dblab.inha.ac.kr)
[††] 준회원, 텔코웨어 DB솔루션팀
(E-mail : yijang@telcoware.com)
[†††] 준회원, 호서대학교 정보보호학과 전임강사
[††††] 인하대학교 정보공학과 박사과정
(E-mail : dwlee@dblab.inha.ac.kr)
[†††††] 인하대학교 정보공학과 통합박사과정
(E-mail : eosanghun@dblab.inha.ac.kr)
[††††††] 정회원, 인하대학교 컴퓨터공학부 교수
(E-mail : hybae@inha.ac.kr)

※ 본 연구는 건설교통부 첨단도시기술개발사업 - 지능형 국토정보기술혁신 사업과제의 연구비지원(07국토정보 C05)에 의해 수행되었습니다.

1. 서 론

그리드 데이터베이스(Grid database)는 그리드 컴퓨팅 환경에서 분산된 데이터의 효율적 처리와 사용을 위한 데이터베이스 관리 시스템이다. 그리드 데이터베이스는 종래의 분산 데이터베이스 시스템의 기능을 기본적으로 포함하고, 대용량 자원을 공유 및 고속 연산을 가능하게 한다. 그리드 데이터베이스를 구성하는 각 노드(여기서, 노드란 데이터베이스 관리 시스템이나 클러스터 시스템으로 구성되어진 데이터베이스 관리 시스템이다.)는 데이터의 처리 성능과 가용성 향상을 위해 서로 다른 위치의 데이터를 복제하여 저장하며, 데이터 연합(federation), 데이터 변환(conversion), 데이터 배치(allocation), 데이터 통합(integration) 등의 기능을 제공한다[1,2,3].

그리드 데이터베이스에서는 가용성과 확장성 그리고 데이터 처리의 성능을 향상하기 위하여 동일한 정보를 다른 노드에 중복 복제하여 저장 관리한다. 복제본의 사용으로 시스템의 가용성은 증가되었지만 복제본 데이터의 변경으로 인한 일관성 유지는 더욱 어려워졌다. 이런 일관성 유지를 위하여 많은 기법들이 연구되고 있다[4].

그 중 [5,6]의 갱신 전파 방법은 사용자 요구 기반을 고려하는 알고리즘으로 사용 빈도가 높은 순서로 갱신을 해주는 방법이 있다. 하지만 갱신이 발생하면 인근 노드의 사용빈도를 고려하여 사용빈도가 높은 노드를 중심으로 갱신을 전파하는 문제점이 있다. 이러한 문제를 해결한 또 다른 갱신 전파 방법으로는 수요기반 트리를 이용한 일관성 유지 기법이 있다[7].

[7]은 그리드 환경에서 응용 서비스의 종류 및 데이터의 특성에 따라서 각각의 복제본에 대한 사용빈도가 다른 특성을 가지고 수요기반 트리(Demand Tree)를 사용하여 데이터의 일관성을 유지한다. [7]은 수요기반 트리를 약한 일관성 기법으로 수요가 많은 노드의 데이터부터 갱신 연산을 수행하여 동일한 시간에 보다 많은 사용자에게 갱신된 데이터를 제공해 준다. 하지만 수요기반 트리를 통한 연구에서는 사용자의 수요 빈도에 따라서 갱신 데이터 전파에만 연구가 집중되어, 동시 다발적으로 발생하는 갱신 연산에 대한 일관성 유지에 대한 연구는 부족하다. 유동적으로 변경되는 그리드 환경에서 복제본에 대하여 업데이트 갱신이 발생하고 있을 경우, 또 다른 업데이트가 발생하여 데이터의 갱신 연산이 발생한 경우를 동시 다발적인 갱신 연산이라 정의한다. 이렇게 동시 다발적으로 갱신 연산이 발생할 수 있는 그리드 환경에 부합되는 데이터의 갱신 기법이 필요하다.

본 연구에서는 수요기반 트리를 이용한 그리드 데이터베이스에서 일관성 유지를 위한 버전 관리 기법을 제안한다. 제안 기법은 기존의 수요기반 트리를 확장하여 버전 관리자라는 버전 관리 프로세서를 사용함으로써 복제본들의 버전을 관리하여 복제본 데이터의 일관성을 유지한다. 버전 관리자는 버전 정보에 대한 구조체와 작업 대기열을 가지고 이전 버전 값과 현재 버전 값을 기록한다. 이 두 가지 값은 이후에 복제본 갱신 시 일관성을 유지하는데 이용된다. 버전 관리자에서 관리되는 작업 대기열은 동시 다발적으로 발생하는 갱신 연산에 대한 처리 과정에서 일부 노드의 작업 부하로 인해 새로운 갱신 연산 처리를 중지시켜야 하는 경우 이에 대한 관리를 위해 사용된다. 작업 대기열은 중지된 기간 동안 새롭게 갱신 연산이 발생되는 노드의 위치 정보를 기억하였다가, 향후 작업 부하가 완화되면 대기열에 쌓여 있는 정보를 통해 다시 갱신 연산을 처리함으로써 데이터의 일관성을 보장한다.

제안 기법은 성능평가에서 버전 관리자를 통한 일관성 관리 기법으로 기존의 기법에 비하여 데이터 갱신 연산 시간이 향상된 것을 보인다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로 그리드 데이터베이스에 대해 기술하고, 기존의 갱신 기법에 대하여 소개한다. 3장에서는 본 논문에서 제안하는 버전 관리 구조와 이를 통한 버전 관리 기법에 대하여 설명하고, 4장에서는 제안 기법에 대한 성능 평가를 수행한다. 마지막으로 5장에서 결론 및 향후 연구를 기술한다.

2. 관련연구

2.1 그리드 데이터베이스

인터넷이 보편화되고 컴퓨터 및 네트워크의 성능이 향상됨에 따라서 컴퓨터가 여러 분야에 이용되어지고 있다. 이에 대한 예로 대량의 유전자 정보 처리, 비행체나 발사체 설계, 기후나 환경변화, 고 에너지 물리분야의 데이터분석에 관한 연구이다. 이런 방대한 데이터들의 효율적인 성능향상을 위한 노력으로 분산 데이터베이스와 데이터베이스 클러스터에 노

력하여 왔다. 이제 또 다른 하나의 시도로 자원통합에 있어서 동일 기종 데이터베이스뿐만이 아니라 이기종 데이터베이스들과 대용량 데이터의 통합을 위해 그리드 데이터베이스가 연구 되어지고 있다.

분산 데이터베이스는 컴퓨터 통신망을 이용하여 여러 개의 지역 데이터베이스를 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[8].

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

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[9].

2.2 그리드 데이터베이스에서 복제본 갱신 전파 시 약한 일관성 유지를 위한 수요 기반 트리

수요기반 트리 각각의 그룹은 하나씩의 루트 노드로서 대표 노드를 갖는다. 대표 노드를 기준으로 자식 노드들은 트리 형태의 네트워크 구조로 연결된다. 수요란, 단일 노드에서의 특정 데이터에 대한 단위 시간 당 읽기 연산 요청 빈도로 정의되고 각 노드는

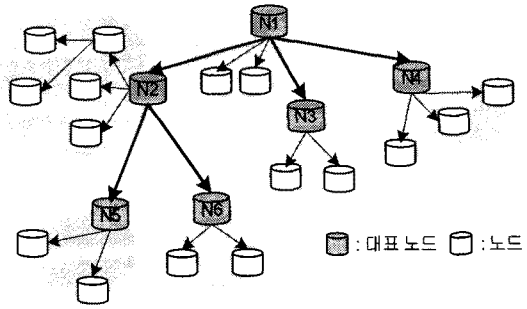


그림 1. 수요기반 트리의 다중 계층 구조

저장되는 모든 데이터에 대한 수요를 기록하고, 이 정보는 주기적으로 갱신된다.

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. 대표 노드는 대부분의 경우 데이터가 최초로 발생되고 저장되는 노드가 선택된다.

수요 기반 트리는 항상 완전 트리 형태를 유지한다. 모든 노드에서 트리의 높이가 항상 일정하고 한쪽으로 기울어지는 쏠림 문제(Skew)가 없어야 한다. 각각의 노드는 C개의 자식 노드를 가질 수 있다. 하나의 노드가 가질 수 있는 최대의 자식 노드수 C는 임의로 정의할 수 있으며, 완전 트리로 구성함으로 인해 최하위 노드를 제외한 모든 부모 노드는 C개의 자식 노드를 포함한다.

그림 1은 수요기반 트리의 기본 구조를 나타낸다. 지역적으로 서로 떨어져 있는 노드들에 현재 복제본이 저장되어 있으며, 각 노드의 복제본에 대한 수요를 기준으로 순서대로 계층 구조로 연결되어 있다.

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

3. 일관성 유지를 위한 버전 관리 기법

본 절에서는 일관성 유지를 위한 버전 관리 기법을 제안한다. 먼저 제안 기법을 위한 시스템의 환경을 설

명한 후, 버전 관리를 위한 데이터 구조를 설명한다. 그 후 일관성 유지를 위한 갱신 알고리즘을 살펴본다.

3.1 그리드 데이터베이스에서 일관성 유지를 위한 버전 관리 구조

그림 2은 그리드 데이터베이스 환경에서의 수요기반 트리를 사용하여 일관성을 유지하기 위한 버전 관리자가 확장된 구조이다.

모든 A는 복제본 데이터로 갱신된 순서에 따라 서로 다른 버전 정보를 가지고 있으며, 만일 모든 노드의 복제본 정보가 동일하게 갱신된 상태라면 모두 동일한 버전 정보를 포함한다.

그림 2는 버전 관리자가 포함된 형태의 수요기반 트리 구조를 나타낸다. 각 노드에 존재하는 모든 노드는 현재 버전 4로서 모든 노드의 복제본이 동일한 버전 값을 가지고 있다. 버전이 동일하면 복제본 데이터도 동일하다는 것을 뜻한다.

또한 이러한 복제본 간의 버전 정보 관리를 위해 임의의 노드에 버전 관리자를 두어 버전 정보의 갱신 및 전송을 담당하게 한다. 버전 관리자는 주요한 정보로서 최신의 버전 정보와 그 이전에 해당하는 버전 정보를 함께 관리한다. 또한, 작업 부하로 인해 각 노드에서의 갱신 연산 처리가 원활하게 진행되지 못하는 경우 일시적으로 갱신 연산을 중지시키거나 재시작시키는 제어 역할도 수행한다. 버전 관리자는 임의의 노드에서 동작될 수 있으며, 모든 동일 복제본에 대해 단 하나의 복제본 관리자만 운영된다.

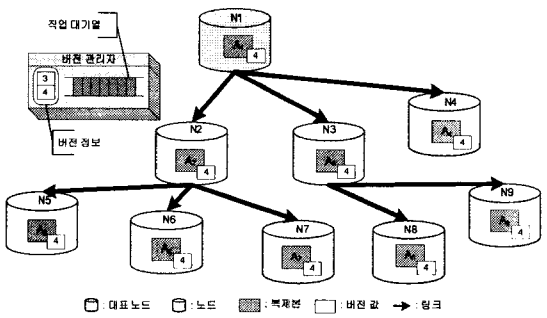


그림 2. 수요 기반 트리를 이용한 확장된 그리드 데이터베이스 구조

3.2 버전 관리를 위한 자료 구조

그림 3은 버전정보의 구조를 나타내는 그림이다.

버전은 이전버전과 다음버전 두 가지로 나뉜다. 이전버전은 다음버전 보다 한 단계 이전의 버전을 갖는다. 버전정보를 이렇게 두 가지 타입(TYPE)으로 저장하는 이유는 여러 노드에서 불규칙적으로 갱신 정보가 입력되는데 반하여 갱신은 순차적으로 실행되어야 하기 때문이다. 이전버전과 현재버전을 통하여 갱신이 순차적으로 일관성을 가지고 처리 되는 것을 보장한다.

우선순위 대기열은 노드가 가지고 있는 대기열로서 동시 다발적으로 발생하는 갱신 정보를 저장하고 있는 대기열이다. 노드는 대기열에 저장된 데이터의 버전정보를 확인하여 순차적으로 데이터를 가져와 갱신 연산을 수행한다. 대기열 안에 저장되는 데이터는 실제 갱신정보와 버전 정보가 저장된다. 노드에서는 우선순위 대기열 중에서 자신이 가진 버전의 다음 버전 정보를 순차적으로 수행한다. 현재 수행 중인 버전의 다음 버전 데이터가 없다면 다음 버전의 데이터를 받을 때까지 노드는 갱신 연산을 대기한다. 또한 노드에서는 대기열의 한계치 이상의 데이터가 쌓일 경우 더 이상의 갱신 정보를 받을 수 없음으로 갱신 정보 지연 메시지를 버전 관리자로 전달한다. 그림 4는 우선순위 대기열의 구조를 나타낸다.

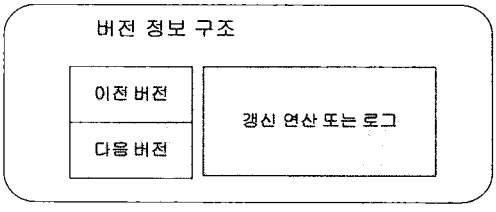


그림 3. 버전 정보 메시지 구조

그림 4. 우선순위 대기열 구조

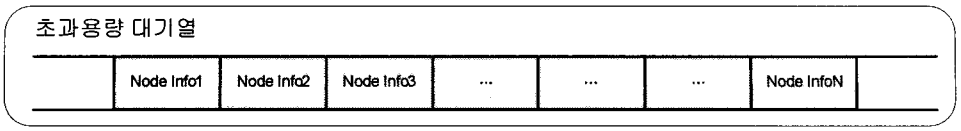


그림 5. 초과용량 대기열 구조

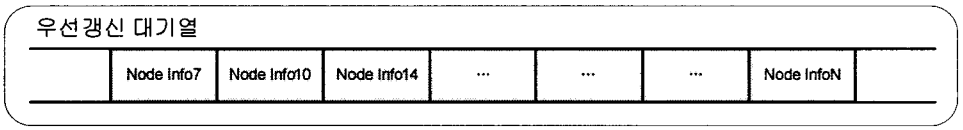


그림 6. 우선갱신 대기열 구조

초과용량 대기열은 노드에서 갱신 지연 메시지를 전달하였을 경우 갱신 연산이 지연된 노드의 정보를 저장하고 있는 버전 관리자의 대기열이다. 초과용량 대기열에서는 갱신 연산이 지연된 노드의 정보를 저장하며, 더 이상의 갱신 연산이 각각의 노드들로부터 발생되지 않게 함으로써 시스템 전체적으로 일관성을 유지한다. 또한, 초과용량 대기열에 지연된 노드의 정보가 삽입되면 우선 갱신 대기열이 동작되며, 이 후에 발생되는 갱신 연산 메시지는 우선 갱신 대기열에 저장된다. 그림 5는 초과용량 대기열을 나타내는 그림이다.

우선 갱신 대기열은 지연된 노드가 발생한 이후의 모든 갱신 연산 요청 메시지를 저장한다. 우선 갱신 대기열에서 갱신 연산 요청 메시지를 순차적으로 저장함으로써 초과용량 대기열에 있는 모든 노드가 지연이 해제 되었을 경우의 갱신 연산의 순차성을 보장하여 지연된 이후의 동시다발적으로 발생하는 갱신 연산의 일관성을 보장할 수 있다. 그림 6은 우선갱신 대기열을 나타내는 그림이다.

3.3 버전 관리자를 통한 일관성 유지 알고리즘

버전 관리자를 통한 갱신 연산의 과정이 임의의

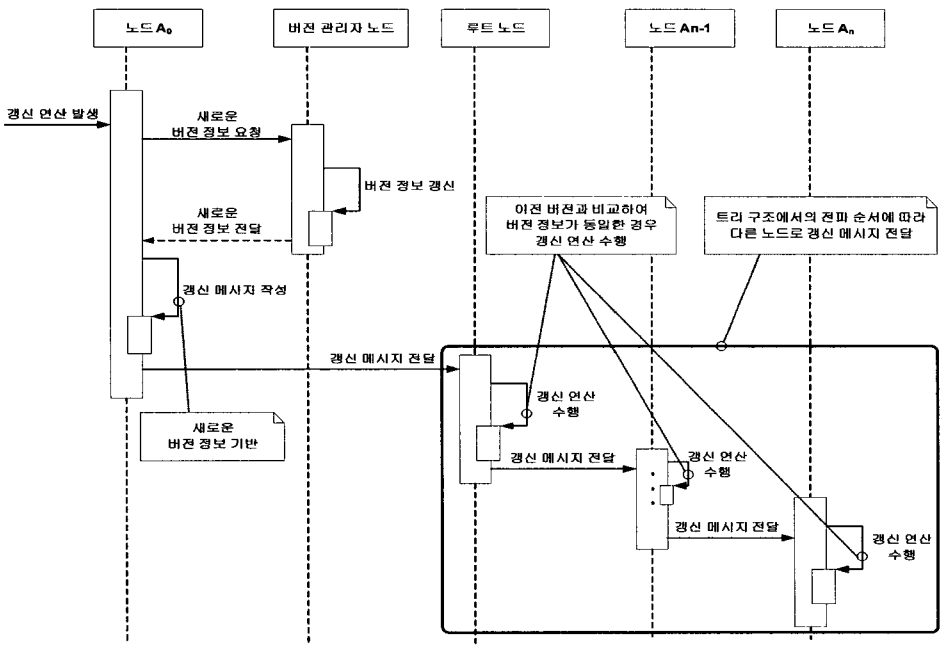


그림 7. UML을 이용한 업데이트 진행 과정

노드A에 갱신 연산 발생하였을 경우 먼저 버전 관리자가 있는 노드로 새로운 버전 정보를 요청한다. 그 후 버전 관리자는 현재 저장되어 있는 버전 정보를 갱신하여 새로운 버전을 노드A로 전달하며, 노드A는 전송 받은 버전 정보를 기반으로 갱신 메시지를 작성한다. 갱신 메시지가 작성 되었다면, 갱신 메시지를 트리 구조에서의 전파 순서에 따라 다른 노드로 전달하며, 갱신 메시지를 전달 받은 노드들은 갱신 메시지의 버전 정보에서 이전 버전과 현재 복제본의 버전 정보를 비교한다. 이때 버전 정보가 동일한 경우 갱신 연산을 수행하고 버전을 갱신 메시지의 다음 버전으로 교체한다. 만약 이전 버전과 상이한 버전일 경우 갱신 연산을 갱신 연산 대기열에 저장하고 복제본의 버전이 교체될 때까지 기다린다. 그림 7은 일관성 유지를 위한 진행 과정을 UML을 통하여 나타낸다.

그림 8은 버전 관리자를 통하여 갱신 연산이 처리되는 과정을 나타낸 그림이다.

그림 8은 노드 N3에서 갱신 연산이 발생한 경우에 대한 처리 순서로서 루트 노드로의 전달 과정까지 나타나 있다. 우선, N3은 버전 관리자에게 버전 정보에 대한 요청을 한다. 버전 관리자는 기존에 가지고 있던 버전 $\frac{3}{4}$를 갱신하여 $\frac{4}{5}$로 바꾼 후 갱신이 발생한 N3 노드로 바뀐 버전 정보를 전달한다. 버전 정보를 받은 노드 N3은 버전 정보를 포함하는 갱신 메시지를 생성한다. 갱신 메시지는 수요기반 트리에서의 갱신 전파 순서에 따라 우선 노드 N3로 전달된 후 루트 노드로 전달된다. 갱신 메시지를 전달 받은 노드들은 갱신 메시지의 이전 버전 정보와 현재 노드의 복제본의 버전 정보를 비교하여 동일한 경우 갱신

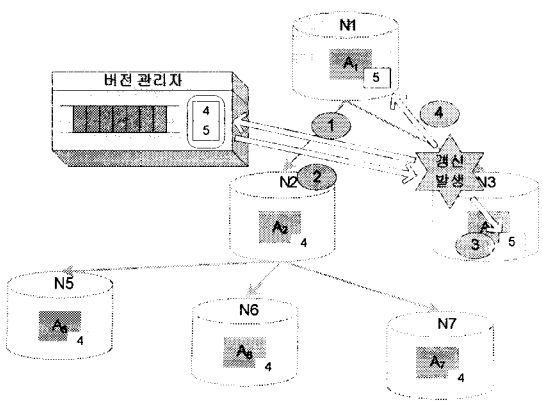


그림 8. 갱신 메시지 생성 및 전파 예

연산 처리를 한다. 버전 정보가 맞지 않는 경우에는 노드 내의 우선순위 대기열에 쌓여 있게 된다.

[알고리즘 1] 노드에서 갱신 연산이 발생하여 버전 관리자로 메시지를 전송하는 알고리즘 이다.

[알고리즘 1] 노드에서 갱신 연산을 위해 버전 관리자로 전송하는 메시지

```
Algorithm UpdateData( pData )
Input
   pData            :     갱신 요청 데이터
Output
   Success          :     정상적인 함수 실행 종료 메시지
   ERR_SEND_MSG     :     네트워크 오류 메시지
   ERR_GET_VERSION  :     버전 정보를 획득하지 못한 에러 메시지

Begin
01 : If(GetNewVersion(pData.ver) == FALSE) then
02 :    Return ErrorHandling(ERR_GET_VERSION);
03 : END if
04 : If(pData.state == UPDATE_WAIT) then
05 :    waitingUpdateList(pData);
06 : Else
07 :    If (SendtoMsg(pData,pData.parentNode) = FALSE ) then
08 :       Return ErrHandling(ERR_SEND_MSG);
09 :    END if
10 :    If (SendtoMsg(pData,pData.childNode) = FALSE ) then
11 :       Return ErrHandling(ERR_SEND_MSG);
12 :    END if
13 : END if
14 : Return Success
End
```

[알고리즘 1]은 갱신 연산이 발생하였다면 버전관리자에게 갱신할 데이터의 버전 정보갱신을 위한 메시지를 전달한다. 먼저 현재의 버전 정보를 버전 관리자에게 전달하여 다음 버전의 정보를 얻는다(01~02). 버전 관리자에게 다음 버전의 정보를 얻는다. 이때 버전 정보의 상태가 UPDATE_WAIT 라면 버전 관리자의 초과용량 대기열에 데이터 정보가 삽입되며 현재 데이터의 정보도 갱신 연산을 중지하고 기다린다(04~06). UPDATE_WAIT 메시지가 아니면 자신의 주변 노드로 갱신 메시지를 전달한다(07~12).

[알고리즘 2]는 버전관리자가 노드로부터 갱신 연산에 관련된 메시지를 받은 경우의 진행 과정을 나타낸다.

버전관리자는 계속적으로 메시지를 기다리고 있다. 만약 메시지가 들어오면 메시지를 분석하여 각각에 해당하는 메시지를 전달한다(01~35). 각각의 메시지는 다음과 같은 일을 한다.

UPDATEMSG 는 일반적인 갱신 메시지를 받는다. 일반적인 갱신 연산은 *WaitingFlag*를 확인하여

[알고리즘 2] 노드로부터 메시지를 받은 경우 버전관리자의 진행과정

```
Algorithm QueryProcessing()
Input
   Msg        :      노드로부터 전송된 데이터
Output
   ERR_SEND_MSG     :      네트워크 오류 메시지
   ERR_GET_VERSION :      버전 정보를 획득하지 못한 에러 메시지
Variables
   Result     :      펜딩 데이터 해제시에 사용되는 데이터 결과

Begin
01 : While()
02 : {
03 :   Msg := recivepacket();
04 :   Switch(Msg.type)
05 :   {
06 :    Case UPDATEMSG :
07 :         If (WaitingFlag == TRUE) then
08 :         {
09 :            StoredWaitingQ(Msg);
10 :            If (SendtoMsg(Msg,Msg.RealNode) = FALSE ) then
11 :               Return ErrHandling(ERR_SEND_MSG);
12 :            End if
13 :           Break;
14 :         }
15 :         exchangeVersion(Msg);
16 :         If (SendtoMsg(Msg,Msg.RealNode) = FALSE ) then
17 :            Return ErrHandling(ERR_SEND_MSG);
18 :         End if
19 :         Break;
20 :   Case PENDINGMSG :
21 :        InsertPendingNode(Msg);
22 :        WaitRuningFlag();
23 :        Break;
24 :   Case IMPENDING :
25 :         DeltPendingNode(Msg);
26 :         Result := GetPendingCount();
27 :        If (Result == EMPTY) then
28 :        {
29 :           If (SendtoMsg(Result,Result.RealNode) = FALSE ) then
30 :              Return ErrHandling(ERR_SEND_MSG);
31 :           End if
32 :        }
33 :       , Break;
34 :   }End switch
35 : }end while
End
```

*WaitingFlag*가 *FALSE*인 경우 버전의 메시지를 다음 버전으로 변경한다. 그리고 *VMSUCMSG* 메시지를 노드로 전송한다. 만약 *WaitingFlag*가 *TRUE*일 경우는 데이터는 *StoredWaitingQ()*함수를 통하여 우선순위대기열에 저장된다. 또한 *ExchangeVersion()*은 현재 저장된 버전정보를 증가하여 새로 들어온 버전 정보를 변경하는 일을 한다(6~19).

PENDINGMSG는 지연된 노드의 정보를 저장한다. 지연된 노드의 정보를 저장하고 *WaitRuningFlag()*함수를 호출하여 더 이상의 갱신 연산이 발생하지 않게 막는다. 차후 발생하는 갱신 메시지는 우선갱신 대기열에 저장된다(20~23).

IMPENDING는 지연된 노드로부터 지연 해제의 메시지가 전달된 것이다. 초과용량 대기열에 있는 지연되는 노드를 지연 해제 한 후 현재 대기열에 노드 정보가 존재하지 않는다면 현재까지 저장된 우선갱신 대기열의 데이터를 각각의 노드로 전달한다. 만약 초과용량 대기열에 아직 지연된 노드가 있다면 대기열의 노드가 지연 해제 될 때 대기한다(24~33).

[알고리즘 3]는 노드에서 갱신 관련 메시지를 받은 후의 처리 과정을 나타내는 알고리즘 이다.

노드는 데이터베이스의 일반적 역할을 수행하고 있으며 입력된 메시지를 각각의 경우에 따라서 구분하여 처리한다.

VMSUCMSG는 버전관리자에서 버전의 갱신 성공 메시지가 전송되었다는 뜻이다. 받아 온 데이터의 버전이 현재 저장된 버전보다 작거나 같다면 갱신 연산을 진행할 필요가 없다(09~12). 다음 버전이라면 *InsertFirstPQ()*함수에 저장된다(13~17). 우선순위 대기열이 초과된 상태가 아니라면 메시지를 *InsertPT()*에 삽입한다(18~20). 결과 값이 *FULLPQ*면 우선순위 대기열의 공간이 전부 사용되었기 때문에 버전 관리자에게 지연 메시지를 보낸다.

VMWAITMSG는 버전 관리자가 지연된 노드의 정보를 받은 경우임에도 불구하고 노드에서는 버전 관리자에게 갱신 메시지를 전송하여 더 이상의 갱신이 발생하지 않고 보류할 수 있도록 하기 위하여 버전 관리자가 노드에게 전송하는 메시지 이다. *WaitingProcess()*는 갱신 연산을 요청한 테이블의 버전과 정보를 저장하여, **VMREUPDATEMSG**의 메시지를 받을 때까지 갱신 연산을 지연 시킨다(28~30).

VMREUPDATEMSG는 버전관리자에서 지연이 해제된 후에 전송된 메시지이다. 노드에서 이 메시지를 받으면 버전관리자에서의 지연 때문에 갱신이 지연되었던 데이터를 갱신이 이루어 질 수 있도록 변경한다. 그 후 데이터 갱신 연산을 진행한다.

CHIFULMSG와 CHIEMPMSG 는 예외 처리를 위한 메시지로서 4장 예외처리에서 자세히 살펴본다.

〔알고리즘 3〕 갱신 관련 메시지를 받은 후의 처리과정

```
Algorithm UpdateProcing()
Input
   Msg          :      노드로부터 전송된 데이터
Output
   Success              :      정상적인 함수 실행 종료 메시지
   ERR_SEND_MSG         :      네트워크 오류 메시지
   ERR_GET_VERSION      :      버전 정보를 획득하지 못한 에러 메시지
Variables
   Result       :      펜딩 데이터 해제시에 사용되는 데이터 결과

Begin
01 : While()
02 : {
03 :   Msg := recivepacket();
04 :   Switch(Msg.type)
05 :   {
06 :    Case VMSUCMSG :
07 :         verNum := Versioncheck(Msg);
08 :         If ( verNum ≤ 0 ) then
09 :         {//현재 버전 정보보다 작거나 같으면 이미 갱신 연산이 이루어진 데이터
10 :            Break;
11 :         }
12 :         End if
13 :         If ( verNum == IncreVersion(Msg)) then
14 :         {
15 :             InsertFirstPQ(Msg);
16 :             Break;
17 :         }
18 :         End if
19 :         If (InsertPQ(Msg) == FULLPQ) then
20 :         {
21 :            Msg.state := PENDING;
22 :            If (SendtoMsg(Msg,VMSAddress) = FALSE ) then
23 :                Return ErrHandling(ERR_SEND_MSG);
24 :          If (SendtoMsg(Msg,CurNode.parent) = FALSE ) then
25 :            Return ErrHandling(ERR_SEND_MSG);
26 :         }
27 :        Break;
28 :   Case VMWAITMSG :
29 :         WaitingProcess(Msg);
30 :         Break;
31 :   Case VMREUPDATEMSG :
32 :        If (FindUpdateInfo(Msg) == TRUE) then
33 :        {
34 :           UpdateData(Msg);
35 :         }
36 :         Break;
37 :   Case NOTEMPMSG :
38 :        wantedUpdatedata(Msg);
39 :        sendblockPQ(Msg);
40 :        Break;
41 :   Case HASEMPMSG :
42 :        deltUnblockPQ(Msg);
43 :         Break;
44 :   }End switch
45 : }End while
End
```

4. 예외 처리 방법

본 장에서는 버전 관리자를 통한 일관성 유지 기법에서 발생할 수 있는 예외적인 상황과 예외적인 상황을 처리 하는 방법에 대하여 설명한다.

노드에서는 우선순위 대기열을 가지고 갱신 메시지를 저장하여 순차적으로 갱신 연산을 실행한다. 이럴 경우 메시지가 노드의 대기열 한계치를 초과하게 되는 경우가 발생 할 수 있다. 이런 경우는 버전관리자에게 지연 메시지를 보내서 더 이상의 갱신 연산이 발생이 발생하지 않게 보류할 수 있다. 하지만 현재까지 실행된 갱신 메시지를 노드에서 더 이상 받지 못하여 갱신 데이터를 유실할 수 있는 경우가 발생할 수 있다.

또한 다음 버전의 갱신 데이터 버전을 기다리는 상태에서 다음 버전의 데이터는 삽입되지 않고 차후의 버전들이 먼저 삽입되어 우선순위 대기열의 저장 공간 한계치를 초과하는 상태가 발생할 수 있다. 대기열의 한계치를 초과한 경우 더 이상의 데이터를 받을 수 없고 현재까지 처리된 데이터의 다음 갱신 버전이 대기열에 없어서 처리를 할 수도 없는 데드락(Deadlock)현상이 발생할 수 있다.

위와 같은 예외적인 경우를 방지하기 위하여 다음과 같은 2가지의 메시지를 사용한다. 첫째는 **NOTEMPMSG**의 메시지로써 하위 노드의 우선순위 대기열에 더 이상의 가용공간이 없을 경우 상위 노드로 전송되는 메시지이다. 둘째로, **HASEMPMSG**의 메시지로 하위 노드에서 우선순위 대기열에 사용할 수 있는 공간이 발생하면 전송하는 메시지이다.

노드에서 더 이상의 우선순위 대기열의 가용공간이 없을 경우 CHIFULMSG 메시지와 자신이 현재 기다리고 있는 버전의 정보를 상위 노드로 전송한다. 상위 노드로 메시지를 전송하는 이유는 노드에서 갱신 데이터를 더 이상 대기열에 저장할 수 있는 공간이 부족하기 때문에 상위 노드에서 메시지를 전달하지 않고 하위 노드에 전달할 메시지를 보관하고 있는 것이다. 차후 하위 노드에서 우순 순위 대기열의 빈 공간이 발생하였다는 메시지를 전달하면 상위노드는 자신의 노드에 저장되었던 갱신 메시지를 전달한다.

자신의 노드에서 **HASEMPMSG**의 메시지를 받았을 경우, *WantedUpdatedata()*는 자신의 하위 노드에서 필요로 하는 데이터의 버전 정보를 보내주는 역할을 하는 함수 이다. 이 함수를 사용함으로써 하위 노드에서의 우선순위 대기열의 데드락 현상을 방지한다.

또한, *sendblockPQ()*는 우선순위 대기열의 데이터를 하위 노드로 전송하는 것을 막는 역할과 현재

노드에서 갱신 연산이 수행된 데이터에 관해서도 갱신 메시지의 삭제를 지연하는 역할을 한다. 이것은 하위 노드에서 더 이상의 갱신 데이터를 저장할 수 없기 때문에 상위 노드에서 데이터의 삭제와 전송을 지연하여 하위 노드에서 데이터를 유실하지 않게 보장한다. 또한 자신의 노드도 하위 노드로 갱신 데이터를 전송하지 못하여 데이터가 쌓여서 우선순위 대기열이 저장할 수 있는 한계치를 초과할 경우 자신의 상위 노드로 **NOTEMPMSG** 메시지를 전달한다.

만약 하위 노드에서 **HASEMPMSG**의 메시지를 상위 노드에 보내면 갱신 데이터를 전송하고 수행된 전달된 데이터는 삭제한다.

5. 성능평가

본 장에서는 제안 기법의 평가를 위한 실험 환경에 대해 설명하고, 복제본 수에 따른 갱신 완료 시간과 접근 빈도의 변화에 대한 갱신 등의 환경 변화에 의한 제안 기법과 기존 기법과의 성능 비교를 수행한다.

5.1 실험 환경

본 논문에서는 제안 기법의 평가를 위해 C/C++언어 기반 시뮬레이션 툴인 CSIM를 사용하였다[14]. CSIM는 분산 환경에서의 컴퓨팅 모델, 알고리즘 평가 등 대부분의 시뮬레이션이 가능한 툴이다.

제안 기법의 구현을 위해 다수의 클라이언트와 서버를 구성하였으며, 모든 서버에 제안하는 대기열 자료구조를 저장하였다. 구현을 위하여 MS Visual C++ 6.0을 사용하였다. 전체적인 구성은 그리드 환경에 적합하도록 복제본의 사용을 허용 하였으며, 이질적인 분산 환경에서의 성능 평가를 위하여 여러 노드에서의 각기 다른 대기열 사이즈와 질의 처리 속도 등의 범위를 두어서 평가 하였다.

표 1는 갱신 데이터 처리 성능 평가를 위한 실험 환경이며, 각 노드의 저장소 크기, 로그에 대한 대기열 크기 등의 정보는 실제 실험평가에서 대기열의 길이에 따른 결과값의 변화가 미미하여 이에 대한 평가는 생략하였다.

실험을 위해 제안 기법을 DAF-tree라 부르며 기존의 빠른 일관성(Fast-Consistency: FC) 유지 기법에 대한 실험 평가를 수행한다. 비교 대상으로서 빠른 FC 기법과 이 기법을 확장하여 대표 노드에 대한 개념을 포함시킨 FCL(Fast-Consistency with Leader) 기법과 ARCS(Adaptive Replica Consistency Service)를 사용한다. 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.

실험 평가의 대상이 되는 인자로는 임의의 갱신 연산에 대해 모든 노드가 갱신 완료될 때 까지 걸리는 시간과 접근 빈도의 변화 비율에 따른 단위 시간 당 갱신 반영 정보의 검색 비율 및 네트워크 사용 비율이 있다. 단위 시간 당 정보의 검색 비율은 해당 값이 높을수록 접근 빈도가 동적으로 변화하는 환경에서의 갱신 정보 반영 비율이 높다는 것을 의미한다.

5.2 실험 평가

본 실험 평가에서는 복제본 수에 따른 갱신 완료 시간, 접근 빈도의 변화에 따른 갱신 데이터가 반영된 정보에 대한 검색 비율, 접근 빈도의 변화에 따른 네트워크 사용량을 주요 평가 인자로 보고 이것을 집중적으로 평가하였다.

그림 9은 복제본 수에 따라 갱신 연산이 완료되는 시간을 측정한 결과 그래프이다. DAF-tree의 경우 FC와 FCL 기법에 비해 빠른 완료 시간을 보인다. 이는 FC와 FCL에 비해 DAF-tree에서의 갱신 연산이 적은 메시지 교환 방식을 사용하기 때문이다. DAF-tree에서는 이미 접근 빈도에 따른 우선 순위

표 1. 실험 환경

실험 요소	실험 데이터	실험 요소	실험 데이터
실험 시간	30000 unit	갱신 연산 처리 시간	4~7 unit
서버 수	20~400 개	갱신 정보 전송 시간	2~4 unit
노드 당 테이블 수	0~300 개	갱신 질의 입력 간격	0.1 unit
접근 빈도 변화 비율	10~12unit		

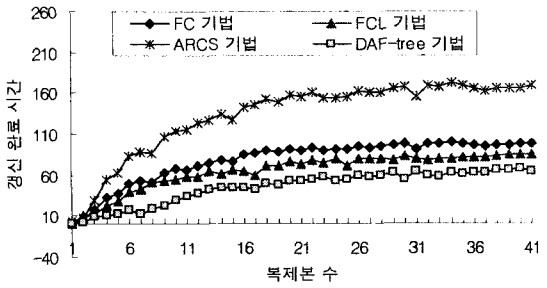


그림 9. 복제본 수에 따른 갱신 완료 시간

가 결정되어 있기 때문에 갱신 메시지가 별 다른 과정 없이 순차적으로 전달이 된다. 그러나 FC와 FCL 기법의 경우 갱신 메시지를 전달하기 전에 우선 순위 비교를 위한 메시지 교환을 선행하게 된다. 이로 인해 다음 그림 9과 같은 성능의 차이가 발생하게 된다.

다음은 각 노드의 접근 빈도 값을 지속적으로 변화시켰을 때의 단위 시간 당 갱신 반영 정보의 검색 비율 및 네트워크 사용 비율에 대해 실험한다. 각 노드의 접근 빈도 변화는 초기 접근 빈도 값 대비 변경 비율로서 산출되며, 비율이 클수록 DAF-tree에서의 노드 간의 위상 변화가 심해진다.

그림 10은 접근 빈도의 변화에 따른 갱신 데이터가 반영된 정보에 대한 검색 비율을 나타낸 그래프이다. 단위 시간 당 갱신된 정보를 검색해 가는 비율이 접근 빈도의 변화에 따라 크게 떨어지는 것을 볼 수 있다. 또한 기존 기법의 감소 비율이 DAF-tree 기법에 비해 더 큰 것은 기존의 기법이 네트워크의 부하로 인해 노드 간의 접근 빈도 값 비교가 원활하지 않기 때문이다.

그림 11는 접근 빈도의 변화에 따른 네트워크 사용량을 패킷 단위로 나타낸 그래프이다. 제안 기법에 비해 네트워크 사용량이 크게 증가함을 알 수 있다.

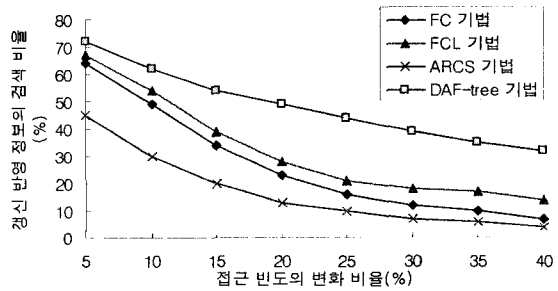


그림 10. 접근 빈도의 변화에 따른 갱신 데이터가 반영된 정보에 대한 검색 비율

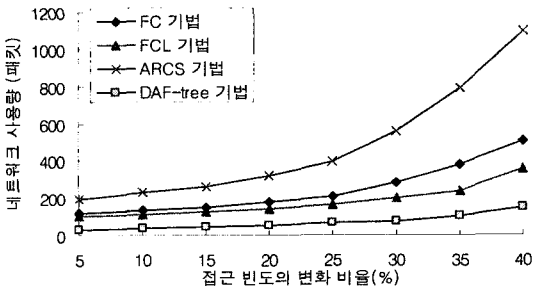


그림 11. 접근 빈도의 변화에 따른 네트워크 사용량

또한, DAF-tree를 구성하는 각 노드의 임계값(threshold), 즉 하위 노드의 개수를 4개에서 20개까지 늘려가며 평가를 하였으며, 그 외에 네트워크 속도와 각 노드의 처리 속도에 변화를 주었을 때 전체적인 성능의 변화는 있으나 FC 및 FCL 기법에 비해 항상 좋은 성능을 나타내었다. 그러나, 최적의 성능을 위한 DAF-tree의 임계값은 환경에 따라 다르게 적용될 수 있다.

6. 결 론

본 연구는 수많은 노드에서 복제본 데이터를 허용하는 그리드 데이터베이스에서 일관성 유지를 위한 버전 관리 기법을 제안하고, 제안 기법과 다른 기법과의 성능 평가를 수행하였다. 제안 기법은 복제본 데이터의 동시 다발적인 갱신 연산으로 인한 일관성의 유지를 위하여 그리드 데이터베이스에서 버전 관리자라는 프로세서를 사용함으로써 캐쉬를 관리한다.

버전 관리자는 복제본의 데이터 일관성을 유지하기 위하여 버전 정보에 대한 구조체를 가지며, 버전 정보는 이전 버전 값과 다음 버전 값의 구조로 구성된다. 이 두 가지 값을 통하여 복제본의 일관성을 유지한다. 또한 버전 관리자에서 관리되는 작업 대기열은 동시 다발적으로 발생하는 갱신 연산에 처리 과정에서 일부 노드에서의 작업 부하로 갱신 연산의 지연이 발생할 경우 갱신 연산이 발생되는 노드의 위치 정보를 기억한다. 향후 작업 부하가 완화되면 대기열에 저장되어 있는 정보를 통해 다시 갱신 연산을 처리함으로써 동시 다발적으로 발생하는 데이터의 일관성을 보장한다. 성능 평가에서는 제안 기법이 기존의 기법 보다 5~25% 정도 향상됨을 보였다. 이것은 버전 관리자를 통하여 일관성을 유지함으로써 기존

의 기법에 비하여 향상됨을 보인다.

제안 기법은 노드의 수가 많고, 복제본 데이터를 가지는 그리드 데이터베이스 환경이나 분산 데이터베이스 환경에서 유용하게 사용될 수 있다.

본 연구에서는 고정적인 네트워크 구성을 통한 연구 결과로써 평가되었으나, 향후 연구로는 그리드 환경에 부합되는 유동적인 환경에서의 네트워크 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] J. A. Elias and L. N. 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.

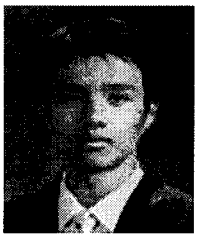
[6] J. A. Elias and L. N. Moldes, "Generalization of The Fast Consistency Algorithm To a Grid With Multiple High Demand Zones," *Computational Science, ICCS*, pp. 275-284, 2003.

[7] R. Ge, Y. I. Jang, S. Y. Park, and H. Y. Bae, "Replica Update Propagation Using Demand-Based Tree for Weak Consistency in the Grid Database," *Journal of Korea Multimedia Society*, Vol.9, No.12, pp. 1542- 1551, 2006.

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

[9] 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.

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



신 숭 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

장 용 일

1997년 인하대학교 전자계산공학과 (공학사)
2001년 인하대학교 컴퓨터공학부 (공학석사)
2007년 인하대학교 컴퓨터정보공학과 (공학박사)
2008년~현재 텔코웨어 DB솔루션팀

관심분야 : 웹 & 모바일 GIS, 데이터베이스 클러스터, 위치기반서비스, 그리드 데이터베이스

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

어 상 훈

2003년 인하대학교 전자계산공학과 (공학사)
2003년~현재 인하대학교 정보공학과 통합박사과정
관심분야 : 데이터 스트림 관리 시스템, 데이터베이스 클러스터, 상황인지 서비스

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