Journal of Korea Multimedia Society Vol. 11, No. 7, July 2008(pp. 914-927)

그리드 데이터베이스에서 질의 처리를 위한 캐쉬 관리 기반의 부하분산 기법

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

요 약

그리드 컴퓨팅 환경에서 대용량 데이터의 처리와 가용성 향상, 데이터 통합을 위해 그리드 데이터베이스 관리 시스템이 사용된다. 그리드 데이터베이스 관리 시스템은 효율적인 질의처리를 위해 여러 노드로 질의를 분산하여 처리한다. 하지만 질의 처리가 임의의 노드에 집중되어 처리 성능이 감소되고, 작업 부하의 불균형이 발생한다. 본 논문에서는 그리드 데이터베이스에서 질의 처리를 위한 캐쉬 관리 기반의 부하분산 기법을 제안한다. 제안 기법에서는 여러 노드에 있는 캐쉬들을 관리하기 위해 캐쉬 관리자를 사용하며, 캐쉬 관리자는 노드를 지역별 그룹으로 연결하고 자신의 그룹 안에 있는 노드의 캐싱된 메타 정보를 관리한다. 노드는 캐쉬 관리자를 통해 질의를 전달 할 최적의 메타 정보를 캐싱한다. 노드에서 캐싱된 메타 정보를 통해 질의를 전달하여 노드의 부하를 분산한다. 제안 기법은 캐쉬 기반으로 부하가 적은 노드에서 질의를 처리하여 노드들의 부하를 분산하여 질의 처리시에 향상된 성능을 보인다.

Load Balancing Method for Query Processing Based on Cache Management in the Grid Database

Soong-Sun Shin[†], Sung-Ha Back[††], Sang-Hun Eo[†††], Dong-Wook Lee[††††], Gyoung-Bae Kim[†††††], Weon-Il Chung[††††††], Hae-Young Bae[†††††††]

ABSTRACT

Grid database management systems are used for large data processing, high availability and data integration in grid computing. Furthermore the grid database management systems are in the use of manipulating the queries that are sent to distributed nodes for efficient query processing. However, when the query processing is concentrated in a random node, it will be occurred with imbalance workload and decreased query processing. In this paper we propose a load balancing method for query processing based on cache Management in grid databases. This proposed method focuses on managing a cache in nodes by cache manager. The cache manager connects a node to area group and then the cache manager maintains a cached meta information in node. A node is used for caching the efficient meta information which is propagated to other node using cache manager. The workload of node is distributed by using caching meta information of node. This paper shows that there is an obvious improvement compared with existing methods, through adopting the proposed algorithm.

Key words: Grid database(그리드 데이터베이스), Cache management(캐쉬 관리자), Load Balance(부하분산), 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 : 2007년 12월 14일, 완료일 : 2008년 4월 21일
[†] 준회원, 인하대학교 정보공학과 박사과정
(E-mail : hermit@dblab.inha.ac.kr)
[††] 인하대학교 정보공학과 박사과정
(E-mail : shbaek@dblab.inha.ac.kr)
[†††] 인하대학교 정보공학과 통합박사과정
(E-mail : eosanghun@dblab.inha.ac.kr)
[††††] 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)
※ 본 연구는 건설교통부 첨단도시기술개발사업 - 지능형 국토정보기술혁신 사업과제의 연구비지원(07국토정보C05)에 의해 수행되었습니다.

1. 서 론

그리드 데이터베이스는 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 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]. 본 논문에서 노드는 데이터베이스 관리 시스템으로 구성되며, 또한 클러스터 시스템으로 구성된 데이터베이스 관리 시스템이다.

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.

사용자 질의는 보통 여러 데이터들을 요구하게 되며, 자신이 가지고 있지 않은 데이터들을 처리 하기 위해 해당 데이터를 포함하는 노드로 질의가 전송되어 처리 된다. 그리드 데이터베이스에서는 질의 전달 최적화를 위해 캐쉬를 사용한다[2]. 캐쉬에 빈번히 사용되는 데이터의 메타 정보를 메타 데이터베이스에서 가져와 캐싱하며, 캐싱된 정보를 통해 질의 전달의 비용을 감소한다[3]. 기존의 캐쉬 관리 기법은 임의의 메타 정보들의 전달로 인하여 질의를 처리할 때, 여러 노드들이 중복된 복제본 데이터를 갖고 있는 상황에서 질의가 한 노드로 집중되어 작업부하의 불균형으로 시스템의 성능이 저하되는 문제가 있다[4,5].

작업 부하의 불균형으로 인한 성능 저하 문제를 해결하기 위해 기존의 연구로 링 기반 연결 구조를 이용한 부하분산 기법이 있다[6]. 이 기법은 복제본을 링 구조로 연결하며 작업 부하가 임계치를 초과하면 질의를 정방향 링크를 따라서 질의를 다른 노드로 전달하여 부하를 분산한다. 하지만 부하가 정방향 연결을 따라서 계속적으로 발생한 경우 질의 전달이 계속적으로 발생하여 성능이 저하되는 문제가 있다. 또 다른 연구로 데이터의 복제와 이주를 통한 부하분산 기법이 있다[7,8]. 이 기법은 부하가 높은 노드의 데이터를 다른 노드로 복제 하거나 이주하여 노드의 부하를 분산한다. 데이터 이주와 복제에서는 한정된 저장 공간에서 계속적인 데이터의 이주와 복제로 노드의 저장공간이 증가하는 문제 및 이주와 복제의 트레이드 오프 문제를 해결해야 한다.

본 논문에서는 그리드 데이터베이스에서 질의 처리를 위한 캐쉬 관리 기반의 부하분산 기법을 제안한다. 그리드 데이터베이스 각각의 노드는 캐쉬를 사용하여 빈번히 사용되는 데이터의 메타 정보를 저장하여 효율적인 질의 처리를 가능하게 한다. 여러 노드에 있는 캐쉬들을 관리하기 위해 캐쉬 관리자가 사용된다. 캐쉬 관리자는 노드를 지역별 그룹으로 연결하고 자신의 캐쉬관리자 그룹 안에 있는 노드의 캐싱된 메타 정보를 관리한다. 노드에서는 빈번히 사용되는 데이터의 메타 정보를 캐싱 하기 위해 캐쉬 관리자로 이벤트 메시지를 전달한다. 이벤트 메시지를 전달 받은 캐쉬 관리자는 메타 데이터베이스로부터 메타 정보 리스트를 얻는다. 이렇게 얻어진 메타 정보 리스트를 통해 복제본 데이터가 있는 노드들의 시스템 정보를 가져온다. 캐쉬 관리자는 가져온 시스템 정보들을 취합하여 시스템들 중 최적의 메타 정보를 노드로 전달한다. 노드에서는 자신의 노드를 모니터링 하여 자신이 정한 임계치의 값을 넘은 경우를 과부하라고 한다. 노드에서는 이런 과부하로 인한 시스템 성능의 저하가 발생한 경우 노드는 과부하 이벤트 메시지를 캐쉬 관리자에게 전달한다. 과부하 메시지를 전달 받은 캐쉬 관리자는 과부하 노드의 데이터 정보를 캐싱하고 있는 노드들의 캐싱된 메타 정보를 변경하여 과부하 노드의 부하를 분산한다.

제안 기법은 캐쉬 기반으로 부하가 적은 노드에서 질의를 처리하여 노드들의 부하를 분산하여 시스템의 성능을 향상했다. 성능 평가를 통해 제안기법이 기존의 기법보다 질의 처리시에 향상된 성능을 보인다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로 그리드 데이터베이스에서의 질의 처리 과정과 기존의 질의 처리의 부하분산 기법 대해서 소개한다. 3장에서는 본 논문에서 제안하는 캐쉬 관리 기반의 부하분산 기법을 위해 부하 분산을 위한 자료 구조와 알고리즘 그리고 부하가 분산되는 예를 보인다. 4장에서는 제안 기법에 대한 성능 평가를 수행한다. 마지막으로 5장에서 결론 및 향후 연구를 기술한다.

2. 관련연구

2.1 그리드 데이터베이스에서의 질의 처리

그리드 데이터베이스에서의 질의 처리에 대한 대표적인 연구로는 분산 질의 처리에 대한 연구인 OGSA(Open Grid Services Architecture)-DQP (Distributed Query Processor)가 있다. DQP는 OGSA 구조 기반의 분산 질의 처리 방법으로서 그리드 기반의 서비스 형태로 제공되며, 여러 노드에서의 분산 질의 처리를 위한 분석, 최적화, 분할 및 스케줄링 등의 기능을 포함하고 있다. 주요한 기능으로는 질의 평가 서비스로서 각 노드의 성능에 대한 평가에 따라 질의 수행 계획을 조정하는데 이용된다[9,10].

DQP는 하나의 질의에 대해 얻어지는 수행 계획을 노드에서의 처리 단위로 분할한 다음 각 영역을 질의 평가 서비스(QES: Query Evaluation Service)를 통해 처리 과정이 수행될 노드 및 질의 수행 계획에 대한 재조정 과정을 수행하게 된다.

그림 1은 전형적인 수많은 분산 질의처리에서 사용되는 질의 처리 패러다임의 구조이다. 이 그림은 2개의 최적화 패러다임 구조를 설명하며, polar에 사용된다[11]. 패러다임은 크게 2가지로 나뉘며 첫 번째 구조는 단일 노드의 최적화에 사용되며, 두 번째 구조는 분산된 노드들 사이에서의 최적화에 사용되며, 분산 질의 처리에서 특히 각각의 컴포넌트들 사이에서 데이터의 변환에 대해 자세히 설명하고 있다.

그림 1은 일반적으로 사용되는 DQP의 기본 구조이다.

질의를 입력으로 받게 되면 해당 질의는 데이터 소스에 있는 메타 데이터를 참조하여 자료형에 부합하는지를 검사하기 위한 구문 분석 단계로 접어든다. 구문 분석 결과는 트리 구조로 표현되며 논리적 최적화를 통해 논리적 계획이 생성 된다. 논리적 계획은 질의를 실행하기 위해 필요 되는 트리들을 가르키는 또 다른 노드의 표현이다. 물리적 최적화는 논리적 계획을 통해 물리적 계획으로 변경한다. 물리적 계획은 실제 실행을 위해 어떻게 정확한 데이터에 대하여 접근할 것인지 실행할 것인지를 결정한다. 논리적 최적화와 물리적 최적화는 단일 노드 최적화와 연속 계획처리로 나뉜다. 질의 분배자는 연속 계획을 분배하거나 하부계획으로 나누며, 스케줄러를 통해 자원을 할당 한다.

2.2 링 기반 연결 구조를 이용한 부하분산 기법

링 기반 연결 구조는 모든 복제본을 하나의 링 형태로 연결하며, 각각의 링크는 정방향과 역방향의 방향성을 갖는다. 정방향 링크는 노드의 작업 부하가 한계 값을 벗어난 상태에서 다른 노드로 질의를 전달

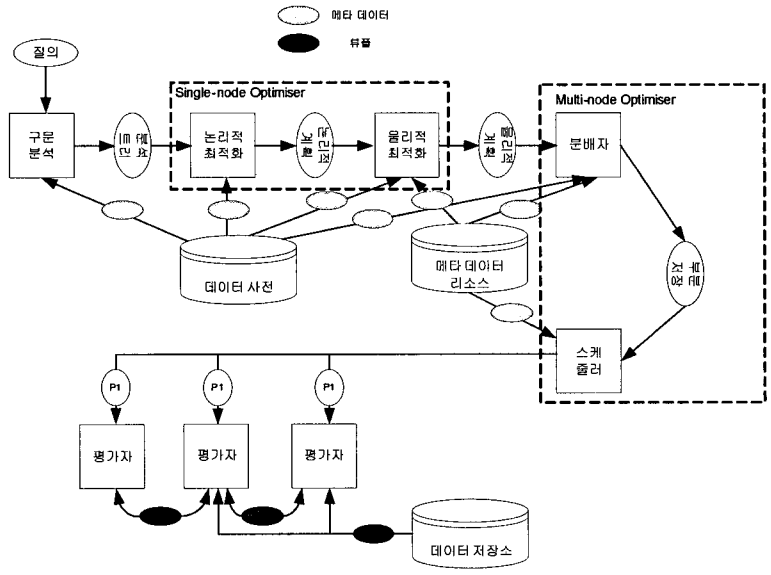


그림 1. 일반적인 DQP에서의 질의 처리 과정

할 때 사용된다. 역방향 링크는 노드의 작업 부하가 한계 값을 벗어난 경우 이전 노드로 자신의 상태 정보를 전달할 때 사용된다. 링 기반 연결 구조로 구성되는 데이터베이스의 형태는 그림 2과 같다[6].

그림 2에서 데이터 A, B, C는 각각의 노드에 복제본의 형태로 분포되어 있다. 연결 구조는 노드와 상관없이 복제본을 기준으로 구성되어 있다. 예를 들어, 데이터 B는 B1→B2→B3의 순서로 연결이 되어 있으며, B2의 작업 부하가 한계치를 넘은 경우 노드 N2로 입력되는 질의 중에서 B와 관련된 질의는 정방향 링크로 연결된 B3로 전달된다. 마찬가지로 작업 부하가 한계치를 넘은 노드 N3에서 A3에 대한 질의가 입력된 경우 복제본 A를 포함하는 노드 N5로 질의가 전달된다. 역방향 링크는 노드의 과부하 상태 또는 보통 상태를 전달할 때 사용된다.

정방향 링크와 역방향 링크는 평상시에는 그림 2에서 설명한대로 사용된다. 그러나 임의 노드에 사용자 질의가 집중되어 더 이상 질의 처리가 불가능한 경우에는 정방향 링크의 구조가 달라진다. 현재의 구조에서 해당 노드를 이전 노드의 정방향 링크에서 제거해야 한다.

과부하 노드가 발생하면, 해당 노드는 자신의 상태 정보와 자신의 정방향 링크 정보를 포함하는 메시지를 생성한다. 생성된 메시지는 자신의 역방향 링크를 통해 이전 노드로 전달된다. 메시지를 전달 받은 이전 노드는 자신의 정방향 링크를 메시지에 포함된 링크 정보로 변경한다. 만일 이전 노드 또한 과부하 상태라면 해당 메시지를 또 다른 이전 노드로 전송한다.

만일 모든 노드들 중 하나의 노드만 보통 상태이고 다른 노드들이 과부하 상태라면 모든 노드의 정방향 링크는 이 노드를 가리킨다. 모든 노드가 정해진

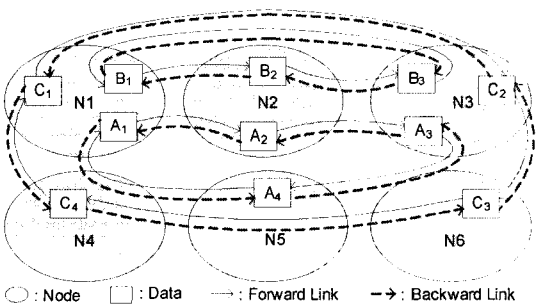


그림 2. 그리드 데이터베이스 환경의 복제본 데이터의 링 기반 연결 구조

한계 값을 초과하여 사용자 질의를 수행하는 경우, 제안 기법은 모든 노드의 정방향 링크를 삭제한다. 이 상태에서는 다른 노드로 질의를 전송하는 것이 필요 없다. 모든 노드가 자신의 한계치를 넘은 상태에서 질의를 수행하고 있기 때문에 질의를 재전송하는 것은 네트워크의 부하만 가중시킨다. 임의의 노드가 과부하 상태에서 보통 상태로 바뀌면 과부하 상태로 바뀔 때와 마찬가지로 해당 노드는 자신의 역방향 링크로 메시지를 전달한다. 이전 노드는 이 메시지를 받고, 자신의 정방향 링크를 해당 노드로 수정한다. 또한, 이전 노드의 상태가 과부하 상태라면 또 다른 이전 노드로 메시지를 전송한다. 링 기반 연결 구조는 자신의 노드에 과부하가 발생 하였을 경우에만 자신에게 전달된 질의를 복제본을 갖고 있는 노드로 전달한다. 전체의 시스템 중에서 질의를 처리할 최적의 노드를 찾는 것이 아니기 때문에 질의 처리시에 성능이 저하된다. 또한 부하 분산할 때 정방향 연결 구조를 따라서 질의를 전달하여 질의를 처리할 수 있는 최적의 노드에게 질의를 전달하는 것이 아니기 때문에 질의 처리의 성능이 저하되는 문제점을 갖는다.

2.3 데이터 복제와 이주를 통한 부하 분산기법

데이터의 복제와 이주를 통한 부하 분산기법에서 노드는 각자의 로컬 영역 안에서의 클러스터 시스템으로 구성된다. 각각의 노드는 자신의 주변 노드들을 결정하고 부하 분산 기법을 이용하여 자신의 노드와 주변 노드들의 부하정보를 확인하여 각각의 노드들의 부하를 분산한다[7,8].

노드들은 일정한 주기를 기준으로 인접한 노드들과 부하정보를 교환한다. 이때 만약 인접한 노드에서 자신의 노드보다 부하가 임계치 이상으로 높을 경우 부하를 분산할 데이터를 찾게 된다. 먼저 데이터가 자주 사용되는 데이터인지 찾은 후, 이 데이터가 데이터 이주 및 복제 된 데이터 인지 고려 한다. 그리고 임계치가 이상 높은 노드의 부하를 분산하기 위해 데이터를 가져올 경우, 자신의 디스크 공간의 용량을 확인 한다. 이때 디스크 공간이 충족되면, 부하가 높은 노드에서는 부하가 낮은 노드로 자신의 총 부하와 가용할 수 있는 디스크 공간의 정보를 전달한다. 정보를 전달 받은 노드는 자신의 노드에 데이터를 복제할지 이주할지를 결정한다. 이때 부하가 높은 노드의 부하 정보인 각 노드들의 부하와 디스크의 가용 공간

을 확인하여 데이터의 이주와 복제의 특성에 부합되도록 데이터를 가져온다.

데이터의 이주와 복제를 통한 부하 분산 기법은 복제본 데이터의 허용으로 질의를 처리할 때 여러 노드로 질의가 분배 되어서 질의 처리의 가용성이 증가하는 이점이 발생한다. 하지만 수많은 복제본이 분산되어 있기 때문에 복제본의 허용은 부하의 균형을 맞추는 것이 어렵다. 또한 복제본 데이터의 사용으로 가용성이 증가 하지만 디스크 공간의 비용을 증가하는 문제점을 갖는다.

데이터의 이주는 그리드 환경에서 부하 균형을 맞추는 것이 가능하며, 디스크의 소모를 줄인다. 하지만 질의를 처리할 경우 가용성이 저하되는 단점이 있으며, 또한 데이터를 이주할 경우 다른 노드로 이주된 데이터를 찾지 못하여 발생하는 데이터 링크의 오류가 발생 할 수 있다.

데이터의 이주와 복제는 자신의 노드에 임계치 이상의 부하가 나타났을 경우 부하를 분산한다. 이때 자신의 부하를 분산하기 위해 데이터를 복제 하거나 이주 하는데 각각의 방법은 트레이드 오프의 문제점을 발생한다. 노드에 과부하가 발생 하였을 경우에만 자신의 주변 노드로 데이터를 이주하거나 복제하여 부하를 분산하기 때문에 질의 처리 시에 전체적인 시스템의 부하를 조율하여 최적의 성능을 나타낼 수 없다.

3. 캐쉬 관리 기반의 부하분산 기법

본 장에서는 질의 처리를 위한 캐쉬 관리 기반의 부하분산 기법을 제안한다. 먼저 제안 기법을 위한 시스템의 환경을 설명한 후, 캐쉬 관리를 위해 사용하는 데이터 구조 및 분산 알고리즘을 설명한다. 그 후 캐쉬 관리에서 질의 전달을 통해 부하가 분산되는 예 대하여 설명한다.

3.1 그리드 데이터베이스에서 캐쉬 관리

그림 3은 캐쉬 관리 기반의 부하 분산을 위한 확장된 그리드 데이터베이스 구조이다.

각각의 노드에서는 캐쉬를 갖는다. 그리드 데이터베이스에서는 캐쉬를 통해 빈번히 사용되는 데이터의 메타 정보를 저장하여 질의 전달의 효율성을 증대한다.

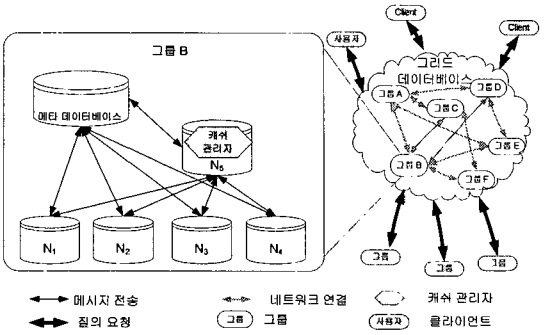


그림 3. 캐쉬 관리 기반의 부하 분산을 위한 확장된 그리드 데이터베이스 구조

그림 3은 부하 분산을 위해 그리드 데이터베이스에 캐쉬 관리자를 추가하여 확장된 그리드 데이터베이스에 대한 구조를 설명한 그림이다. 각각의 노드들은 캐쉬 테이블을 가지며, 메타 정보를 캐쉬 테이블에 저장하여 질의 전달하여 부하의 균형을 맞추어서 효율적인 질의 처리를 할 수 있다. N_1, N_2, N_3, … N_N은 각각의 노드를 나타내며 서로 네트워크를 통해 연결된다. 유저들은 자신이 원하는 데이터를 사용하기 위해 노드에 질의를 전달한다. 노드에서는 데이터가 자신의 로컬 영역에 저장되어 있어서 처리 할 수 있다면 질의를 수행한다. 만약 자신이 가지고 있지 않은 정보에 대한 데이터일 경우 메타 정보를 캐싱하여 질의를 데이터를 가지고 있는 노드로 전달한다.

캐쉬 관리자는 임의의 노드들 중에서 데이터베이스 관리자를 통해 실행되며, 각각의 노드들의 캐쉬 정보를 리스트로 만들어 캐쉬 테이블로 관리하며 여러 가지 색인 기법을 사용하여 관리 할 수 있다.

캐쉬 관리자는 방대한 양의 캐싱된 정보를 관리하기 위해 데이터베이스 관리자를 통해 처리 용량에 따라 노드를 그룹으로 나누어 관리하며, 자신이 실행된 노드를 기준으로 그룹을 결정한다. 그룹은 캐쉬 관리자가 실행된 노드에서 네트워크 응답속도를 통해 빠른 응답속도를 갖는 임의의 노드들을 그룹으로 묶어 관리한다.

3.2 캐쉬 관리를 위한 데이터 구조

캐쉬 관리자는 각각의 노드에서 캐쉬의 변경이 필요한 경우 발생하는 메시지를 이벤트라고 정의 한다. 노드에서는 이벤트가 발생할 경우 자신의 그룹내의 캐쉬 관리자에게 이벤트 메시지를 전달하여 처리한

다. 이벤트의 발생과 처리 과정은 3.3절에서 자세히 살펴본다.

캐쉬 관리자는 이벤트가 발생한 경우 메타 데이터베이스를 통해 메타 정보를 받는다. 메타 정보를 받은 후 복제본을 갖고 있는 여러 노드들 중에서 최적의 질의 처리 성능을 갖는 노드의 메타 정보를 이벤트를 요청한 노드에 전송한다.

캐쉬 관리자는 이벤트를 요청한 노드의 캐쉬에 최적의 성능을 가질 수 있는 메타 정보를 캐싱하기 위해 각각의 노드에서 시스템의 정보를 얻는다. 노드에서 이벤트가 요청되면, 캐쉬 관리자는 메타 정보 리스트를 얻는다. 그 후 캐쉬 관리자는 메타 정보 리스트를 통해 각각의 노드들의 시스템 정보를 얻어오며, 시스템 정보를 통해 최적의 노드를 선택한다.

그림 4는 캐쉬 관리자를 통해 최적의 노드를 선택하기 위해서 각각의 노드들로부터 얻는 노드 정보 데이터를 나타낸 그림이다.

CPU는 각각의 노드에 있는 CPU의 평균 사용량을 나타내는 것이다. TP(Transaction Processing)는 현재 노드에서 처리된 트랜잭션의 처리량을 나타내며, 마지막으로 NC(Network Cost)는 네트워크의 통신 비용을 나타낸다.

캐쉬 관리자는 저장되어 있는 시스템 정보 그림 4와 같은 데이터 구조를 통해 최적의 메타 정보를 선택 한 후 메타 정보가 필요한 노드로 캐싱할 정보를 전송한다.

그림 5는 노드에서 저장하고 있는 캐쉬 테이블 구조이다. 노드에서는 캐쉬 테이블 구조에 있는 매핑 테이블의 레코드들을 통해 빈번히 사용되는 데이터를 효율적으로 질의 처리가 가능한 노드로 전달한다.

그림 5에서 그룹 ID와 캐쉬 관리자 주소는 헤더에 포함되는 자료구조이다. Group ID는 캐쉬 관리자를 구분하기 위한 그룹 번호이고, 캐쉬 관리자 주소는 캐쉬 관리자가 있는 노드의 주소이다. 이 구조를 통해 노드에서 이벤트 요청 시에 자신이 속한 관리자로 메시지를 전송한다.

매핑 테이블의 한 레코드는 테이블 이름과 노드 주소로 구성되며 자신이 가지고 있는 메타 정보의 이름과 주소를 나타낸다. 이 구조를 통해 요청 질의에 대한 데이터를 찾고 원하는 데이터가 있는 주소로 질의를 전달한다.

평균 CPU 사용량 (CPU)	평균 트랜잭션 처리량 (TP)	네트워크 비용 (NC)

그림 4. 노드의 시스템 부하에 대한 정보 데이터

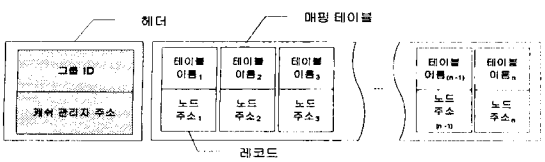


그림 5. 캐쉬 테이블

3.3 부하 분산 알고리즘

부하분산 알고리즘은 3.2절에서 설명한 시스템의 정보를 통해 부하의 균형을 유지하여 부하 분산을 할 수 있다. 본 절에서는 그림 6에서 설명하는 노드에서의 이벤트 발생과 이벤트를 처리하는 캐쉬 관리자의 순서도를 통해 각각의 알고리즘에 대하여 설명한다.

그림 6은 이벤트를 요청하는 노드와 요청된 이벤트를 처리하는 캐쉬 관리자의 순서도 이다.

그림 6의 순서도의 흐름을 따라서 각 노드에서 이벤트가 발생하는 과정을 3.3.1 절에서 설명하고, 3.3.2 절에서는 캐쉬 관리자를 통한 최적화 노드의 선택 방법에 대하여 설명한다. 3.3.3절에서는 캐쉬 관리자에서의 처리 과정인 과부하 노드의 처리 과정에 대하여 설명하고, 3.3.4절에서는 캐쉬 미스의 처리 과정에 대하여 살펴 본다. 노드에서의 이벤트 발생에서 이벤트 발생 알고리즘이 실행되며, 전달된 이벤트를 통해 캐쉬 관리자를 통한 최적화 노드 선택 방법이 선택된다. 3.3절의 알고리즘은 그림 6의 순서도에 기반하여 동작된다.

3.3.1 이벤트 발생

이벤트는 빈번히 사용되는 데이터의 메타 정보를 캐쉬에 저장하기 위해 발생하는 메시지이다. 노드는 자신의 노드를 모니터링 하며, 사용자의 요청으로 빈번히 사용되는 데이터에 접근하는 질의가 임계치를 초과 하였을 경우 그 데이터의 메타 정보를 캐쉬에 저장하기 위해 이벤트 메시지를 발생한다. 발생된 이벤트 메시지는 캐쉬 관리자로 전달되어 처리된다.

[알고리즘 3-1]은 노드에서의 이벤트 발생과정 알고리즘에 대하여 설명한다.

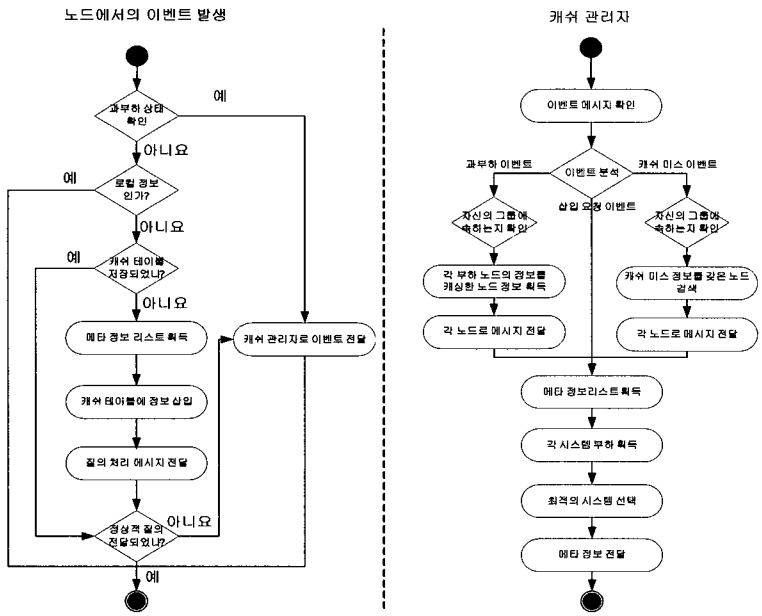


그림 6. 노드와 캐쉬의 이벤트 처리 과정 순서도

〔알고리즘 3-1〕 노드에서 이벤트의 발생과정

```
Algorithm EventCreate( pData )
Input
    pEvent      :     입력된 이벤트 요청 데이터
Output
    Meta        :     요청된 데이터에 필요한 메타 정보를 반환
    ERR_NO_VALUES :   요청한 데이터가 없는 것으로 판단
    ERR_SEND_MSG :    네트워크 오류 메시지
    SUC_SEND_MSG :    요청 데이터를 로컬 영역에서 처리할 수 없어서 다른 노드로 전송
Variables
    CacheInfo   :     자신의 캐쉬에 저장된 데이터
    CacheTable  :     캐쉬된 정보들을 관리하는 테이블

Begin
01 : Meta := FindLocalCatalog(pData);
02 : If(Meta = NULL) End
03 :    CacheInfo := CacheTable.find(CacheInfo);
04 :    If(CacheInfo = NULL) then
05 :       Meta := GetMetaListFromMDB(pData);
06 :       If(Meta = NULL) then
07 :          Return ErrHandling(ERR_NO_VALUES);
08 :       Else
09 :          If (SendtoMsg(pData,Meta) = FALSE ) then
10 :             Return ErrHandling(ERR_SEND_MSG);
11 :          END if
12 :          CacheTable.Input(Meta.NodeAddress);
13 :          If (SendtoMsg(Meta,CacheTable.NodeAddress) = FALSE ) then
14 :             Return ErrHandling(ERR_SEND_MSG);
15 :          End if
16 :             Return SUC_SEND_MSG;
17 :       End if
18 :    Else
19 :       If (SendtoMsg(pData,CacheInfo.NodeAddress) = FALSE) then
20 :          Return ErrHandling(ERR_SEND_MSG);
21 : End if
22 : Else
23 :    Return Meta;
24 : End if
End
```

줄 01 에서는 요청된 데이터가 자신의 로컬 영역 내에 존재하는지 확인하는 함수이다. 만약 자신의 로컬 영역 내에 존재 하는 함수라면, 줄 23 을 통해 요청 데이터의 정보를 반환한다. 줄 02 에서 줄 21 까지는 로컬 영역에 메타 정보가 없을 경우를 설명하는 코드이다. 줄 02 에서 줄 03 까지는 자신의 캐쉬 테이블을 검사하여 요청 데이터의 정보를 확인하며, 캐쉬 테이블에 정보가 있다면 데이터를 줄 19 에서 줄 20 의 코드를 통해 해당 노드로 질의를 전달한다. 캐쉬 테이블에 요청 데이터의 정보가 없다면 줄 04 에서 줄 06 까지 코드를 통해 메타 데이터베이스로부터 메타 정보 리스트를 가져오며, GetMetaListFromMDB() 는 메타 데이터베이스에 접속하여 메타 정보를 가져오는 함수 이다. 메타 정보 리스트가 없다면 잘못된 정보 데이터이다. 캐쉬 관리자를 통해 메타 정보를 캐싱하기 위해서는 처리 시간이 필요함으로 줄 09 와 줄 10 을 통해 요청 데이터를 먼저 메타 정보 리스트를 보고 전달한다. 줄 13 에서 줄 15 는 캐싱된 메타 정보를 캐쉬 관리자에게 전송하고 오류 메시지를 처리하기 위한 코드이며, 줄 16 은 성공 메시지를 처리하는 코드 이다. 캐쉬 관리자에게 메시지를 전달한 후 줄 19 에서 줄 21 을 통해 자신의 캐쉬 테이블에 저장된 정보를 통해 원본 데이터가 있는 노드로 메시지를 전달한다. 메시지 전달이 성공 하였으면 성공

메시지를 반환하며, 전달이 실패 하였을 경우 오류 메시지를 반환한다.

3.3.2 캐쉬 관리자를 통한 최적화 노드 선택 방법

효율적인 질의 처리를 하기 위해 노드에서는 이벤트 메시지를 캐쉬 관리자에게 전달한다. 이벤트를 받은 캐쉬 관리자는 메타 데이터베이스를 통해 각 복제본 데이터의 메타 정보 리스트를 받는다. 그리고 메타 정보 리스트를 통해 복제본 데이터가 있는 각각의 노드의 시스템 정보를 얻는다. 이질적인 그리드 데이터베이스 환경에서는 각각의 노드의 시스템 환경이 다르기 때문에 각각의 노드 시스템 가용성을 비교해야 한다. 수식 (3-1)과 수식 (3-2)는 각각의 시스템 정보들 중에서 통해 가장 효율적인 노드를 선택한다. 각 그리드 데이터베이스는 자신의 노드를 모니터링 하고 있으며 각각의 스케줄러를 통해 초당 트랜잭션의 실행 정보 등을 알 수 있다.

수식 (3-1)은 다른 이질적인 시스템들 중에서 노드의 가용성을 계산하는 수식이다. 각각의 노드들은 이질적인 시스템으로 평균 CPU 사용량 대비 시간 t를 통해 자신의 노드의 가용성을 나타낼 수 있다. 평균 CPU 사용량은 각기 시간 t 동안 자신의 노드에서 백분율(%)로 나타나며 C로 명명한다. 평균 트랜잭션 처리량은 t 시간 동안 처리된 트랜잭션의 개수를 나타내며 TP 으로 명명된다. 마지막으로 NC는 네트워크의 응답 시간을 나타낸다.

TPC는 CPU의 사용량과 트랜잭션의 처리량으로 계산된 초당 트랜잭션 처리량 이며, WL은 각각의 노드들이 작업 부하를 수치로 나타낸 값이다.

$$TPC_{unit} = (1 - C_t) \bullet TP_t \qquad \text{수식 (3-1)}$$

수식 (3-1)은 각 노드들의 초당 트랜잭션 처리량을 나타내는 수식이다. 초당 처리되는 트랜잭션 처리량을 통해 하나의 트랜잭션이 처리되는 시간을 나타낸다.

$$WL_n = \frac{1}{TPC_{unit}} + NC_n \qquad \text{수식 (3-2)}$$

수식 (3-2)는 노드의 작업 부하를 나타내는 수식이다. 수식 (3-1)에서 나타난 트랜잭션의 개수를 통해 하나의 트랜잭션으로 나타내며, 하나의 트랜잭션 값과 네트워크 비용을 합하여 노드의 부하 값을 계산하여 여러 노드들 중 최적의 노드의 정보가 노드에 캐싱된다. 수식 (3-2)을 통해 각각의 노드들 중에서 최적의 노드를 선택한 후 실제적으로 메타 정보를 캐싱하려는 노드로 정보를 전송한다.

[알고리즘 3-2]은 노드에서 받은 이벤트 메시지를 처리하는 과정을 설명하는 알고리즘 이다.

[알고리즘 3-2] 캐쉬 관리자에서 최적화 노드 선택과정

```
Algorithm OptimaChoose ( Meta , RequestNode )
Input
    Meta              : 노드로부터 요청된 메타 정보
    RequestNode       : 이벤트를 요청한 노드 정보
Output
    ERR_NO_VALUSE     :  요청한 데이터가 없음
    ERR_SEND_MSG      :  네트워크 오류 메시지
    SUC_SEND_MSG      : 최적의 메타 정보를 요청 노드로 전송 후의 성공 메시지
Variables
    MetaList          : 메타 데이터베이스로부터 얻어온 복제본 메타 정보 리스트
    SystemInfoList    :  각 노드들의 부하 정보
    MetaInfo          : 최적의 부하 정보를 갖는 노드의 메타 정보

Begin
   01 : MetaList := FindToGroupTable(Meta);
   02 : If(MetaList = NULL) then
   03 :    MetaList := GetMetaListFromMDB(Meta);
   04 :    If(MetaList = NULL) then
   05 :       Return ErrHandling(ERR_NO_VALUSE);
   06 :    End If
   07 : End If
   08 : SystemInfoList := GetSystemInfo(MetaList);
   09 : For i = 0 to i < SystemInfoList.Count step i++
   10 :  MinSystemInfo = Sorting(MinSystemInfo,SystemInfoList[i])
   11 : End For
   12 : MetaInfo = SystemToMetaInfo(MinSystemInfo,MetaList)
   13 : If(SendtoMsg(MetaInfo,RequestNode.Address) =FALSE) then
   14 :  Return ErrHandling(ERR_SEND_MSG);
   15 : End If
   16 : If(SendtoMsg(MetaInfo,MetaInfo.CacheAddress) =FALSE) then
   17 :  Return ErrHandling(ERR_SEND_MSG);
   18 : End If
   19 : Return SUC_SEND_MSG;
End
```

캐쉬 관리자로 이벤트가 요청된 경우 사용 요청 데이터가 캐쉬 그룹 테이블 안에 저장된 메타 정보에 있는지 확인하기 위해 줄 01 에서 줄 02 까지 수행한다. 만약 캐쉬 그룹 테이블에 저장된 메타 정보 리스트가 없다면 줄 03 을 수행하여 메타 데이터베이스로부터 메타 정보 리스트를 얻는다. 줄 04 에서는 메타 정보 리스트를 얻지 못하였을 경우 값이 없다는 ERR_NO_VALUSE 메시지를 반환하고 함수를 종료 한다. GetSystemInfo() 함수를 통해 줄 08 에서는 메타 정보 리스트의 주소를 이용하고, 각 노드의 부하 정보를 얻는다. 얻어온 각각의 노드들의 부하 정보를 통해 부하가 가장 적은 노드의 정보를 얻기 위해 줄 09 에서 줄 11 까지 코드를 수행한다. 줄 10

에서 동작되는 Sorting() 함수는 수식 (3-1)과 수식 (3-2)를 통해 가장 작은 부하를 얻는 함수이다. Sorting() 함수를 통해 줄 12 에서는 메타 정보 리스트로부터 시스템의 부하가 가장 적은 노드의 메타 정보를 MetaInfo 로 얻는다. 줄 13 에서 줄 15 까지는 이벤트를 요청한 노드로 최적의 메타 정보를 전달하며, 전달하지 못할 경우 오류 메시지를 반환한다. 과부하 노드의 처리 및 캐쉬 미스 현상을 해결하기 위해 줄 16 에서 줄 18 까지는 원본 데이터를 가진 그룹의 캐쉬 관리자로 메시지를 전달한다.

3.3.3 시스템 과부하 처리

노드에 캐싱된 정보를 통한 질의뿐만 아니라 여러 노드에서 계속적으로 발생하는 질의의 집중으로 과부하가 발생한다. 노드는 계속적으로 각자 자신의 노드를 모니터링 하며 일정 수준 이상의 과부하 발생시 자신의 과부하 상태에 대하여 캐쉬 관리자에게 전달한다. 과부하 상태를 전달 받은 캐쉬 관리자는 더 이상 다른 노드에서 과부하 노드의 메타 정보를 사용하지 못하게 변경하여, 과부하 노드의 부하를 분산한다.

[알고리즘 3-3]은 노드에서 과부하 요청을 한 경우 캐쉬 관리자에서 과부하 요청을 처리하는 과정을 나타낸다.

자신의 노드에서 과부하가 발생 하였을 경우 자신의 그룹에 해당하는 캐쉬 관리자에게로 과부하 메시지를 전달한다. 과부하 노드가 자신의 그룹에 속하는 노드인지 아닌지를 확인하기 위해 줄 01 에서 03 까지 코드를 통해 확인한다. GroupNodeList() 함수는 캐쉬 관리자를 통해 그룹으로 연결된 노드들의 정보가 저장된다. 줄 04의 NodeToMetaInfo() 함수는 과부하 노드에 저장된 데이터들의 정보를 캐싱하는 노드들의 정보가 저장되어 있다. NodeToMetaInfo() 함수를 통해 과부하 노드의 데이터 정보를 캐싱하고 있는 노드들의 캐싱된 정보 메타 리스트인 CachedMetaList를 얻는다. CachedMetaList 값이 NULL 값이면 SECCESS 메시지를 줄 05 에서 줄 07 을 이용하여 반환하며, SUCCESS 는 과부하 노드의 데이터 정보를 다른 노드에서 캐싱하고 있지 않다는 것을 의미한다. 과부하 노드의 데이터를 캐싱하여 사용하고 있는 노드들의 정보를 변경하기 위해 줄 08 에서 줄 12 까지 코드를 이용하여 메시지를 전달하는 과정을 보인다. 과부하 노드의 정보를 저장하고 있는 줄 08 에서 메타 정보의 개수 만큼 반복하여, 각 캐쉬 관리자에게 새로운 캐쉬 정보를 줄 09 를 이용하여 찾을 것을 전달한다. 모든 과정을 마친 후 성공 메시지를 반환하는 과정이 줄 13 을 통해 나타난다.

(알고리즘 3-3) 캐쉬에서의 과부하 처리과정

```
Algorithm HighWorkLoad ( HotNode )
Input
    HotNode          :      과부하 노드로 부터 들어온 정보
Output
    ERR_NOT_GROUPNODE :     자신에 그룹에 속하는 노드가 아닌 오류 메시지
    SUCCESS          :      함수 성공 실행
Variables
    CachedMetaList   :      자신의 노드를 캐싱하는 노드들의 정보

Begin
    01 : If(GroupNodeList(HotNode) = FALSE) then
    02 :   Return ErrHandling(ERR_NOT_GROUPNODE);
    03 : End If
    04 : CachedMetaList := NodeToMetaInfo( HotNode )
    05 : If(CachedMetaList = NULL) then
    06 :   Return SUCCESS;
    07 : End If
    08 : For i = 0 to i < CachedMetaList.Count step i++
    09 :  If(SendtoMsg(CachedMetaList[i].Meta.    CacheMetaList[i].CacheAddress)    =
          FALSE) then
    10 :    ErrHandling(ERR_SEND_MSG);
    11 :  End if
    12 : End For
    13 : Return SUCCESS;
End
```

3.3.4 캐쉬 미스 처리

본 절은 캐쉬 미스로 인한 비일관성으로 여러 노드에서 질의가 잘못 전달 되어 네트워크 비용이 증가하는 문제를 해결하는 방법을 설명한다. 여러 노드에서 복제본 데이터의 메타 정보를 저장하고 있을 때, 원본 데이터가 변경 되었을 경우 변경되지 않은 메타 정보를 가진 캐쉬를 통해 질의가 타 노드로 잘못 전달된다. 이렇게 잘못 전달되는 질의의 반복으로 네트워크 속도가 저하되며 처리 성능이 저하된다.

캐쉬 관리자는 자신의 그룹에 속한 노드들의 캐싱된 정보를 저장하고 있으며, 이러한 정보를 통해 여러 노드에서 반복적으로 캐쉬 미스 현상이 발생하는 것을 방지한다.

[알고리즘 3-4]은 캐싱된 데이터를 통해 반복적인 캐쉬 미스 문제를 해결하는 과정이다.

[알고리즘 3-4]는 노드에서 캐쉬 미스가 발생하여 캐쉬 미스 이벤트를 캐쉬 관리자에게 전달한 후의 과정을 나타낸다. 캐쉬 미스된 메타 정보의 그룹과 현재 자신의 캐쉬 그룹이 같은지를 줄 01 을 이용하

(알고리즘 3-4) 캐쉬 미스 처리 과정

```
Algorithm CahceMiss ( MissMetaInfo )
Input
    MissMetaInfo      :      캐쉬 미스된 메타 정보
Output
    ERR_SEND_MSG      :      네트워크 오류 메시지
    SUCCESS           :      함수 성공 실행
Variables
    CacheMgr          :      현재 캐쉬 관리자 정보
    MissMetaList      :      잘못된 캐싱 정보를 가진 노드들의 정보

Begin
    01 : If (MissMetaInfo.Group != CacheMgr.Group) then
    02 :    If(SendtoMsg(MissMetaInfo, MissMetaInfo.CacheAddress)=FALSE) then
    03 :        Return ErrHandling(ERR_SEND_MSG);
    04 :    End if
    05 : End if
    06 : MissMetaList = GetGroupMetaInfo(MissMetaInfo);
    07 : For i = 0 to i < MissMetaList.Count step i++
    08 :    If(SendtoMsg(MissMetaList[i].Meta, MissMetaList[i].CacheAddress) = FALSE)
         then
    09 :        ErrHandling(ERR_SEND_MSG);
    10 :    End if
    11 : End for
    12 : Return SUCCESS;
End
```

여 확인한다. 줄 01 의 확인을 통해 자신의 그룹에 있는 노드의 원본 데이터가 변경되어 캐쉬 미스가 발생하는지를 확인 할 수 있다. 만약 그룹이 같지 않다면 줄 02 에서 줄 03 을 통해 원본 데이터가 변경된 노드가 있는 캐쉬 관리자에게 캐쉬 미스를 전달한다. 자신의 그룹 안에 있는 메타 정보 리스트를 통해 원본 데이터가 변경된 데이터의 메타 정보를 캐싱하고 있는 노드들에서 정보를 얻기 위해 줄 06 를 이용한다. MissMetaList 는 잘못된 캐싱 정보를 가지고 있는 노드들의 정보이다. 줄 07 에서 줄 11 까지는 잘못된 캐싱 정보를 가진 노드들이 속한 캐쉬 관리자에게 전달하는 메시지 이다. 캐쉬 미스의 오류로 여러 노드에서 계속적으로 발생하는 문제를 방지하기 위해 각각의 캐쉬 관리자에게 잘못된 오류 전달 메시지를 전달한다. 줄 07 에서의 반복 숫자만큼 각각의 캐쉬 관리자에게 캐쉬 미스의 메시지를 줄 08 과 09 를 통해 전달한다.

4. 캐쉬 관리 기반의 부하분산 기법

본 장에서는 제안 기법의 평가를 위한 실험 환경에 대해 설명하고, 클라이언트 수, 질의 집중도 등의 환경 변화에 따른 제안 기법과 기존 기법과의 성능 비교를 수행한다.

4.1 실험환경

본 논문에서는 제안 기법의 평가를 위해 C/C++언어 기반 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 서버에 제안하는 캐쉬의 자료구조를 저장했다. 제안 기법의 비교 평가 대상이 되는 기존의 기법의 구성을 위해 링 기반 연결 구조를 이용한 부하 분산 기법과 데이터의 이주와 복제를 통한 부하를 분산 기법을 위해 각각의 기능을 구현했다. 구현을 위해 MS Visual C++ 6.0을 사용했다. 전체적인 구성은 그리드 환경에 적합하도록 복제본의 사용을 허용 하였으며, 이질적인 분산 환경에서의 성능 평가를 위해 여러 노드에서의 각기 다른 큐 사이즈와 질의 처리 속도 등등을 범위를 두어서 평가했다.

실험 환경에서 작업 부하 최대값은 각 노드의 대기열의 길이를 사용한다. 대기열의 길이가 길어질수록 작업 대기시간이 길어져 전체적인 처리 성능이 저하되는 것을 알 수 있다. 대기열의 길이는 1~20개의 범위를 두어 설정하였으나, 실제 실험평가에서 대기열의 길이에 따른 결과값의 변화가 미미하여 이에 대한 평가는 생략했다.

실험을 위한 비교 평가 대상은 첫 번째로 링 기반 구조를 이용한 부하 분산 기법으로 노드의 사용량 임계치를 추가하면 다음 복제본을 가진 노드로 질의를 전달하는 방식이다. 두 번째 방식은 데이터의 복제와 이주를 통한 부하 분산 기법으로 사용 임계치가 일정 수준 이상으로 올라가면 데이터를 이주 하거나 복제하여 부하를 분산 하는 기법이다. 그리고 제안 기법은 본 논문에서 제안하는 구조를 그대로 구현하여 사용하였으며, 수식 (3-1)과 수식 (3-2)를 통해 최적의 노드를 얻는 알고리즘을 구현했다. 본 논문에서는 위와 같이 복제본 데이터를 링 기반 연결 구조

표 1. 실험 환경

실험 요소	실험 데이터	실험 요소	실험 데이터
실험 시간	30000 unit	질의 처리 시간	4~7 unit
서버 수	20~400 개	질의 전송 시간	2~4 unit
클라이언트 수	100~900 개	질의 입력 간격	0.1 unit
데이터 개수	1000 개	복제본 개수	10~250 개

를 통해 연결하여 부하 분산 하는 기법과 데이터의 이주 및 복제를 통해 부하를 분산하는 방법 통해 제안기법의 비교 평가를 수행한다.

노드 단위로 적용되는 기존의 기법들과의 성능 비교를 위해서는 데이터의 분포 상태와 사용자 질의 패턴에 큰 영향을 받기 때문에 이를 고려한 성능평가가 필요하다. 이에 대한 연구는 현재 실험 중에 있으며, 향후 연구에서 다루도록 한다.

4.2 성능평가

본 실험 평가에서는 클라이언트의 증가에 따른 처리 성능 변화, 복제본 데이터 증가의 따른 처리 성능 변화 그리고 질의 집중 비율 변화에 따른 처리 성능을 주요 평가 인자로 보고 이것을 집중적으로 평가했다. 모든 결과는 사용 요청 질의를 보내고 처리된 후의 질의 처리량으로 나타내었다.

그림 7에서는 클라이언트 개수의 변화에 따른 처리 성능의 변화를 나타낸다. 질의는 전체 노드로 고르게 분포되어 전송되며, 동일한 복제본의 데이터의 개수를 가지고 실험평가를 진행했다.

그림 7에서 보면 질의 처리 성능 중 제안 기법이 가장 효과적이며, 그 다음으로 데이터 이주와 복제, 링 기반 연결 구조 기법 순으로 나타났다. 링 기반 연결 구조 기법은 데이터의 요청이 빈번하더라도 시스템의 부하가 높지 않다면 부하 분산을 하지 않으며, 부하 분산이 필요한 경우에도 부하를 분산하기 위해 다른 노드로 질의를 전달할 때 다른 노드의 부하 상황을 고려하지 못하고 자신의 부하만을 줄이려고 부하를 분산하기 때문에 처리 성능이 가장 낮게 나타났다. 데이터의 이주와 복제 기법은 노드에서 부하가 발생하면 자주 사용되는 노드로 데이터를 이주하거나 데이터를 복제하여 빈번한 데이터의 사용에는 질의 전달이 필요 없지만 최적의 노드에서 질의 처리 하는 것이 아니기 때문에 제안 기법에 비하여 성능이 저하된 것으로 판단된다. 본 제안 기법은 캐쉬 관리자를 통해 이질적인 시스템들 사이에서 여러 노드들의 부하 정보를 통해 최적의 노드의 데이터 정보를 캐싱하여 처리량이 가장 많은 것을 보였다.

본 절에서는 복제본 데이터 증가에 따른 처리 성능을 측정했다. 그림 8은 질의 처리시에 원본 데이터의 개수를 점차 증가하여 나타나는 성능 측정 그래프이며, 각각의 클라이언트의 수는 동일하다.

그림 8에서 데이터 복제본의 개수가 증가 할수록 각 기법들의 성능이 좋게 나타났다. 모든 기법이 성능이 좋게 나온 것은 복제본이 증가 할수록 부하가 집중되지 않고 여러 노드로 분산되기 때문이다. 제안 기법은 복제본 노드들의 메타 정보를 통해 질의 처리를 할 경우 최적의 성능을 낼 수 있는 노드의 메타 정보를 캐싱하여 질의를 처리 하기 때문에 가장 좋은 성능을 나타냈다. 링 기반 연결 구조를 이용한 부하 분산 기법은 복제본의 개수가 적은 상태에서 자신의 노드의 부하만을 고려하여 정방향 연결을 따라 부하를 분산하여 다른 노드의 부하를 예측하지 못한 질의 전달로 처리 성능이 가장 낮았다. 데이터의 복제와 이주를 통한 부하 분산 기법은 빈번히 사용되는 데이터의 복제본 데이터가 부족할 경우 복제본 데이터를 복제 하거나 이주하여 부하를 분산하는데 데이터의 용량이 큰 경우 데이터 복제 및 이주를 할 경우의 비용이 크며, 과부하가 발생 하였을 경우 자신의 주변 노드로 부하를 분산하여 전체 시스템에서의 최적의 질의 처리가 가능한 노드를 선택 할 수 없기 때문에 링 기반 연결 구조를 이용한 부하 분산 기법 보다는 처리량이 많으며, 제안 기법 보다는 처리 성능이

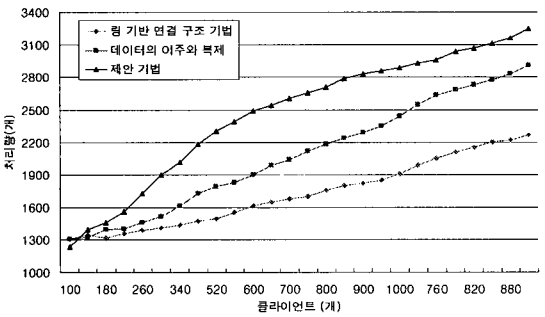


그림 7. 클라이언트 수 증가율에 따른 처리 성능 변화

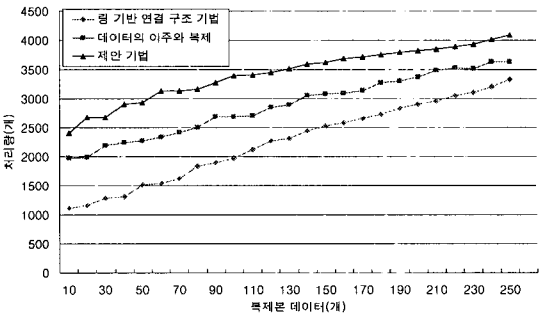


그림 8. 복제본 데이터 증가에 따른 처리 성능 변화

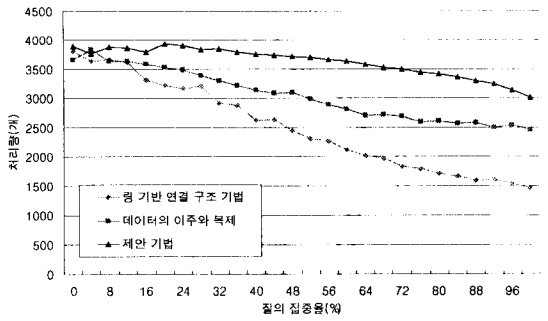


그림 9. 질의 집중 비율에 따른 처리 성능 변화

낮게 나타났다.

그림 9는 전체 복제본에 대한 질의 집중 비율에 따른 처리 성능 변화를 나타내는 성능 평가이며, 복제본의 개수와 클라이언트의 개수는 동일하다.

그림 9에서 각각의 기법은 전체 복제본 노드의 질의 집중 비율이 증가 할수록 성능이 저하되었지만, 링 기반 연결 구조 기법과 데이터 이주와 복제를 통한 부하 분산 기법에 비하여 좋은 성능을 나타내었다. 링 기반 연결 구조를 이용한 부하 분산 기법은 복제본 데이터를 정형화된 링 기반 형태로 연결하여 자신의 노드에서 과부하가 발생하였을 경우 질의를 복제본 데이터를 갖고 있는 다른 노드로 전달한다. 이때 링 기반 연결 구조는 질의를 처리 할 수 있는 최적의 노드를 찾는 것이 아니라 이미 연결된 복제본 데이터를 갖고 있는 노드로 질의가 전달되어 가장 낮은 처리 성능을 나타냈다. 데이터의 이주와 복제를 이용한 부하 분산 기법은 질의 집중률이 증가 할수록 빈번히 사용되는 데이터를 이주 및 복제를 통해 질의가 집중되는 현상을 분산했다. 하지만 자신에게 사용되는 질의 집중 현상을 방지 하였음에도 불구하고 시스템 전체적으로 최적의 노드를 찾아서 질의 처리를 수행하지 않아서 제안 기법 보다는 낮은 성능을 나타냈으며, 링 기반 연결 구조 보다는 높은 성능을 나타내었다. 제안 기법은 질의 집중 비율이 높아 질수록 처리 성능이 안 좋아졌으나, 본 논문에서 제안한 최적의 노드 선택 기법을 사용하여 질의를 처리할 수 있는 최적의 노드를 선택하여 질의를 처리하여 가장 좋은 성능 나타났다.

5. 결론 및 향후 연구

제안 기법은 여러 복제본들이 있는 노드에 질의가 집중되는 현상이 발생하여서 시스템의 과부하가 발생하는 문제점을 해결하는 방법을 캐쉬 관리를 통해 제시했다.

노드들의 사용되는 캐쉬를 관리 하기 위해 캐쉬 관리자를 사용한다. 노드에서는 자주 사용되는 데이터를 통해 이벤트가 발생하였을 경우 이벤트를 캐쉬 관리자에게 전달한다. 이벤트를 전달 받은 캐쉬 관리자는 메타 데이터베이스를 통해 메타 정보 리스트를 받아온다. 캐쉬 관리자를 통해 여러 노드들의 시스템 정보를 분석하여 최적의 메타 정보를 캐싱한다. 그리고 각 노드들에게는 새로운 최적의 메타 정보를 재전송하여 캐싱된 메타 정보를 통해 질의를 처리하여 기존의 기법에 비해서 보다 빠르고 안정적인 성능을 나타내며, 기존의 기법에 비하여 약 10%~35% 까지의 성능이 향상됨을 보였다.

제안 기법은 노드의 추가와 삭제가 자유롭고 다른 노드와의 연결 구조가 동적이며, 노드의 수가 방대한 환경에서 유용하게 사용될 수 있다. 캐쉬 관리 기반의 부하 분산 기법으로 여러 분산 환경의 시스템 및 어플리케이션에도 적용을 할 수 있다.

향후 연구로는 각종 응용 환경에서 네트워크 구성과 그리드 데이터베이스에서의 캐쉬에 대한 서로 다른 기법 간의 호환성에 대한 연구 및, 이질적인 시스템들의 성능을 분석하기 위한 보다 정교한 성능 분석 기법이 필요하다.

참 고 문 헌

[1] S. Malaika, A. Eisenberg and J. Melton, "Standards for database on the grid," *ACM SIGMOD Record archive*, Vol.32, Issue.3, pp. 92-100, 2003.

[2] 신승선, 장용일, 이순조, 배해영, "그리드 데이터베이스에서 질의 전달 최적화를 위한 캐쉬 관리 기법," *한국멀티미디어학회 논문지*, Vol.10, No. 1, pp. 13-25, 2007.

[3] A. B. M. Russel and A.I. Khan, "Towards dynamic data grid framework for research," *In Proc. Fourth Australasian Symposium on Grid Computing and e-Research (AusGrid 2006), Hobart, Australia. CRPIT*, *54*. Buyya, R. and Ma, T., Eds., ACS. pp. 9-16,

2006.

[4] J. Gray, P. Helland, P. O"Neil, and D. Shasha, "The dangers of replication and a solution," *Proc. ACM SIGMOD Conference*, pp. 173-182, 1996.

[5] Z. Lan, V. E. Taylor, and GregBryan, "Dynamic load balancing of SAMR applications on distributed systems," *Scientific Programming*, pp. 319-328, 2002.

[6] Y. I. Jang, H. S. Kim, S. Y. Park, J. D. Lee, and H. Y. Bae, "Extendible hashing based recovery method in a shared-nothing spatial database cluster," *ICCSA 2006, LNCS 3983*, pp. 1126-1135, 2006.

[7] A. Mondal, K. Goda, and M. Kitsuregawa, "Effective load-balancing via migration. and replication in spatial GRIDs," *Proc. DEXA*, 12. 2003.

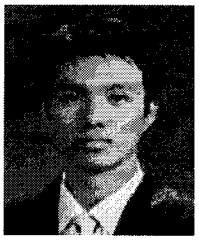
[8] S. E. Brastsberg and R. Humborstad, "Online scaling in highly availabe database," *Proceedings of the 27th VLDB Conference*, 2001.

[9] M. N. Alpdemir, A Mukherjee, N. W. Paton, P. Watson, A. A. A. Fernandes, A. Gounaris, and J. Smith. "OGSA-DQP: A service-based distributed query processor for the grid," *In Simon J.Cox, editor, Proceedings of UK e-Science All Hands Meeting Nottingham. EPSRC*, Sep. 2003.

[10] M. N. Alpdemir, N. W. Paton and A. A. A. Fernandes, "Distributed query processing in an OGSA environment," *Internal Report*, Sep. 2002.

[11] UK Database Task Force, "OGSA-DQP 3.1 user's documentation," 2006. http://www.ogsadai.org.uk/documentation/ogsa-dqp 3.1/.

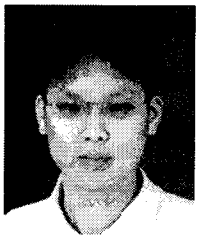
[12] Mesquite Software. Inc. CSIM19 The simulation engine, 2005, http://www.mesquite. com.



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



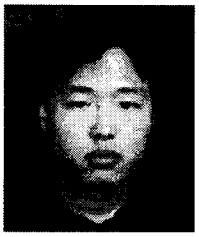
백 성 하

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

관심분야 : 데이터 스트림 관리 시스템, 데이터베이스 클러스터, 위치기반서비스

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

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

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

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