Journal of Korea Multimedia Society Vol. 10, No. 1, January 2007(pp. 13-25)

그리드 데이터베이스에서 질의 전달 최적화를 위한 캐쉬 관리 기법

신숭선[†], 장용일[††], 이순조[†††], 배해영[††††]

요 약

그리드 데이터베이스에서는 질의 전달 최적화를 위해 캐쉬를 사용한다. 캐쉬에 빈번히 사용되는 데이터의 메타 정보를 메타 데이터베이스에서 가져와 캐싱하며, 캐싱된 정보를 통하여 질의 전달의 비용을 감소시킨다. 기존의 캐쉬 관리 기법은 질의 전달 시 복제본의 사용빈도를 고려하지 않은 데이터의 임의의 메타 정보를 캐싱하기 때문에 사용이 불균형적인 문제가 있다. 그리고, 원본 데이터가 변경 되었을 경우에 기존의 메타 정보를 가진 캐쉬를 통하여 질의가 타 노드로 잘못 전달되며 이러한 과정은 여러 노드에서 반복 수행되어 네트워크 비용을 증가시킨다. 따라서 기존의 캐쉬 관리 기법은 복제본의 사용비율 불균형과 타 노드로의 잘못된 질의 전달로 인한 네트워크 비용 증가 문제의 해결이 필요하다. 본 논문에서는 질의 전달 최적화를 위한 캐쉬 관리 기법을 제안한다. 제안 기법은 캐쉬 매니저라는 관리 프로세서를 사용하여 캐쉬를 관리한다. 캐쉬 매니저는 자주 사용되는 복제본이 저장된 노드의 사용빈도를 비교하여 적게 사용된 노드의 복제본 메타 정보를 캐싱함으로써 질의 전달을 최적화한다. 또한 캐쉬 매니저를 통해 타 노드로 잘못 전달되는 질의를 줄여 질의 처리 시간을 단축하고 네트워크 비용을 줄인다. 제안 기법은 성능평가를 통해 네트워크 비용과 처리시간이 감소되어 기존의 방식에 비하여 향상된 성능을 보인다.

Cache Management Method for Query Forwarding Optimization in the Grid Database

Soong-Sun Shin[†], Yong-Il Jang[††], Soon-Jo Lee[†††], Hae-Young Bae[††††]

ABSTRACT

A cache is used for optimization of query forwarding in the Grid database. To decrease network transmission cost, frequently used data is cached from meta database. Existing cache management method has a unbalanced resource problem, because it doesn't manage replicated data in each node. Also, it increases network cost by cache misses. In the case of data modification, if cache is not updated, queries can be transferred to wrong nodes and it can be occurred others nodes which have same cache. Therefore, it is necessary to solve the problems of existing method that are using unbalanced resource of replica and increasing network cost by cache misses. In this paper, cache management method for query forwarding optimization is proposed. The proposed method manages caches through cache manager. To optimize query forwarding, the cache manager makes caching data from lower loaded replicated node. The query processing cost and the network cost will decrease for the reducing of wrong query forwarding. The performance evaluation shows that proposed method performs better than the existing method.

Key words: Grid Database(그리드 데이터베이스), Cache Management(캐쉬 관리), Query Forwarding (질의 전달), Data Replication(복제본)

※ 교신저자(Corresponding Author) : 신숭선, 주소 : 인천광역시 남구 용현동 253 (402-751), 전화 : 032)860-8712, FAX : 032)862-9845, E-mail : hermit@dblab.inha.ac.kr
접수일 : 2006년 9월 4일, 완료일 : 2006년 11월 27일
[†] 준회원, 인하대학교 컴퓨터정보공학과 석사과정
[††] 준회원, 인하대학교 컴퓨터정보공학과 박사과정
(E-mail : yijang@dblab.inha.ac.kr)
[†††] 서원대학교 컴퓨터공학과 부교수
(E-mail : sjlee@seowon.ac.kr)
[††††] 종신회원, 인하대학교 대학원 원장
(E-mail : hybae@inha.ac.kr)
※본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구 센터 육성·지원사업의 연구결과로 수행되었음

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 공유하며, 고속 연산을 가능하게 한다. 그리드 데이터베이스를 구성하는 각 노드(여기서, 노드란 데이터베이스 관리 시스템으로 구성되며, 또한 클러스터 시스템으로 구성되어진 데이터베이스 관리 시스템이다.)는 데이터의 처리 성능과 가용성 향상을 위해 서로 다른 위치에 데이터를 복제하여 저장하며, 데이터 연합(federation), 데이터 변환(conversion), 데이터 배치(allocation), 데이터 통합(integration) 등의 기능을 제공한다[1].

수많은 노드와 데이터를 갖는 그리드 데이터베이스에서는 질의 전달 최적화를 위해 캐쉬(cache)를 사용한다. 캐쉬에서는 자주 사용되는 메타 정보를 캐싱(caching)하여 인터넷의 성능을 향상시키며, 서버의 부하와 전체 네트워크의 트래픽을 감소시키고, 사용자에게 캐싱된 정보를 통한 질의 전달로 빠른 응답시간을 제공한다.

기존의 캐쉬 관리 기법은 메타 데이터베이스에서 임의의 메타 정보를 가져와 캐쉬에 저장함으로써 복제본 데이터의 사용빈도를 고려할 수 없었다. 복제본 데이터의 사용빈도를 고려하지 못하는 메타 정보의 사용으로 데이터의 사용이 불균형적이여서 시스템의 성능 저하가 나타날 수 있다. 그리고 여러 노드에서 복제본 데이터의 메타 정보를 저장하고 있을 때, 원본 데이터가 변경 되었을 경우 변경되지 않은 메타 정보를 가진 캐쉬를 통하여 질의가 타 노드로 잘못 전달될 수 있다. 이런 캐쉬의 비일관성(inconsistency)으로 인하여 여러 노드에서 질의가 잘못 전달되어 네트워크 비용을 증가하며, 처리시간의 증가와 시스템의 성능저하를 가져올 수 있다.

제안 기법은 기존의 캐쉬 관리 기법의 데이터 사용비율 불균형과 캐쉬의 비일관성에 의한 잘못된 질의 전달 문제를 해결하기 위해서 캐쉬 매니저라는 관리 프로세서를 사용하여 노드들의 캐쉬를 관리한다. 캐쉬 매니저는 노드들의 효율적인 캐쉬 관리를 위하여 임의의 노드들을 그룹으로 묶은 후 캐쉬 테이블에 노드들의 메타정보를 저장함으로써 노드들의 캐쉬를 효율적으로 관리한다.

노드에서 캐쉬의 변경이 필요한 경우 캐쉬 매니저로 이벤트를 전송한다. 이벤트를 전송 받은 캐쉬 매니저는 메타 데이터베이스를 통하여 복제본의 메타 정보를 가져온다. 복제본의 메타 정보를 통해 자주 사용되는 복제본을 저장하고 있는 노드의 사용빈도를 비교하여 적게 사용되는 노드의 복제본 메타 정보를 캐싱함으로써 질의 전달을 최적화한다. 또한 캐싱된 메타 정보를 통해 타 노드로 질의 전달 시에 캐쉬의 비일관성으로 인한 잘못된 질의 전달을 줄이기 위하여 캐쉬 매니저는 캐쉬 테이블을 이용하여 캐싱된 메타 정보의 일관성을 보장한다.

제안 기법은 성능평가에서 네트워크 비용과 처리시간의 절약으로 기존의 방식에 비하여 향상된 성능을 보인다.

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

2. 관련 연구

본 장에서는 그리드 데이터베이스의 기능 및 요구사항들과 이에 따른 기존 연구들을 설명하고, 기존의 캐쉬 관리 기법을 설명한다.

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

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[2].

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 균형 있게 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[3].

2.2 캐쉬 관리 기법

캐쉬는 여러 용도로 사용되며, 대부분의 캐쉬는 캐쉬 적중률을 높이기 위한 캐쉬 교체 전략과 캐쉬 데이터의 일관성을 유지하기 위한 알고리즘에 대한 것으로 구분되고 있다.

캐쉬의 적중률을 높이기 위한 공통적인 관심사는 한정된 저장 영역의 효율적인 운용이다. 일반적으로 사용 빈도가 높은 객체를 캐쉬의 저장영역에 저장해야 캐쉬의 성능을 높일 수 있다. 따라서 효율적인 교체 기법은 캐쉬의 성능을 향상시키는 중요한 요인이다[4].

현재까지 FIFO(First In First Out), LRU(Least Recently Uesd), LFU(Least Frequently Used), LUV, SIZE, LRUMIN과 같은 여러 교체 기법들이 연구되어 왔다. LRU는 새로운 객체가 저장 될 공간을 마련하기 위하여 가장 최근에 사용되지 않은 객체를 저장 영역에서 교체하는 기법이다. LRU는 최근에 오랫동안 사용되지 않은 객체가 미래에 참조되지 않을 가능성이 가장 높은 개체라는 특성을 이용하는 방식이다. LFU는 참조의 빈도수를 기반으로 자주 사용되지 않는 개체를 우선적으로 교체하는 방식이다. SIZE는 새로운 객체가 저장 될 공간을 마련하기 위하여, 저장 영역에 저장된 객체 중에서 가장 큰 객체를 교체하는 기법이다[5,6].

캐쉬 데이터의 일관성 유지를 위한 기법은 서버로 보내지는 데이터 요구를 줄이기 위하여 DBMS는 클라이언트의 캐쉬를 사용할 수 있다. 이러한 캐싱된 정보는 여러 클라이언트 사이트에서 사용되기 때문에 모든 클라이언트는 일관성 있는 값을 보장해 주어야 한다. 캐쉬의 일관성 유지 기법은 클라이언트의 트랜잭션이 접근하는 캐시된 페이지 복사본의 유효성(validity)에 따라 크게 탐지기반(detection based) 프로토콜 및 회피기반(avoidance based)프로토콜로 분류될 수 있다[7-10].

탐지기반 기법과 회피기반 기법은 최신의 값이 아닌(stale) 데이터에 트랜잭션이 접근하지 못하도록 하는 방법에 따라 구별된다. 탐지기반 프로토콜은 클라이언트 캐쉬에 있는 최신의 값이 아닌 데이터 복사본의 존재를 허용한다. 따라서 최종 완료를 하기 전에 접근한 캐싱된 데이터의 유효성을 검사해야 한다. 서버는 클라이언트가 유효성 검사를 할 수 있는 최신의 정보를 유지해야 한다. 회피기반 프로토콜은 최신이 아닌 데이터를 결코 접근하지 않게 한다. 이것은 중복 관리 기법인 ROWA(read-one write-all)에 기반하기 때문이다. 따라서 회피기반 프로토콜에서는 트랜잭션이 완료됐을 때 존재하는 모든 갱신된 복사본들이 같은 값을 갖도록 보장할 수 있다.

3. 질의 전달 최적화를 위한 캐쉬 관리 기법

본 장에서는 논문에서 제안하는 그리드 데이터베이스에서 질의 전달 최적화를 위한 구조와 캐쉬 매니저, 캐쉬 구조를 살펴보고 질의 전달 최적화를 위한

캐쉬 관리 기법에 대하여 설명한다.

3.1 그리드 데이터베이스에서 질의 전달 최적화를 위한 캐쉬 관리 구조

(그림 1)은 질의 전달 최적화를 보여주기 위한 그리드 데이터베이스의 전체적인 구조가 나타나 있다. 각각의 노드들은 캐쉬 테이블을 저장하며, 메타 데이터베이스를 통하여 메타 정보를 얻는다. N_1, N_2, N_3,.....,N_N 은 각각의 노드를 나타내며 서로 네트워크를 통하여 연결된다. 또한 클라이언트는 노드들에게 질의를 전송하며, 노드에서는 자신이 처리할 수 없는 질의에 대해서 다른 노드로 질의를 전달한다.

캐쉬 매니저는 임의의 노드들 중에서 데이터베이스 관리자를 통해 실행되는 하나의 프로세서이며, 각각의 노드들의 캐쉬 정보를 리스트(list)로 만들어 캐쉬 테이블로 관리하며 여러 가지 색인 기법을 사용한다.

캐쉬 매니저는 방대한 양의 캐싱된 정보를 관리하기 위하여 데이터베이스 관리자를 통하여 처리 용량에 따라 노드를 그룹으로 나누어 관리하며, 자신이 실행된 노드를 기준으로 인접해 있는 임의의 노드들을 그룹으로 연결한다. 그룹의 크기는 동적으로 변화되는데 이것은 그리드 데이터베이스 환경이 유동적으로 변화하기 때문이다.

캐쉬 매니저는 각각의 노드에서 캐쉬의 변경이 필요한 경우 발생하는 메시지를 이벤트(삽입, 변경)라고 정의한다. 노드에서 이벤트가 발생할 경우 캐쉬 매니저를 가진 노드로 전송하여 이벤트를 처리한다. 이벤트의 발생과 처리 과정은 3.2절에서 자세히 살펴본다.

캐쉬 매니저는 이벤트가 발생한 경우 메타 데이터베이스를 통하여 메타 정보를 받는다. 메타 정보를 받은 후 캐쉬테이블을 재구성 하고 이벤트를 요청한 노드에 갱신된 캐쉬 정보를 전송한다.

(그림 2)은 캐쉬의 전체 구조이며 헤더와 매핑테이블 구조로 이루어진다.

(그림 2)에서 Group ID와 Cache Manager Address는 헤더에 포함되는 자료구조이다. Group ID는 캐쉬 매니저를 구분하기 위한 그룹 번호이고, Cache Manager Address는 캐쉬 매니저가 있는 노드의 주소이다. 이 구조를 통하여 노드에서 이벤트 요청 시에 자신이 속한 매니저로 메시지를 전송한다.

매핑테이블의 한 레코드는 NAME과 ADDRESS로 구성되며 데이터 이름과 주소를 나타낸다. 이 구조를 통하여 요청 질의에 대한 데이터를 찾고 원하는 데이터가 있는 주소로 질의를 전달한다.

3.2 질의 전달 최적화를 위한 캐쉬 관리 기법

질의 전달 최적화를 위한 캐쉬 관리는 일정한 공

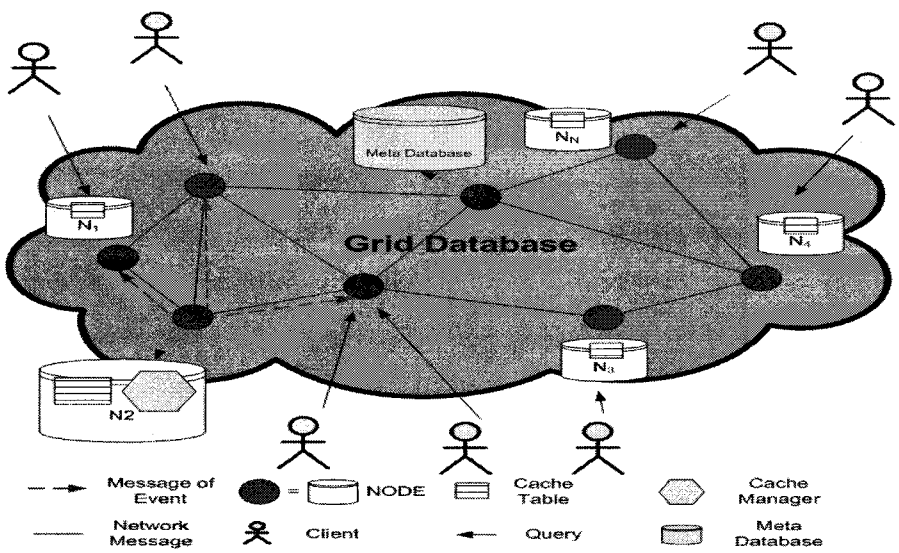


그림 1. 그리드 데이터베이스에서 질의 전달 최적화를 위한 캐쉬 관리 구조

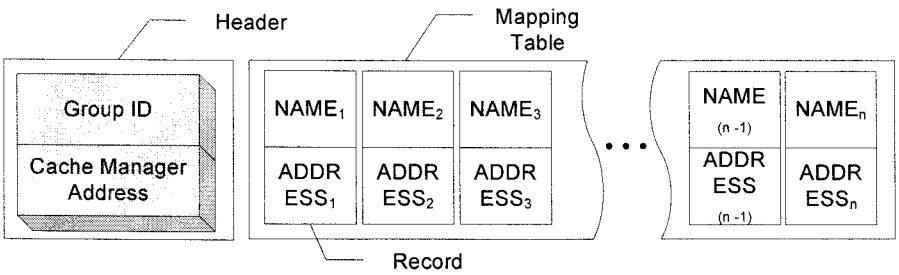


그림 2. Cache structure

간에서의 메타 정보의 삽입과 교체로 나타난다. 노드에서 빈번히 사용되는 메타 정보의 삽입과 변경을 위해 메시지가 발생하는 경우를 이벤트 발생이라고 정의한다. 이벤트의 발생과 처리과정을 설명함으로써 캐쉬가 관리되는 과정에 대하여 살펴본다.

3.2.1 이벤트 발생과정

이벤트는 삽입과 변경의 두 가지의 경우를 갖는다. 삽입 이벤트는 빈번히 사용되는 데이터의 메타 정보를 캐쉬에 저장하기 위해 발생하는 메시지이다. 삽입 이벤트는 자신의 노드를 모니터링 하여 사용자 요청으로 빈번히 사용되는 데이터에 접근하는 질의 요청 시 나타난다. 삽입 이벤트가 발생하면 캐쉬 매니저로 전송하며, 또한 자주 사용되는 데이터의 정보를 얻기 위하여 사용빈도를 저장한다.

(알고리즘 1)은 삽입 이벤트의 발생과정을 나타낸다.

사용 요청 데이터가 로컬 메타 정보에 있는지 검사한다(01라인). 캐쉬에 요청 데이터의 메타 정보가 있는지 검사한다(02라인). 메타 데이터베이스에서 메타 정보를 얻어와 질의를 전달한다(03~04라인). 이때 *GetMetaListFromMDB()*는 메타 데이터베이스에 접속하여 메타 정보를 가져오는 함수이다(03라인). 빈번히 사용되는 데이터인지 검사하여 자주 사용되는 데이터라면 캐쉬 매니저로 이벤트를 전송한다(05~06라인). 캐쉬에 메타 정보가 저장되어 있다면 메타 정보를 통해 질의를 전달한다(09라인).

```
Input
Request data : 요청 데이타
Ouput
Void
Begin
01 :  if (Request data is not founded in local meta data) then
02: if (Request data is not found in cache) then
03 :     GetMetaListFromMDB()
04 :     Send request query to destination node
05 :    if (Request data is not frequently used data) then
06 :     Send INPUT event message to Cache manager
07 :    endif
08 :   else
09 :    Send request query to destination node
10 :   endif
11 :  else
12 :   process user's request query
13 :  endif
End
```

알고리즘 1. 삽입 이벤트 발생 알고리즘

```
Input
result : 질의 결과
Ouput
Void
Begin
01 : if(result is query forwarding error)then
02 :  if ( Error is Cache misses ) then
03 :   Send UPDATE event message to Cache manager
04 :   GetMetaListFromMDB()
05 :   Send request query to destination node
06 :   Caching metadata of frequently data
07 :  endif
08 : endif
09 : Exception Handling
End
```

알고리즘 2. 변경 이벤트 발생 알고리즘

(알고리즘 2)은 변경 이벤트의 발생과정을 나타낸다. 변경 이벤트는 원본 데이터가 변경되어 메타 정보의 변경이 필요한 경우 나타나는 메시지이다. 캐쉬의 변경을 위해서 변경 이벤트가 발생되면 캐쉬 매니저로 이벤트를 전송한다.

캐싱된 메타 정보를 통한 잘못된 질의 전달로 질의 처리 오류 메시지를 받은 경우에 나타나는 알고리즘이다. 결과 값이 질의 포워딩 오류인 경우인가 검사한다(01라인). 질의 포워딩 오류라면 질의가 타 노드로 잘못 전달되어진 오류인가 검사한다(02라인). 잘못 전달되어진 오류라면 캐쉬의 변경을 위하여 캐쉬 매니저로 변경 이벤트를 전송한다(03라인). 최적의 메타 정보를 얻기 위해서는 캐쉬 매니저에서 처리 과정이 필요 하므로 임시 메타 정보를 얻어와 질의를 전달한다(04~05라인). 그리고, 메타 정보를 캐싱한다(06라인).

3.2.2 이벤트 처리과정

노드에서 전송한 삽입과 변경 이벤트를 캐쉬 매니저에서 처리 하는 과정에 대하여 살펴본다.

■ 삽입 이벤트

삽입 이벤트를 전송받은 경우 캐쉬 매니저는 이벤트를 분석한다. 이 과정을 통해 분석된 정보를 가지고 메타 데이터베이스를 통하여 복제본의 메타 정보의 리스트를 얻는다. 그리고, 메타 정보의 리스트에 있는 각각의 노드에 접속 하여 노드의 사용비율을 얻어 온 후 가장 사용비율이 적은 노드의 주소를 캐쉬테이블에 저장한다. 마지막으로 이벤트를 발생한 노드로 갱신된 캐쉬 정보를 전송한다.

(그림 3)는 캐쉬 매니저에서 삽입 이벤트를 처리하는 과정을 나타내는 그림이다.

(그림 3)에서 N_2와 N_4에는 원본 데이터인 D데이터를 가지며, N_1에는 D데이터에 대한 메타 정보가 캐싱되어 있다. N_3가 삽입 이벤트를 요청하면 캐쉬 매니저는 삽입 요청 이벤트를 받아 전역 관리기를 통하여 메타 데이터베이스에 접속한다. 메타 데이터베이스를 통하여 복제본의 메타 정보의 리스트를 받는다. 그 후 복제본이 있는 각각의 노드에 접속하여 노드의 사용비율을 얻는다. 사용비율이 가장적은 노드의 순서로 캐쉬 매니저에 있는 캐쉬테이블에서 변경된 메타 정보를 갖는 캐쉬의 정보를 갱신한다. 그리고 이벤트를 발생한 노드로 갱신된 캐쉬 정보를 전송한다.

캐쉬 매니저를 통한 데이터의 삽입을 통하여 효율적인 질의 전달이 가능하다. 이것은 일반적인 노드는 메타 데이터베이스를 통해 임의의 메타 정보를 캐싱함으로써 데이터의 사용에 있어서 질의 전달의 불균형적인 문제가 있다. 그렇지만 캐쉬 매니저는 캐쉬테이블에 메타 정보 삽입 시에 원본 데이터를 가진 노드들 중 가장 사용비율이 적은 노드의 복제본 메타 정보를 삽입 하여 질의 전송을 최적화하고 안정화한다.

■ 변경 이벤트

캐쉬 매니저는 노드로부터 변경 이벤트를 받은 경

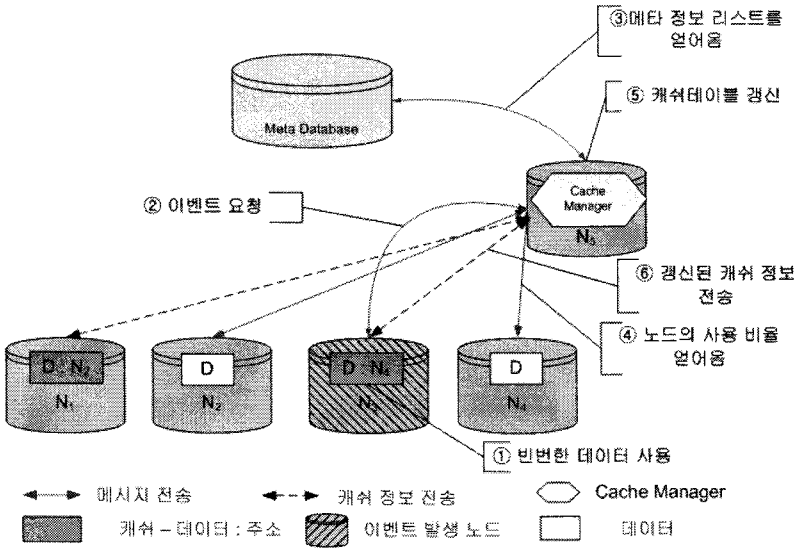


그림 3. 데이터의 삽입 과정

우 이벤트를 분석한다. 분석된 이벤트 메시지를 통하여 변경되지 않은 메타 정보를 가지고 있는 캐쉬테이블 검색한다. 그리고 갱신된 데이터의 복제본 정보를 얻기 위해 메타 데이터베이스를 통하여 복제본의 메타 정보 리스트를 가져온다. 메타 정보 리스트를 통하여 복제본들이 저장된 노드에 접속하여 노드의 사용비율을 얻는다. 복제본을 가진 노드의 메타 정보를 사용비율이 가장적은 순서로 캐쉬테이블의 저장된 메타 정보와 변경한다. 그리고 캐쉬테이블에서 변경되지 않은 메타 정보를 갖는 노드로 캐쉬 정보를 전송한다.

복제본을 가진 노드의 사용비율이 가장적은 메타 정보의 순서로 캐쉬테이블의 주소를 변경하고 캐쉬 정보를 전송하는 것은 여러 캐싱된 메타 정보 중 하나의 주소 값만을 사용하여 발생할 수 있는 사용비율의 불균형 현상을 최소화 하고 최적화하기 위함이다.

(그림 4)는 캐쉬 매니저에서 변경 이벤트를 처리하는 과정을 나타내는 그림이다.

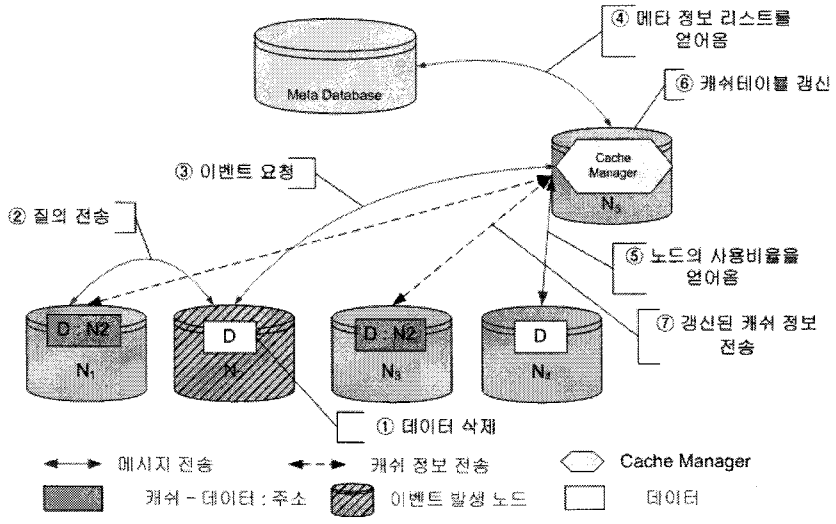


그림 4. 데이터의 변경 과정

(그림 4)는 N_1과 N_3의 캐쉬에는 N_2의 D데이터의 메타 정보가 저장되고, N_2와 N_4에는 원본 데이터인 D데이터가 저장되어 있다. 이런 환경에서 N_2에서 D 데이터가 삭제되었을 경우 변경 이벤트가 처리되는 과정이다.

N_1에서는 D데이터 사용 요청 질의를 받고, 캐싱된 메타 정보를 통하여 N_2로 질의를 전달한다. 질의를 전달 받은 N_2에서 D데이터가 삭제되어 데이터가 없는 경우 질의를 전달한 노드로 오류 메시지를 전송한다. 오류 메시지를 받은 노드는 캐쉬 매니저로 변경 이벤트를 전송한다.

변경 이벤트를 전송 받은 캐쉬 매니저는 삭제된 D데이터와 메타 정보가 일치하는 캐쉬테이블을 검색한다. 그리고 전역 메타 관리기를 통하여 메타 데이터베이스에 접속한다. 메타 데이터베이스를 통하여 복제본의 메타 정보의 리스트를 받는다. 그 후 복제본이 있는 노드의 사용비율을 얻는다. 사용비율이 가장적은 노드의 순서로 캐쉬 매니저에 있는 캐쉬테이블의 메타 정보를 변경한다. 그리고 캐쉬테이블을 통해 메타 정보가 변경된 노드로 갱신된 캐쉬 정보를 전송한다.

(그림 5)는 (그림 4)에서 N_1에 있는 메타 정보가 캐싱되어 있는 구조이다. N_2 노드에서 D데이터가 삭제되기 전의 캐쉬와 삭제 된 후의 캐쉬이다. (그림 5-a)에서는 D데이터에 대한 질의 요청을 N_2로 전달한다. 하지만 데이터의 삭제로 캐쉬가 갱신되면 (그림 5-b)는 D데이터의 주소 값을 N_4로 갖는다. 이 후 D데이터의 질의 요청이 있을 경우 N_4의 노드로 질의를 전달한다.

기존의 캐쉬 방식에서 원본 데이터가 변경되어도 캐쉬는 변경되지 않은 데이터의 메타 정보를 저장하고 있다. 그리고 요청 질의가 들어 올 경우 질의를 처리하기 위해 캐싱된 메타 정보를 통하여 질의를 잘못 전달할 수 있다. 질의를 잘못 전달받은 노드는 요청 질의를 처리하기 위하여 메타 정보를 얻어오며, 또 한번의 메타 정보를 검색을 통하여 다른 노드로 질의를 전달한다. 이렇게 여러 노드에서 변경된 데이터를 찾기 위해 반복 수행함으로써 네트워크 비용과 처리 시간의 증가를 가져온다.

기존의 방식의 문제점을 보완하기 위해 캐쉬 매니저는 캐싱된 메타 정보를 통해 질의를 전달하였을 경우 질의 전달 오류 메시지를 받으면 변경 이벤트를 발생하여 캐쉬 매니저로 전송한다. 이벤트를 전송받은 캐쉬 매니저는 변경된 원본 데이터의 메타 정보를 가지고 있는 캐쉬테이블의 주소 갱신하여 변경된 캐쉬 정보를 전송한다. 캐쉬 매니저로 인한 캐쉬테이블 갱신으로 사용자가 원본 데이터를 사용하려고 할 때 데이터의 변경으로 인한 잘못된 질의 전달을 최소화하여 질의 처리 시간과 네트워크 비용을 감소한다. 이것은 노드 하나에서 캐싱된 정보를 갱신하여 시스템의 성능을 향상시키는 것이 아니라 변경된 원본 데이터의 메타 정보를 갖는 모든 노드의 캐쉬를 갱신하여 전체적인 시스템 성능에 향상을 갖는다.

■ 이벤트 처리 알고리즘

(알고리즘 3)는 캐쉬 매니저에서 삽입과 변경 이벤

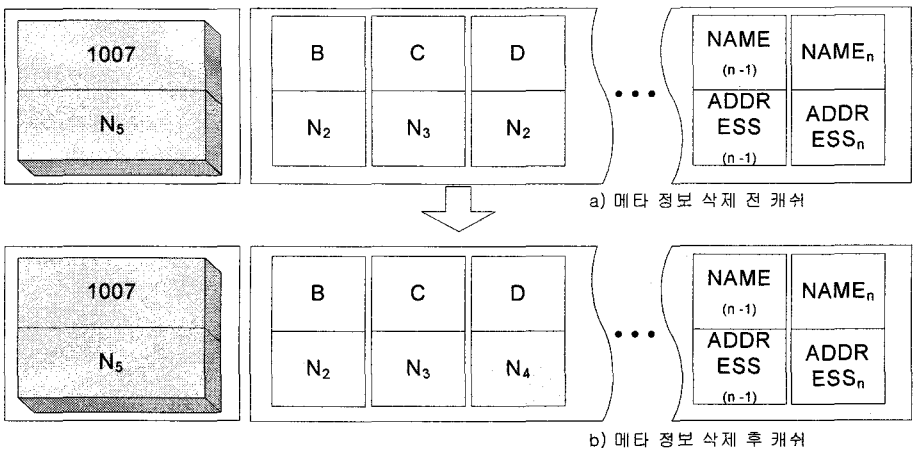


그림 5. 메타 정보전과 삭제 후의 캐쉬 구조

```
Input
event message : 이벤트 메시지
result  :  질의 결과(삽입, 변경 등을 저장)
Output
Void
Begin
01 :  if (Result is request delete) then
02 :    Exchange cache table list = cache_table_search(event message)
03 :       if (Count of exchange cache table list is not exist) then
04 :            exit request Query message
05 :       endif
06 :  endif
07 :  GetMetaListFromMDB()
08 :  for (Count of meta data list)
09 :    Get data frequency form Node that has a real data
10 :  endfor
11 : if (Result is request input) then
12 :    Lowest frequency data address stored in cache table
13 :    Send cache table message to event was created
14 :  else
15 :    Data address stored in cache table by ascending order of frequency
16 :    Send cache table message to each own node
17 :  endif
End
```

알고리즘 3. 이벤트 처리 알고리즘

트를 처리하는 알고리즘이다. cache_table_search() 함수는 캐쉬 매니저가 가진 캐쉬테이블에서 변경된 원본 데이터의 메타 정보를 갖는 캐쉬 정보를 검색하는 함수이다. 인자로는 이벤트 메시지에서 들어온 데이터를 갖는다.

질의를 분석하여 나온 결과가 변경 이벤트인지 검사한다(01라인). 변경 이벤트일 경우 변경된 데이터의 메타 정보를 갖는 캐쉬 정보만을 캐쉬테이블을 검색하여 리스트로 만든다(02라인). 검사된 캐쉬 정보가 존재하지 않는다면 이벤트를 종료한다(03~04라인).

변경 이벤트가 아닐 경우 메타 데이터베이스에 접속하여 복제본의 메타 정보 리스트를 얻는다(07라인). 메타 정보 리스트를 통하여 복제본들을 가진 노드의 사용비율을 얻는다(08~09라인). 이벤트 메시지가 삽입일 경우 복제본 데이터를 가진 노드 중 사용비율이 적은 노드의 주소를 캐쉬테이블에 삽입하여 이벤트가 발생한 노드로 갱신된 캐쉬 정보를 전송한다(11~12라인). 삽입 이벤트가 아닐 경우는 변경 이벤트 메시지로써 복제본을 가진 노드의 사용비율 집합을 계산하여 작은 비율을 갖는 노드의 복제본 메타 정보부터 변경될 캐쉬테이블에 저장된다. 갱신된 캐쉬 정보는 각각 자신의 노드로 전송된다(16~17라인).

4. 성능 평가

본 장에서는 질의 전달 최적화를 위한 캐쉬 관리 기법의 평가를 위한 실험 환경에 대해 설명하고, 질의에 대한 집중도 변화, 원본 데이터변경 시의 변화, 캐쉬 크기에 따른 성능의 변화에 대하여 제안 기법과 기존 기법과의 성능을 비교 수행한다.

4.1 실험 환경

본 연구에서는 제안 기법의 평가를 위해 MCC에서 개발한 CSIM 툴을 사용하여 평가 하였다. CSIM는 C/C++언어 기반의 시뮬레이션 툴로 분산 환경의 성능평가 및 알고리즘 평가 등 대부분의 시뮬레이션이 가능한 툴이며, 이미 여러 논문에서 CSIM을 통한 성능평가가 작성된 바 있다[11].

표 1. 실험 환경

실험 요소	데이터 범위	실험 요소	데이터 범위
실험시간	3000unit	질의 입력 간격	0.1 unit
서버 수	5~40 개	캐쉬 매니저 처리 시간	2 ~ 3 unit
총 데이터 개수	100 개		
질의 처리시간	4~7 unit	메타데이터베이스에서 데이터의 검색 시간	2 ~ 3 unit
네트워크 전송시간	0.5 unit		

제안 기법의 구현을 위해 다수의 서버와 하나의 메타 데이터베이스, 또한 캐쉬 매니저 프로세스를 구현 하였다. 또한 모든 서버에 제안한 캐쉬 구조를 저장 하였다. MS Visual C++ 6.0을 사용하여 구현 하였으며, 전체적인 구성은 가능한 실제 환경에 적합하게 구현 하였으며, 평가 환경은 다음과 같다.

모든 평가 환경은 실험 시간 3000unit 동안의 질의 처리 결과로 나타난다. 서버는 총 5~40개로 나타내었다. 총 데이터의 개수는 100 개 이며, 캐쉬의 크기는 데이터의 개수를 증가 하여 나타내었다.

실험을 위한 비교 평가 대상은 첫 번째로 캐쉬를 가지고 있지 않은 경우이다. 이것은 자신이 가지고 있지 않은 데이터에 대한 질의 요청을 받을 경우 항상 메타 데이터베이스에 접속하여 데이터가 있는 곳의 주소를 받아와서 질의를 전달하는 방식을 사용하였으며, 명칭을 General 로 사용한다.

두 번째 방식은 캐쉬 방식은 각각의 노드에서 캐쉬를 관리하는 경우이다. 이 경우에는 자신이 가지고 있지 않은 데이터에 대한 질의 요청을 받을 경우 캐쉬에서 데이터를 찾아서 질의를 전달하며, 캐쉬에 원하는 데이터가 존재하지 않을 경우 메타 데이터베이스에 접속하여 메타 정보를 가져온다. 이후 명칭을 Cache 로 사용한다.

그리고 제안 기법은 본 연구에서 제안 하는 구조를 구현하며, 이후 본 제안 기법의 명칭을 Manager 로 한다.

4.2 실험 평가

본 실험 평가에서는 질의에 대한 집중도 변화 비율, 원본 데이터의 변경 시에 메타 정보를 캐싱한 노드 비율에 따른 성능 변화, 캐쉬 크기에 따른 성능의 변화 비율을 주요 평가 인자로 보고 이것을 집중적으로 평가 하였다.

모든 결과는 사용 요청 질의를 보내고 처리된 후의 처리 시간과 질의 처리량으로 나타내었고, 평가는 백분율로 환산하여 나타낸다. 또한 전체 0 ~ 100 % 의 비율을 갖는다.

(그림 6)는 사용자 요청 질의에 대한 집중도 변화 비율을 나타낸다. 질의는 노드 전체로 균등하게 보내진다.

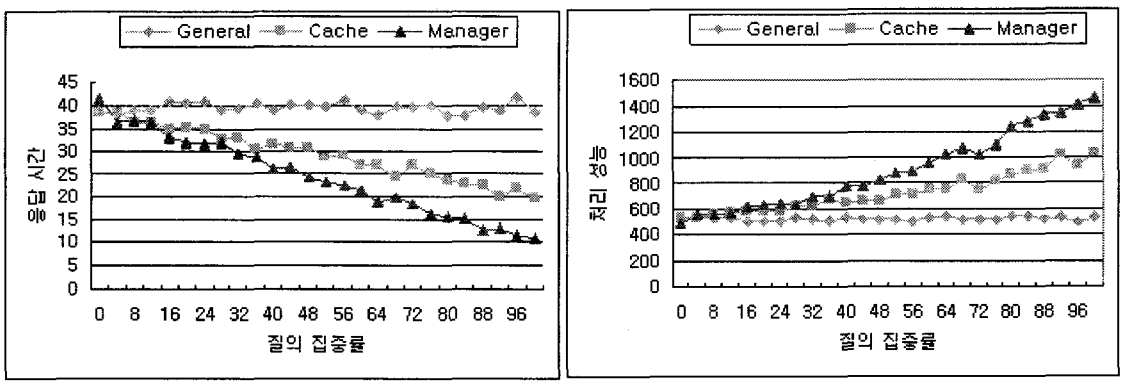


그림 6. 질의 집중도 변화 비율

(그림 6)에서는 동일한 질의의 요청 비율이 점차적으로 증가 하는 것을 나타내는 성능평가이다. General 방식은 동일한 질의의 요청을 받아도 메타 데이터베이스에 계속적으로 접속을 하기 때문에 변화가 보이지 않는다. 하지만 Cache 방식과 Manager 방식은 처리 성능이 향상 된 것을 볼 수 있다. 이것은 Cache 방식과 Manager 방식은 캐싱된 메타 정보를 통하여 빈번히 접속되는 노드에 대한 정보를 저장하여 나타나는 것으로 판단된다.

질의 집중도가 높아질수록 Manager 방식이 Cache 방식 보다 처리 성능이 더 좋게 나타났다. 이것은 메타 데이터베이스로부터 임의의 데이터 주소를 얻는 Cache방식과는 달리 캐쉬에 새로운 메타 정보를 삽입할 때 캐쉬 매니저에서 가장 사용율이 적은 노드의 복제본의 메타 정보를 캐싱하여 질의 전송 최적화를 유지하여 나타나는 현상으로 판단된다.

(그림 7)은 캐싱된 메타 정보의 원본 데이터가 변경 되었을 경우의 질의 처리성능을 나타낸다.

(그림 7)에서는 원본 데이터가 변경된 노드로 캐싱된 메타 정보를 통하여 질의를 전달할 경우 메타 정보를 캐싱하고 있는 노드수의 비율을 변경하여 나타내는 성능평가이다.

General 방식은 원본 데이터가 변경되어도 메타 정보를 가져오고, 변경되지 않아도 메타 정보를 가져오기 때문에 성능변화가 일정하게 나왔다.

Cache 방식과 Manager 방식에서는 여러 노드의 캐싱된 메타 정보의 중복이 많을수록 Manager 방식이 처리 성능 좋았다. 이것은 본 논문에서 제안한 캐쉬 관리 기법에서 원본 데이터 변경 시에 캐쉬 매니저를 통하여 캐쉬테이블을 갱신하여 갱신된 캐쉬 정보를 전송하여 나타나는 현상으로 판단된다.

(그림 8)은 캐쉬 크기 비율에 따른 처리 성능의 변화를 나타낸다.

(그림 8)은 Cache 방식과 Manager 방식에서 메타

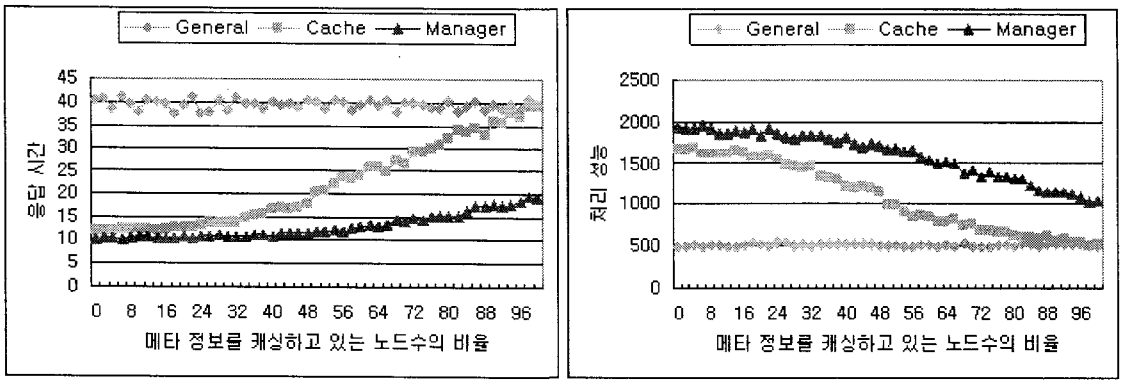


그림 7. 원본 데이터 변경 시 성능 변화

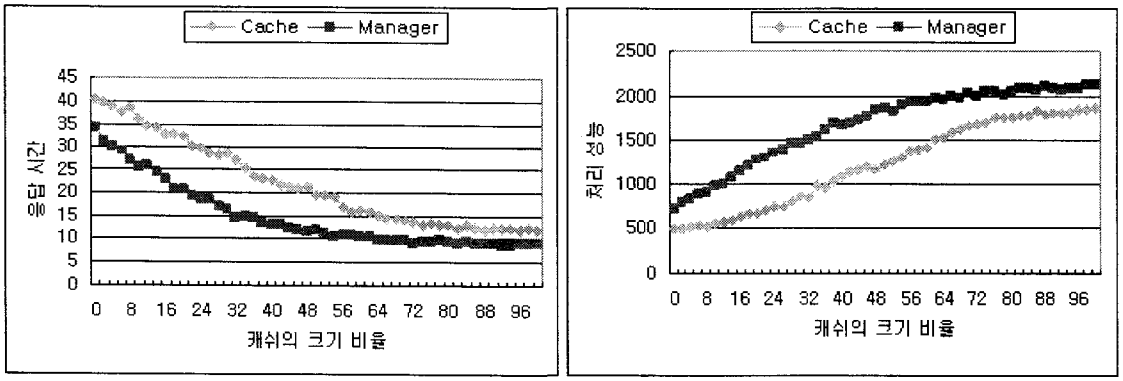


그림 8. 캐쉬의 크기 변화 비율

정보를 저장 할 수 있는 캐쉬 크기가 같다는 가정 하에 테스트한 성능 평가이다. General 방식은 캐싱할 수 있는 공간이 없다는 가정 하에 성능 평가에서 배제하였다.

Cache 방식과 Manager 방식은 각각은 데이터를 가지고 있는 크기가 점점 증가함에 따라서 처리 성능이 점차 향상되는 것을 볼 수 있다. 이것은 메타 데이터베이스를 거치는 질의가 점차 줄어서 나타나는 현상으로 판단된다.

또한, Manager 방식이 Cache 방식보다 좋은 것으로 나타난다. Cache 방식에서는 캐쉬 사이즈를 계속적으로 크게 하여 빈번하게 사용되는 모든 메타 정보를 캐싱하여도 임의적으로 사용되는 메타 정보가 불균형적으로 저장되어 성능저하가 나타난다. 이러한 문제점을 해결하여 Manager에서는 캐쉬 매니저를 사용하여 빈번하지 않은 데이터의 메타 정보를 캐싱하여 캐쉬를 최적화함으로써 시스템의 성능이 향상된다. 또한 70% 이상일 때에는 Manager 와 Cache에서 더 이상의 성능의 향상이 보이지 않는다. 이것은 빈번히 사용되는 거의 모든 정보가 캐싱되어 시스템의 성능이 더 이상 향상되지 않는 것으로 판단된다.

캐쉬 크기 변화 비율의 성능 평가로 캐쉬의 사이즈는 클수록 좋긴 하지만 캐쉬의 크기를 무한정 크게 놓을 수 없기에 캐쉬 사이즈가 전체 시스템의 40%이하 일 때 효율성을 극대화 시킬 수 있다고 판단된다.

5. 결론 및 향후 연구

그리드 데이터베이스에서 질의 전달 최적화를 위해 캐쉬를 사용한다. 캐쉬에서는 자주 사용되는 메타 정보를 캐싱하여 서버의 부하와 전체 네트워크 트래픽을 감소하여 사용자에게 캐싱된 정보를 통한 질의 전달로 빠른 응답 시간을 제공한다.

본 논문에서는 질의 전달 최적화를 위한 캐쉬 관리 기법을 제안하고, 이에 대한 다른 기법과의 성능 평가를 수행하였다. 제안 기법은 복제본 데이터의 불균형적 사용과 캐싱된 메타 정보의 일관성을 고려하여 그리드 데이터베이스에서 캐쉬 매니저라는 관리 프로세서를 사용하여 캐쉬를 관리한다.

캐쉬 매니저를 사용함으로써 복제본이 저장된 노드의 사용빈도를 고려하여 복제본의 사용비율을 균등하게 하였다. 그리고 메타 정보의 비일관성으로 인한 네트워크 증가 문제를 해결하였다. 복제본의 사용빈도를 고려함에 따라서 사용빈도가 적은 메타 정보를 노드에 캐싱하여 복제본 데이터의 불균형적인 사용을 해소하여 시스템 성능의 향상을 나타내었다. 또한 네트워크 비용의 증가 문제를 해결하기 위하여 캐쉬 일관성을 보장함으로써 잘못 전달되는 질의의 사용을 줄여서 네트워크 증가 문제를 해결하여 시스템 성능을 향상 하였다. 성능 평가에서는 그룹의 캐싱된 메타정보를 취합해 이를 바탕으로 캐싱된 정보를 변경하여 기존 기법에 비해 좋은 성능을 보였다.

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

향후 연구로는 각종 응용 환경에서 네트워크 구성과 그리드 데이터베이스에서의 캐쉬에 대한 서로 다른 기법 간의 성능 평가가 필요하다.

참 고 문 헌

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

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

[3] Maria A., Nieto-Santisteban, Jim Gray, Alexander S. Szalay., James Annis, Aniruddha R. Thakar, and William J. O'Mullane, "When database System Meet Grid," *Proceedings of the 2005 CIDR Conference*, pp.154~161, 2005.

[4] M.A. Blaze, "Caching in Large-Scale Distributed File Systems," *Princeton University, Ph.D. Thesis*, Jan. 1993.

[5] Cristina Duarte Murta, Virgílio Almeida, and Wagner Meira, Jr, "Analyzing Performance of Partitioned Caches for the WWW," *Proceedings of the 3rd International WWW Caching Workshop*, June 1998.

[6] Jia Wang, "A Survey of Web Caching Schemes for the Internet," *ACM Computer Communication Review*, 25(9), pp. 36-46, Oct. 1999.

[7] K. Wilkinson, and M. Neimat, "Maintaining Consistency of Client-Cached Data," *In Pro-*

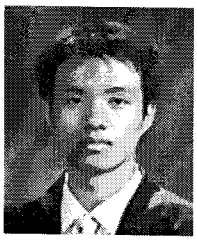
ceedings of the 16th Conference on Very Large Data Bases, Brisbane, 1990.

[8] Michael J. Franklin, Michael J. Carey, and Miron Livny, "Transactional Client-Server Cache," *Transaction on Database Systems*, Vol. 22, No. 3, Sep, 1997.

[9] M. Oszu, U Dayal, and P. Valduriez, *Distributed Object Management*, Morgan Kaufmann Publishers, Inc, 1994.

[10] R. Alonso, D. Barbara, and H. Garcia Molina, "Data Caching Issues in an Information Retrieval System," *ACM Transactions on Database Systems*, Vol. 15, No. 3, Sep. 1990.

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



신 숭 선

2006년 서원대학교 컴퓨터 교육학과 졸업(이학사)
2006년~현재 인하대학교 컴퓨터정보공학과 석사과정
관심분야 : 공간 데이터베이스, 그리드 데이터베이스, 분산 데이터베이스 등

장 용 일

1997년 인하대학교 전자계산공학과 (공학사)
2001년 인하대학교 컴퓨터공학부 (공학석사)
2003년~현재 인하대학교 컴퓨터정보공학과 박사과정
관심분야 : 웹 & 모바일 GIS, 데이터베이스 클러스터, 위치기반서비스, 그리드 데이터베이스

이 순 조

1985년 인하대학교 전자계산학과 (이학사)
1987년 인하대학교 전자계산학과 (이학사)
1995년 인하대학교 전자계산학과 (공학박사)
1995년~1997년 대림대 전자 계산과 교수
1997년~현재 서원대학교 컴퓨터공학과 부교수
관심분야 : 데이터베이스, 실시간 데이터베이스 시스템, GIS, 데이터베이스 시스템의 보안

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
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2004년~2006년 인하대학교 정보통신대학원 원장
2006년~현재 인하대학교 대학원 원장
관심분야 : 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스