Journal of Korea Multimedia Society Vol. 9, No. 9, September 2006(pp. 1107-1117)

그리드 데이터베이스에서 링 기반 연결 구조를 이용한 부하 분산 기법

장용일[†], 신숭선[††], 박순영[†††], 배해영[††††]

요 약

본 논문에서는 복제 데이터 간 링 기반 연결 구조를 이용한 부하 분산 기법을 제안한다. 그리드 데이터베이스에서는 일반적으로 각 노드의 데이터가 처리 성능과 가용성 향상을 위해 서로 다른 위치에 복제되어 저장되고, 사용자 질의는 목적 데이터를 포함하는 노드로 전송된 후 처리된다. 그러나 이러한 환경에서는 작업 부하의 불균형으로 인한 성능 저하가 나타날 수 있다. 기존 연구는 노드의 수가 많고 사용자 질의가 유동적으로 변하는 그리드 데이터베이스에는 적용하기 힘들다. 제안 기법은 각각의 동일 복제본이 포함된 노드들을 하나의 링 구조로 연결한다. 노드의 작업 부하가 한계를 넘게 되면, 이후 입력되는 질의의 목적 데이터에 따라 다음 연결 노드로 질의를 전송한다. 그리고 이 노드는 작업 부하가 적어질 때까지 새로운 질의를 입력받지 않는다. 이후, 이전 노드로 메시지를 전송하여 이 노드로 질의가 전달되지 않도록 연결 구조를 변경한다. 제안 기법은 성능평가를 통해 기존 방식에 비해 다수의 노드를 포함하면서 구조가 동적으로 변화하는 환경에서 기존 기법보다 우수한 성능을 보인다.

A Load Balancing Method Using Ring Network Structure in the Grid Database

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

ABSTRACT

In this paper, a load balancing method using ring network structure in the Grid database is proposed. In the Grid database, generally, data is replicated for performance and availability. And, user's request is transferred to node and processed in that node which has target data. But, in such environment, a decline of performance can be occurred because unbalanced workload. A traditional research is proposed to solve unbalanced load problem. However, the Grid database has a number of systems and user's request always changes dynamically. Therefore, a traditional research can not be applied. The proposed method connects each node which has a same replicated data through ring network structure. If workload is overflowed in some node, user's request is transferred to a linked node which has a target data. And, this node stops another request processing until workload is decreased. Then, it changes the link structure through sending a message to a previous node, to stop request forwarding from a previous node. This paper shows a proposed method increases performance than existing research through performance evaluation and is more suitable for a complex and dynamic environment.

Key words: Grid Database(그리드 데이터베이스), Load Balancing(부하 분산), Ring Network Structure (링 구조), Data Replication(데이터 복제)

※ 교신저자(Corresponding Author) : 배해영, 주소 : 인천광역시 남구 용현동 253(402-751), 전화 : 032)860-8712, FAX : 032)862-9845, E-mail : hybae@dblab.inha.ac.kr
접수일 : 2006년 4월 4일, 완료일 : 2006년 6월 8일
[†] 준회원, 인하대학교 컴퓨터정보공학과 박사과정
(E-mail: yijang@dblab.inha.ac.kr)
[††] 인하대학교 컴퓨터정보공학과 석사과정
(E-mail: ssshin@dblab.inha.ac.kr)
[†††] 한국전자통신연구원 데이터베이스연구팀 선임연구원
(E-mail: sunny@etri.re.kr)
[††††] 인하대학교 컴퓨터공학부 교수
※본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 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 기존 분산 데이터베이스 시스템의 기능을 기본적으로 포함한다. 또한, 그리드 데이터베이스는 대용량 자원을 공유하며 고속 연산을 지원하고, 그리드 데이터베이스를 구성하는 각 노드는 데이터를 처리 성능과 가용성 향상을 위해 서로 다른 위치에 복제하여 저장하며, 데이터 연합(Federation), 데이터 변환, 데이터 배치(Allocation), 데이터 통합 등의 기능을 제공한다[10].

그리드 데이터베이스를 구성하는 각 노드는 수많은 데이터들을 포함하고 있다. 모든 데이터는 어플리케이션에 따라 그 형태와 목적이 다르지만, 가용성과 성능의 향상을 위해 다른 노드에 복제본(Replica)을 구성하게 된다. 하나의 데이터가 여러 복제본을 가지고, 최적의 성능을 낼 수 있는 노드에 각각 배치되는 것이다.

사용자 질의는 보통 여러 데이터들을 요구하게 되는데, 해당 데이터를 포함하는 노드로 질의가 전송되어 처리된다. 이 때 데이터가 저장된 노드를 카탈로그 정보를 통해 찾게 되며, 해당 데이터의 복제본까지 포함한 노드의 리스트가 결과로 얻어진다. 질의 분배를 담당하는 프로세스는 결과값 내의 노드 중 임의의 노드를 선택하여 질의를 보내게 된다.

그러나 질의 처리 노드를 선택하는 과정에서 임의 노드에 사용자 질의가 집중이 되어 처리 성능이 떨어지게 되는 결과를 초래할 수 있다. 질의 처리의 최적화를 위해 노드와 노드 사이의 네트워크 비용, 해당 노드의 현재 작업부하, 사용자 질의의 연산 처리 순서 등을 고려하여 최적의 노드를 선택할 수 있지만, 수많은 노드를 포함하는 그리드 데이터베이스 환경에 적용하기는 힘들다. 모든 노드의 작업 부하 정보를 얻는 비용이 너무나 크기 때문이다 [1,4,5].

기존에 비슷한 환경으로서 분산 시스템 또는 병렬처리 시스템들을 위한 부하 분산(load balancing) 기법들이 제안되었다. 기존 연구는 노드의 수가 많고 사용자 질의가 유동적으로 변하는 그리드 데이터베이스에는 적용하기 힘들다. 또한, 근접 노드들과의 정보 교환을 통해 부하 분산을 유도하는 연구도 제안되었다. 이 방식은 노드의 수에 구애 받지 않지만 데이터를 중심으로 부하 분산이 필요한 그리드 데이터베이스의 환경에는 적합하지 않다[3].

제안 기법은 각각의 동일 복제본이 포함된 노드들을 하나의 링 구조로 연결한다. 각각의 복제본들은 정방향 링크와 역방향 링크를 통해 서로 연결되며, 하나의 노드는 자신이 가지는 복제본의 수만큼 연결 링크들을 갖게 된다. 임의의 노드의 작업 부하가 한계를 넘게 되면 해당복제본의 정방향 링크가 가리키는 노드로 질의가 전달된다. 그리고 이 노드는 작업 부하가 적어질 때까지 다른 질의에 대한 처리를 중지한다. 또한, 역방향 링크를 통해 메시지를 보내어 자신의 이전 노드가 자신에게 질의를 전달하지 않도록 링크 구조를 수정한다. 작업 부하가 적어지면, 다시 메시지를 이전 노드로 보내어 링크 구조를 원래대로 바꾸며, 질의 처리 과정에 참여하게 된다.

제안 기법은 성능평가를 통해 기존 방식에 비해 여러 가지 면에서 성능이 향상됨을 보인다. 또한, 기존 기법보다 그리드 데이터베이스에 적합함을 보인다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구로 그리드 데이터베이스에 대해 기술하고, 기존의 부하 분산 기법들에 대해 소개한다. 3장에서는 본 논문에서 제안하는 링 기반 연결 구조와 이를 통한 부하 분산 기법에 대해 설명하고, 4장에서 제안 기법에 대한 성능 평가를 수행한다. 마지막으로 5장에서 결론을 기술한다.

2. 관련 연구

본 장에서는 그리드 데이터베이스의 기능 및 요구사항들과 이에 따른 기존 연구들을 설명하고, 기존 분산 시스템에서의 부하 분산 기법들을 설명한다.

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 단

순한 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 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.

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[7].

2.2 부하공유 기법

부하 공유 기법은 일종의 스케줄링 문제에 속하며 기법들의 특징에 따라 구분하면 정적(static)기법과 동적(dynamic)기법으로 나뉘어 진다. 정적 기법은 시스템이 진행되기 전에 미리 스케줄링을 완료하여 이후에는 각 호스트의 부하에 관계없이 정해진 스케줄링 결과대로 운영하는 기법이며, 반대로 동적 기법은 매 순간마다 그 당시의 시스템의 상태에 따라 스케줄링을 수행하는 기법이다[11].

기존의 부하 공유 기법들 중 가장 기본적이고 잘 알려져 있는 기법으로는 입찰(bidding)기법이 있고, 이 기법에서는 프로세스를 이주시키고자 하는 호스트에서 프로세스 이주 의도를 전 호스트로 브로드케스트(broadcast)하게 된다. 이 메시지를 받은 다른 호스트들은 자신의 상태와 제공할 수 있는 자원들의 여건들을 메시지에 담아 응답하게 된다. 이 기법은 간단하나 이주시킬 때마다 브로드케스트를 해야 하는 오버헤드가 있으며, 이로 인한 시간지연 때문에 호스트의 상태가 변경되어 올바르지 못한 결정을 내릴 수 가 있다. 또한, 입찰 방식을 통한 부하 공유는 항상 부하 공유를 위해 브로드케스트를 해야 하는 네트워크 비용에 대한 부담이 따른다.

Lin은 그래디언트(gradient)모델에 기반을 둔 부하공유 기법을 제안 하였는데 각 호스트는 자기 자신과 바로 연결된 호스트들과 정보를 교환한다[6].

여기서 호스트는 저부하(light), 보통(moderate), 과부하(heavy)의 세 가지로 나뉘어 진다. 각 호스트들은 저부하 상태 호스트로부터의 최소 거리 값을 가지고, 저부하 호스트는 0값을 가지고 인접 호스트가 저부하 상태가 아니면 인접 호스트의 거리 값은 1을 더한다. 그리고 자신의 값을 다른 인접 호스트들에게 전파하고 이 전파를 받은 호스트는 자신이 저부하 상태가 아니면 그 값에 1을 더하고 인접 호스트에게 전파한다. 저부하 상태면 자신의 값을 0으로 하고 모든 인접 호스트에게 전파한다. 이런 전파과정을 하면 자신의 거리 값은 저부하 상태의 호스트로부터의 최소 거리 값이 된다.

이 기법은 인접 호스트들의 거리 값과 부하를 자신과 비교해야 하므로 부하가 변경될 때 인접 호스트들의 상태를 메시지로 가져와야 하고 계산 결과를 인접 호스트들에게 전파시켜야 한다. 이러한 행동이 모든 호스트들에서 발생하게 되므로 메시지의 양이 굉장히 늘어나게 된다. 그리고 이주하던 프로세스가 전파되는 도중 여러 호스트들이 상태변경을 심하게 발생시키면 네트워크를 떠돌 가능성이 있다. 또한, 그리드 데이터베이스에서는 여러 노드에 복제본이 저장되고, 복제본을 중심으로 작업 부하가 발생하기 때문에 Lin이 제안한 인접 노드를 통한 부하 공유가 불가능하다.

또 다른 부하 공유를 위한 연구로, 클러스터 시스템에서 부하분산을 담당하는 하나 이상의 조정자(coordinator or director)를 두고, 이를 통해 Round-Robin(RR) 알고리즘과 Weighted Round-Robin (WRR) 알고리즘, 그리고 Least-Connection(LC) 알고리즘을 적용하는 방식이 있다. 사용자 질의는 조정자로 먼저 전달되며, 조정자는 정해진 부하 공유 알고리즘을 통해 알맞은 서버를 찾게 된다.

Round-Robin 알고리즘은 서버의 접속 수나 응답

시간을 고려하지 않고 모두 동일하다는 가정 하에 순서적으로 작업 서버를 선택하는 방식으로서 스케줄링에서 소요되는 처리 시간을 효과적으로 단축시킬 수 있다[9].

Weighted Round-Robin(WRR) 알고리즘은 각 서버에 처리용량에 비례한 가중치를 부여하여 부하를 분배하는 방식이다. 따라서 가중치에 따라서 작업 할당 순서가 RR 방식으로 서버들로 보내진다. 하지만 부하가 폭주하는 경우, 부하의 크기를 고려하지 않고 각 요청을 동일하게 간주하므로 단 시간에 폭주하는 시스템의 경우에 부적합하다고 할 수 있다[9].

Least-Connection(LC) 알고리즘은 새로운 서비스 요청이 발생하는 경우, 웹 클러스터 시스템을 구성하는 각각의 서버 중 현재 가장 작은 접속량을 가진 서버를 선택하여 작업을 할당하는 방식이다[9].

3. 링 기반 연결 구조를 이용한 부하 분산 기법

본 장에서는 그리드 데이터베이스에서 복제 데이터들을 하나로 연결하는 링 구조와 각 노드의 부하에 따라 사용자 질의를 다른 노드로 전달하는 부하 분산 과정을 설명한다.

3.1 링 기반 연결 구조

링 기반 연결 구조는 모든 복제본을 하나의 링 형태로 연결한다. 링 구조에 사용되는 각각의 링크는 정방향과 역방향의 방향성을 갖는다. 정방향 링크는 노드의 작업 부하가 한계 값을 벗어난 상태에서 다른 노드로 질의를 전달할 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 1]과 같다.

[그림 1]에서 데이터 A, B, C는 각각의 노드에 복제본의 형태로 분포되어 있다. 연결 구조는 노드와 상관없이 복제본을 기준으로 구성되어 있다. 예를 들어, 데이터 B는 B_1->B_2->B_3의 순서로 연결이 되어 있으며, B_2의 작업 부하가 한계치를 넘은 경우 노드 N2로 입력되는 질의 중에서 B와 관련된 질의는 정방향 링크로 연결된 B_3으로 전달된다. 마찬가지로 작업 부하가 한계치를 넘은 노드 N3에서 A_3에 대한 질의가 입력된 경우 복제본 A를 포함하는 노드 N5로 질의가 전달된다. 역방향 링크는 노드의 과부하 상태 또는 보통상태를 전달할 때 사용된다.

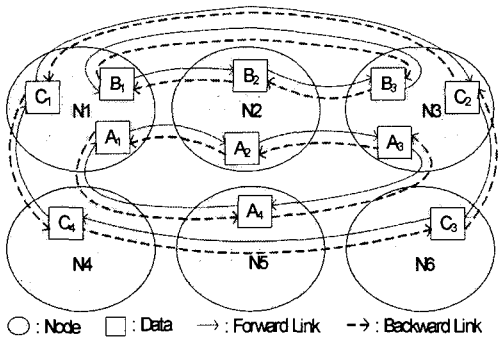


그림 1. 링 기반 연결 구조를 사용한 데이터베이스 구조

[그림 2]는 각 노드의 복제본들의 연결 상태를 저장하는 데이터 구조이다. 각각에 대한 설명은 다음과 같다.

- 복제본 이름(RN: Replica Name): 복제된 데이터의 이름.
- 정방향 링크(FL: Forward Link): 노드의 과부하 상태에서 질의를 다른 노드로 전달할 때 사용되는 링크.
- 역방향 링크(BL: Backward Link): 임의 노드가 과부하 상태가 되어 더 이상의 질의 처리가 불가능해지면, 자신의 상태를 이전 노드로 메시지를 전달하기 위해 사용되는 링크.

[그림 3]은 N3에 저장된 연결 목록이다. N3은 A_3, B_3, C_2를 가지고 있으며, 정방향 링크와 역방향 링크는 다음과 같다.

정방향 링크와 역방향 링크는 평상시에는 [그림 1]에서 설명한대로 사용된다. 그러나 임의 노드에 사용자 질의가 집중되어 더 이상 질의 처리가 불가능한 경우에는 정방향 링크의 구조가 달라진다. 현재의 구

Replica Name (RN)	Forward Link (FL)	Backward Link (BL)

그림 2. 링 구조 관리를 위한 데이터 구조

Replica Name (RN)	Forward Link (FL)	Backward Link (BL)
A	N5	N2
B	N1	N2
C	N6	N1
.....	.....	.....

그림 3. 노드 N3의 연결 목록

조에서 해당 노드를 이전 노드의 정방향 링크에서 제거해야 한다. 예를 들어, [그림 4]에서 노드 N2가 과부하로 더 이상의 질의 처리가 불가능할 경우 연결 구조가 바뀐다. 원래 B_2를 가리키고 있던 정방향 링크가 B_3으로 바뀌게 된다. 따라서 노드 N2가 보통 상태로 바뀔 때까지 N2는 더 이상의 질의 처리를 하지 않아도 되며, B_3을 포함하는 노드 N3으로 질의를 계속해서 보내게 된다. 또한, 노드 N1에서도 과부하로 사용자 질의를 다른 노드로 보내야 할 때 바로 N3으로 보낼 수 있다.

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 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 이전노드로 메시지를 전송한다.

3.2 링 연결 구조 기반의 부하 분산 기법

링 연결 구조로 연결된 환경에서 부하 분산을 위해 필요한 요소는 다음과 같다. 각각의 요소들은 부하 분산 기법 적용 시에 필요한 인자들이다.

- 최대 작업 부하 값(max_workload): 이는 한 노드가 수용할 수 있는 최대 작업 부하 값을 나타낸다. 보통 CPU, 메모리, 디스크 등의 사용량과 사용자 접속 수, 동시 트랜잭션 수 등이 포함된다.
- 현재 작업 부하 값(current_workload): 현재 노드의 작업 부하 값을 나타낸다.
- 현재 노드 번호(current_node): 현재 노드의 번호를 나타낸다. 모든 노드가 자신의 고유한 번호(또는 이름)를 가지게 된다.
- 현재 노드의 상태(current_state): 현재 노드의 상태를 지정한다. 평상시에는 보통(normal) 상태이며, 작업부하가 한계 값을 넘게 되면 과부하(overloaded) 상태가 된다.

부하 분산 과정은 사용자 질의가 입력될 때마다 실행되거나, 주어진 시간 간격마다 실행될 수 있다. 보통 상태에서 과부하 상태로 바뀌는 경우 그 처리 알고리즘은 다음과 같다.

[알고리즘 1]에서 노드가 과부하 상태로 바뀌게 되면, 먼저 자신의 정방향 링크를 재조정한다. 정방향 링크가 자신을 가리키는 경우는 자신을 제외한 모든 노드가 과부하 상태였음을 의미한다. 현재 노드까지 과부하 상태로 바뀌면 모든 노드가 과부하 상태인 것이므로, 정방향 링크를 NULL값으로 바꾼다. 그렇지 않은 경우에는 기존의 FL값이 계속 사용되며, 이외의 경우에는 정방향 링크가 자신을 가리키지 않는다면 해당 질의를 정방향 링크를 통해 전송한다(03~09라인).

이미 정방향 링크가 NULL값인 경우에는 해당 질의를 다른 노드로 전송하지 않고 현재 노드에서 처리한다(10~12라인). 이후 현재 노드는 자신의 상태를 과부하 상태(overloaded)로 변경하고, 자신의 정보와 새롭게 바뀐 정방향 링크 정보를 역방향 링크를 통해 전달한다(13~14라인).

[그림 5]는 알고리즘 1을 설명하기 위한 예제이다. (a)에서 노드 N2는 단 하나의 보통 상태인 노드이다. 만일 N2의 부하값이 max_workload를 초과하게 되면 (b)와 같은 상태가 된다. 변환 과정에서 [알고리즘 1]의 04~06라인에 의해 정방향 링크를 NULL값으로 바꾸고, 13~14라인에 의해 자신의 상태를 과부하

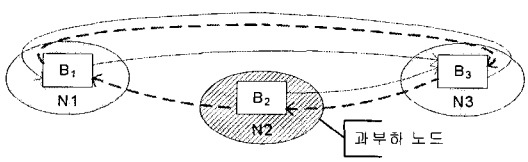


그림 4. 노드의 과부하 상태에서 연결 구조의 변화

```
Input
max_workload: 최대 작업 부하 값
current_workload: 현재 작업 부하 값
current_node: 현재 노드 번호
current_state: 현재 노드의 상태 (normal or overloaded)
FL: 정방향 링크
BL: 역방향 링크
Output
new_FL: 새로운 정방향 링크 값
Begin
01 : if current_workload is higher than max_workload
     and current_state is normal
02 :   if FL is not null value
03 :     if FL is current_node
04 :       new_FL <- null value
05 :       FL <- new_FL
06 :       process user's request
07 :     else
08 :       send user's request to FL
09 :     endif
10 :   else
11 :     process user's request
12 :   endif
13 :   set current_state to overloaded
14 :  send message to BL with current node infor-
      mation and new_FL
15 : endif
End
```

알고리즘 1. 과부하 상태로 전환 시 처리과정

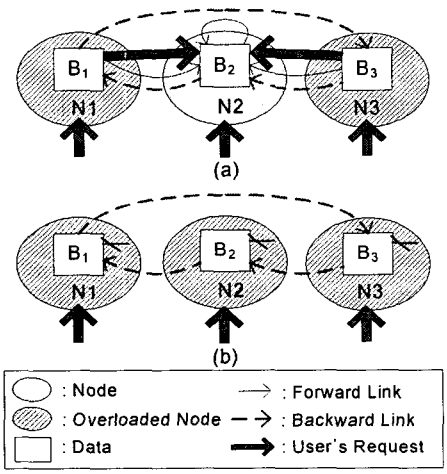


그림 5. 노드의 과부하 상태 전환 시 링크의 변화: (a) 보통 상태의 노드가 하나인 경우 (b) 모든 노드가 과부하 상태인 경우

상태로 바꾼 후 역방향 링크를 통해 해당 메시지를 N1으로 전달한다. 전달된 메시지의 처리 과정은 [알고리즘 3]에서 설명한다. [그림 5]의 (b)에서 모든 노드는 과부하 상태이고, 사용자 질의는 처음 접속한 노드에서 처리된다.

[알고리즘 2]는 노드가 과부하 상태에서 보통 상태로 변경될 때 처리되는 과정이다.

[알고리즘 2]에서 노드가 보통 상태로 바뀌게 되면, 먼저 입력된 사용자 질의를 현재 노드에서 처리한다. 그리고 자신의 노드의 상태를 보통 상태(normal)로 바꾼다(02~03라인). 만일 정방향 링크가 NULL 값인 경우, 이는 현재까지 모든 노드가 과부하 상태였음을 의미하며, 현재 노드 하나만 보통 상태이므로, 정방향 링크를 자신으로 변경해야 한다. 이외의 경우에는 기존의 FL값이 사용된다. 따라서 자신의 노드로 정방향 링크의 값을 변경하고, 역방향 링크를 통해 바뀐 정방향 링크를 메시지로 보낸다(04~08라인).

[그림 6]은 과부하 상태에서 보통 상태로 변화하는 과정을 나타낸 예제이다. N3이 보통 상태로 바뀌는 경우 [알고리즘 2]의 02~03라인에 의해 보통 상태로 전환된다. 또한, 08라인에서 N2로 메시지를 전달하고, N2는 자신의 정방향 링크를 N3로 변경하게 된다([알고리즘 3]에서 설명됨). 04~07라인은 [그림 5]의 (b)인 상태에서 N2가 보통 상태로 전환되는 경우 이용된다. 이를 통해 N2의 정방향 링크는 자신을 가리키게 되며, [그림 5]의 (a)와 같은 상태가 된다.

```
Input
max_workload: 최대 작업 부하 값
current_workload: 현재 작업 부하 값
current_node: 현재 노드 번호
current_state: 현재 노드의 상태 (normal or overloaded)
FL: 정방향 링크
BL: 역방향 링크
Output
new_FL: 새로운 정방향 링크 값
Begin
01 : if current_workload is lower than max_workload
     and current_state is overloaded
02 :   process user's request
03 :   set current_state to normal
04 :   if FL is null value
05 :     new_FL <- current_node
06 :     FL <- new_FL
07 :   endif
08 :   send message to BL with current node
       information and new_FL
09 : endif
End
```

알고리즘 2. 보통 상태로 전환 시 처리과정

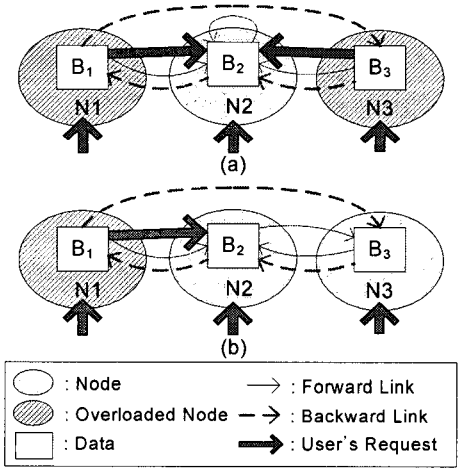


그림 6. 노드의 보통 상태 전환 시 링크의 변화: (a) 보통 상태의 노드가 하나인 경우 (b) N3이 보통 상태로 변한 경우

```
Input
message: 다음 노드로부터 전달된 메시지
current_state: 현재 노드의 상태 (normal or overloaded)
FL: 정방향 링크
BL: 역방향 링크
Output
new_FL: 새로운 정방향 링크 값
Begin
01 :  if a message is received from next node
02 :     new_FL <- new_FL of message
03 :     FL <- new_FL
04 :     discount TTL of message
05 :     if current_state is overloaded and TTL is
         bigger than 0
06 :        send message to BL
07 :     else
08 :        delete message
09 :     endif
10 :  endif
End
```

알고리즘 3. 노드 간 메시지 처리과정

[알고리즘 3]은 다음 노드로부터 전송된 메시지를 처리하는 과정을 나타낸다. 해당 메시지를 전달 받은 노드는 자신의 정방향 링크 값을 메시지 안에 포함된 정방향 링크로 변경한다(02라인). 그리고 메시지의 무한 루프를 방지하기 위해 메시지 내의 TTL 값에서 1을 뺀다(04라인). TTL(Time To Live)은 처음 메시지를 생성하는 노드에서 생성된다. 처음 TTL 값은 모든 복제본의 수로 정해진다. 만일 서버의 상태가 과부하 상태이고 TTL이 0보다 크다면, 해당 메시지는 다시 이전 노드로 전송된다(05~06라인). 그렇지 않다면, 해당 메시지는 현재 노드에서 삭제된다(08라인).

[그림 7]는 [알고리즘 3]의 처리 과정을 설명하는 예제이다. (a)는 [그림 5]의 (b)와 같다. 모든 노드가 과부하 상태이고 정방향 링크는 현재 NULL값이다. 이때 N2가 보통 상태로 전환되면 [알고리즘 2]에 의해 N2는 (b)의 ①에서 N2의 정방향 링크가 자신을 가리키게 된다. [알고리즘 2]의 08라인에서 생성된 메시지는 N1로 전달된다. N1에서는 ②와 같이 [알고리즘 3]의 02~04라인을 통해 자신의 정방향 링크를 N2를 가리킨다. 그리고 N1은 과부하 상태이므로 05~07라인에서 N3로 메시지를 재전송 한다. 마지막으로 ③은 N3에서 처리되는 과정으로서 N2와 동일하게 처리된 후 TTL에 의해 메시지가 종료된다.

이렇게 메시지를 전송하는 이유는 다음과 같다. 만일 N3를 보통 상태로 바꾸면서 N1으로 보낸 메시

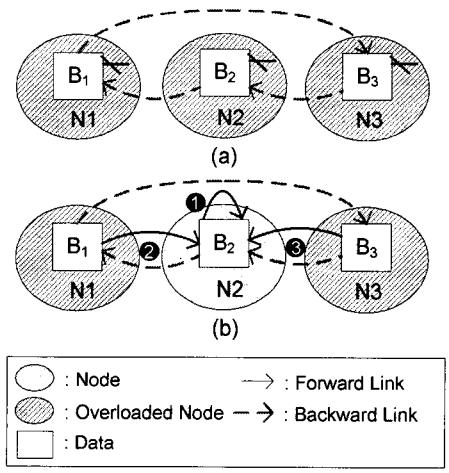


그림 7. 상태 전환 시 노드 간 메시지 전달 과정: (a) 초기 상태, (b) 메시지 전달에 의한 정방향 링크의 변화

지를 N3가 받지 못한다면, N3의 정방향 링크는 NULL값에서 바뀌지 않을 것이다. 따라서 본 논문에서는 정방향 링크의 변경을 위해 메시지를 받은 노드가 과부하 상태인 경우 다음 노드에도 동일한 메시지를 전달하여 노드의 상태 변경 정보를 확산시킨다.

이외에 평상시 상태에서 입력된 사용자 질의는 해당 노드에서 바로 처리하며, 과부하 상태에서는 정방향 링크를 통해 질의를 전송한다. 만일 정방향 링크가 NULL 값이거나 자신의 노드를 가리키는 경우에

는 해당 노드에서 처리한다.

4. 성능 평가

본 장에서는 제안 기법의 평가를 위한 실험 환경에 대해 설명하고, 클라이언트 수, 질의집중도 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 CSIM을 사용하였다. CSIM은 분산 환경에서의 컴퓨팅 모델, 알고리즘 평가 등 대부분의 시뮬레이션이 가능한 툴이다. 이미 여러 논문에서 CSIM을 통한 성능평가가 작성된 바 있다[8].

제안 기법의 구현을 위해 다수의 클라이언트와 서버를 구성하였으며, 모든 서버에 제안한 자료구조를 저장하였다. 제안 기법의 비교 평가 대상이 되는 기존기법의 구성을 위해 코디네이터와 서버의 작업 부하 값을 계산하여 전송하는 기능들을 추가 구현하였다. 전체 구현에 소요된 라인 수는 약 1700라인이며, MS Visual C++ 6.0을 사용하였다. 전체적인 구성은 가능한 실제 환경에 맞게끔 구현하였으며, 평가 환경은 다음과 같이 구성하였다.

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

표 1. 실험 환경

실험 요소	데이터 범위
실험 시간	3000 unit time
서버 수	3~20 개
클라이언트 수	1~40 개
복제본 수	2~20 개
질의 처리 시간	4~7 unit time
질의 전송 시간	2~4 unit time
질의 입력 간격	0.1 unit time
코디네이터 처리시간	0.5~1.5 unit time
작업 부하 최대값	1~20개(작업 대기열의 길이)

실험을 위한 비교 평가 대상은 첫 번째로 부하 분산을 수행하지 않고 입력되는 질의를 해당 노드에서 바로 처리하는 방식을 사용하였으며, 이후 명칭을 SIMPLE로 사용한다.

두 번째 방식은 조정자를 두어 모든 노드의 현재 부하 정보를 취합하고 가장 부하가 적은 노드로 질의를 전송하는 방식으로, 이후 명칭을 COORD로 사용한다. COORD 방식에서 사용자 질의는 먼저 중앙의 코디네이터로 전송된다. 코디네이터는 모든 복제본들에 대한 카탈로그 정보를 가지고 있다. 해당 카탈로그 정보와 취합된 각 노드들의 작업 부하 정보를 바탕으로 최적의 노드를 선택하여 해당 노드로 질의를 전송한다.

그리고 제안 기법은 본 논문에서 제안하는 구조를 그대로 구현하여 사용하며, 명칭을 RING으로 한다.

본 논문에서는 위와 같이 부하 분산을 수행하지 않는 SIMPLE과 코디네이터를 통해 부하 분산을 수행하는 COORD기법을 통해 제안기법의 비교평가를 수행한다. 이는 기존의 부하 분산 기법들의 적용 환경, 부하 분산 방식이 노드 단위로 적용되는데 반해 제안 기법은 복제된 데이터를 기준으로 부하 분산 과정을 수행하기 때문에 평가를 위한 기준을 세우기가 어렵다. 따라서 COORD 방식은 코디네이터가 모든 데이터의 메타 정보를 포함하고 있으며, 본 논문에서 제안하는 복제본 기반의 부하 분산 방식을 적용하였다.

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 실험평가

본 실험 평가에서는 클라이언트의 수, 서버의 수, 질의 집중 비율을 주요 평가 인자로 사용하였다. 이외에도 복제본의 수, 질의 처리 시간, 질의 전송 시간 등의 몇몇 값을 바꾸어 가며 평가하였지만, 실험 결과의 변위 값의 증감 외에는 별 다른 변동사항이 없어 본 논문에서는 설명하지 않는다. 모든 실험 결과는 처리량(Throughput)과 처리 시간(Response Time)

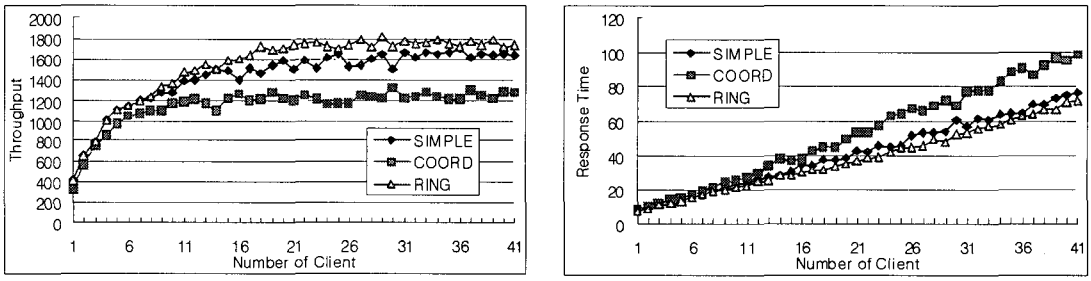
으로 나타낸다.

[그림 5]는 클라이언트 수의 변화에 따른 처리 성능의 변화를 나타낸다. 질의는 전체 노드로 고르게 분포되어 전송된다.

[그림 8]의 a)에서COORD 방식은 클라이언트 수를 11개 이상으로 늘렸을 때 더 이상성능 향상이 되지 않았다. 이는 코디네이터의 처리 한계로 인한 것으로 판단된다. SIMPLE 방식과 RING 방식은 거의 같은 성능을 나타내었으나, RING 방식이 약간의 성능 향상을 나타내고 있다. 이는 RING 방식에서 수행되는 부하 분산으로 일어난 현상이다. [그림 8]의 b)에서는 COORD 방식이 다른 기법과 클라이언트 수에 따른 성능의 차이는 있지만, 코디네이터의 성능을 향상시킬수록 다른 기법과 처리 성능이 비슷해진다.

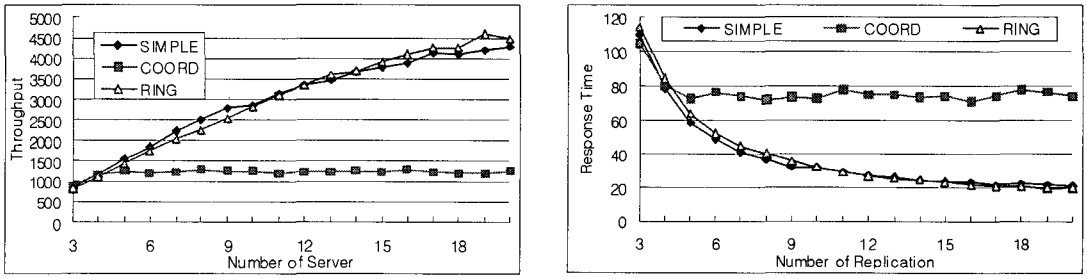
[그림 9]은 서버 수의 변화에 따른 처리 성능의 변화를 나타낸다. 이번 실험을 통해 COORD 방식은 서버의 수와는 상관없이 코디네이터의 처리 성능에 전체 처리량이 결정되는 것을 확인할 수 있다.

[그림 10]은 전체 복제본에 대한 질의 집중 비율을 변화시켰을 때의 성능을 나타낸다. 복제본의 수와는 상관없이 실험 결과는 [그림 10]과 비슷하게 나왔다. SIMPLE 방식은 질의 집중 비율이 증가할수록 성능이 떨어졌으며, COORD 방식은 질의 집중 비율에 상관없이 항상 동일한 성능을 나타내었다. 본 논문에



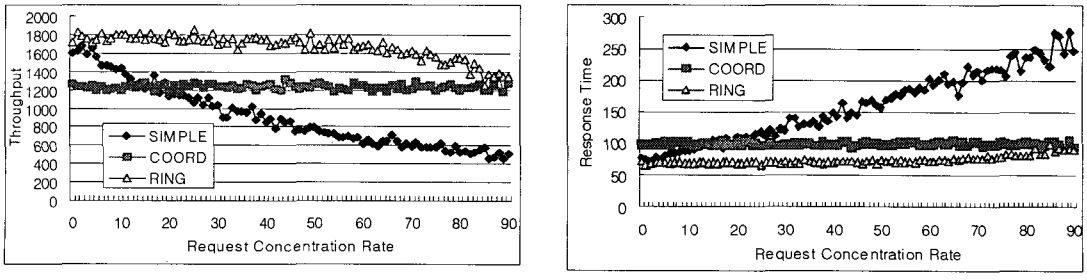
(a) (b)

그림 8. 클라이언트 수 변화에 따른 처리 성능의 변화: (a) 단위 시간 당 처리량, (b) 평균 질의 처리 시간



(a) (b)

그림 9. 서버 수 변화에 따른 처리 성능의 변화: (a) 단위 시간 당 처리량, (b) 평균 질의 처리 시간



(a) (b)

그림 10. 질의 집중 비율에 따른 처리 성능의 변화: (a) 단위 시간 당 처리량, (b) 평균 질의 처리 시간

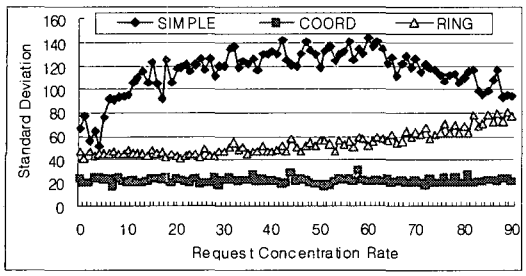


그림 11. 질의 집중 비율에 따른 노드별 분배 정도

서 제안하는 RING 방식은 질의 집중 비율이 높아질수록 처리 성능이 안 좋아졌으나, COORD와 SIMPLE 방식 보다는 좋은 성능을 나타내었다.

[그림 11]은 질의 집중 비율에 따른 표준편차의 변화를 통해 각 방식의질의 분배 정도를 평가하였다. 가장 좋은 분배 비율을 나타낸 것은 COORD 방식이며, SIMPLE 방식의 경우 질의 집중 비율에 따라 큰 폭의 변화를 나타낸다. RING 방식은 질의 집중 비율이 높아질수록 노드의 분배 정도가 안 좋아지고 있다. 이는 작업 부하가 한계 값을 벗어나지 않는 한 부하 분산 연산을 수행하지 않아 나타나는 현상이다.

5. 결론 및 향후 연구

기존의 부하 분산 기법은 노드의 수가 많고 사용자 질의가 유동적으로 변하는 그리드 데이터베이스에는 적용하기 힘들다. 따라서 이러한 환경에 구애받지 않고 최적의 성능을 유지할 수 있는 부하 분신 기법이 요구된다.

본 논문에서는 링 기반 연결 구조를 이용한 부하 분산 기법을 제안하고, 이에 대한 다른 기법과의 성능 평가를 수행하였다. 제안 기법은 그리드 데이터베이스를 구성하는 모든 노드의 복제본들을 중심으로 링 구조를 구성하였으며, 이러한 링 구조를 통한 작업 부하의 분산 과정을 포함한다. 성능 평가에서는 모든 노드의 작업 부하를 취합해 이를 바탕으로 부하 분산을 유도하는 기존 기법에 비해 좋은 성능을 보였다.

제안 기법은 노드의 추가와 삭제가 자유롭고 다른 노드와의 연결 구조가 동적이며, 노드의 수가 방대한 환경에서 유용하게 사용될 수 있다. 복제본 간의 연결 구조를 통해 데이터 중심의 부하 분산 방식을 실현하였으며, 어떠한 어플리케이션에도 적용 가능하다.

향후 연구로는 각종 응용 환경에서 네트워크 구성, 부하 분산 방식 등이 서로 다른 기법 간의 성능 평가가 필요하다.

참 고 문 헌

[1] Anastasiadi, S. Kapidakis, C. Nikolaou, and J. Sairamesh, "A computational economy for dynamic load balancing and data replication," *in Proc. 1st International Conference on Information and Computation Economies*, pp. 166~180, Charleston, SC, 1998.

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

[3] FARBER D.J. & LARSON K.C., "The system architecture of the distributed computer system - the communications system," *Symp. Computer-Communications Networks and Teletraffic* (New York), pp. 21-27., 1972.

[4] J. Gray, P. Helland. P. O'Neil, and D. Shasha, "The dangers of replication and a solution," *ACM SIGMOD International Conference on Management of Data*. Published as SIGMOD Record, 25(2):173~182, ACM, 1996.

[5] Zhiling Lan, Valerie E. Taylor, and Greg Bryan, "Dynamic load balancing of SAMR applications on distributed systems," *Scientific Programming*, 10(4): 319-328, 2002.

[6] F. C. H. Lin and R. M. Keller, "The Gradient Model Load Balancing Method," *IEEE Trans. on Software Engineering*, vol. SE-13, no. 1, pp. 32-38, 1987.

[7] 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.

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

[9] "Job Scheduling Algorithms in Linux Virtual Server," http://www.linuxvirtualserver.org/docs/scheduling.html, Dec. 2003.

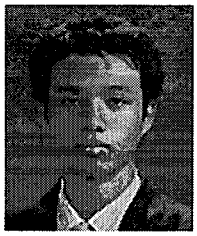
[10] 강철, "On-Demand 환경의 데이터베이스 Grid와 고 가용성," *데이터베이스연구회*, VOL. 00 NO. 00 pp. 209~224, 2005.

[11] 김구수, 엄영익, "분산 시스템에서의 부하 공유 기법 설계 및 성능 평가," *한국정보처리학회 논문지*, 제4권, 제8호, pp. 2092-2105, 1997.

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



신 숭 선

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

박 순 영

1989년 인하대학교 전자계산학과 졸업(이학사)
1991년 인하대학교 대학원 전자계산공학과 졸업(공학석사)
2006년 인하대학교 대학원 컴퓨터정보공학과 졸업(공학박사)
1991년 2월~현재 한국전자통신연구원 데이터베이스 연구팀 선임연구원
관심분야 : 저장 시스템, 데이터베이스 시스템, 센서 데이터베이스

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