확장 가능한 고가용 데이터베이스 클러스터에서 B+트리 색인의 온-라인 재조직 기법

이 충 호† · 배 해 영††

요 약

온-라인 재조직 기법은 인터넷 환경과 같은 동적 환경에서 높은 가용성과 고성능을 제공하기 위한 비공유 데이터베이스 클러스터의 필수적인 기능이다. 기존의 온-라인 재조직 기법은 클러스터 안의 프로세싱 노드에 과부하가 생긴 경우, 과부하 노드의 데이터를 인접 노드로 빠르게 이동시킴으로써 부하 분배를 수행한다. 그러나 동시에 두개 이상의 다중 노드에 과부하가 발생된 경우, 부하 분배를 위해 인접 노드로 여러 번의 반복된 데이터 이동이 발생되고, 재조직 수행동안 시스템의 응답 속도가 늦어지는 문제점이 있다. 본 논문에서는 다중 노드에 발생한 과부하 문제를 빠르고 효율적으로 해결하는 향상된 B+트리 색인의 온-라인 재조직 기법을 제안한다. 제안된 기법은 확장 가능한 데이터베이스 클러스터 환경 하에 온-라인 확장을 통해 새롭게 추가된 노드들에 데이터를 이동시킴으로써 데이터 이동의 회수를 줄이면서 빠른 시간 안에 온-라인 재조직을 수행하도록 한다. 또한 제안된 기법에서는 B+-트리 색인 대신 캐시를 고려한 CSB+-트리 색인을 이용하여 검색과 갱신 연산을 보다 빠르게 처리하도록 한다. 제안된 온-라인 재조직 기법은 확장 가능한 고가용 데이터베이스 클러스터 시스템으로 개발된 최대 결함허용 보장 데이터베이스 클러스터(Ultra Fault-Tolerant Database Cluster) 환경에서 성능 평가를 통해 기존 기법에 비해 빠르고 효율적임을 보인다.

Online Reorganization of B+ tree in a Scalable and Highly Available Database Cluster

Chung Ho Lee† · Hea Young Bae††

ABSTRACT

On-line reorganization in a shared nothing database cluster is crucial to the performance of the database system in a dynamic environment like WWW where the number of users grows rapidly and changing access patterns may exhibit high skew. In the existing method of on-line reorganization have a drawback that needs excessive data migrations in case more than two nodes within a cluster have overload at the same time.In this paper, we propose an advanced B+ tree based on-line reorganization method that solves data skew on multi-nodes. Our method facilitates fast and efficient data migration by including spare nodes that are added to cluster through on-line scaling. Also we apply CSB+ tree (Cache Sensitive B+ tree) to our method instead of B+ tree for fast select and update queries. We conducted performance study and implemented the method on Ultra Fault-Tolerant Database Cluster developed for high scalability and availability. Empirical results demonstrate that our proposed method is indeed effective and fast than the existing method.

키워드 : 온-라인 재조직(On-line reorganization), 온-라인 확장(On-line scaling), 색인 재구성(Index reorganization), 데이터베이스 클러스터 시스템(Database Cluster System), 고 가용성(High availability)

1. 서 론

오늘날 인터넷 상에서 관리되는 데이터의 폭발적인 증가와 전자상거래와 같은 웹 기반 응용 프로그램 보급의 증가로 대량의 데이터 관리와 클라이언트 질의에 대한 빠른 응답시간이 중요한 문제로 대두되고 있다. 특히, 웹은 사용자 수가 급증하거나 접근 패턴의 변경에 따른 작업부하의 편중이 심하게 발생될 수 있는 동적인 환경이다. 예를 들어, 주식 거래 사이트와 같이 데이터 집중적인 응용 프로그램의 경우 예상치 못한 작업부하가 발생할 수 있는데, 전체 주식 거래량의 급격한 증가나 특정 주식에 대한 부하의 집중이 그러한 경우라고 할 수 있다[5]. 이러한 대량의 데이터 집합과 하부 데이터베이스에 대한 읽기/쓰기 접근 패턴을 갖는 인터넷 서비스를 안정되게 제공하기 위한 비용대비 효율적인 하드웨어 플랫폼으로 여러 대의 컴퓨터를 클러스터로 구성하는 방법이 폭 넓게 사용되고 있다. 즉, 사용자의 연결요청을 받아 들이는 웹 서버 여러 대를 하나의 웹 서버 클러스터로 구성하거나 데이터에

※ 본 연구는 정보통신부 정보통신 우수시범학교 지원사업(1999)의 연구 결과임.
† 정 회 원 : 인하대학교 대학원 전자계산공학과
†† 종신회원 : 인하대학교 전자계산공학과 교수
논문접수 : 2002년 4월 18일, 심사완료 : 2002년 7월 22일

대한 처리를 담당하는 데이터 저장 노드를 데이터베이스 클러스터로 구성하는 구조이다. 클러스터 기반의 인터넷 구조는 부분적인 오류에 상관없이 지속적으로 서비스를 제공함으로써 가용성을 극대화할 수 있고 고가의 슈퍼 컴퓨터가 아닌 워크스테이션 여러 대를 클러스터로 구성하여 확장성을 제공할 수 있다[2, 3, 12].

비공유(Shared-Nothing) 구조의 데이터베이스 클러스터 시스템에서 클라이언트 질의 요청에 대한 빠른 응답 속도와 가용성을 높이기 위해 데이터의 분할(Fragmentation) 또는 복제(Replication) 정책을 사용한다. 그러나, 질의 패턴의 변화에 의해 초기의 데이터 분배 정책이 적절치 못해 전체 시스템의 성능을 저하 시킬 수 있으므로 데이터에 대한 온-라인 재조직 기능이 필수적이다[3, 8]. 온-라인 재조직 기법과 관련된 기존의 연구로는 중앙집중 방식의 데이터베이스 시스템과 병렬 데이터베이스 시스템에서 많은 연구가 수행되어 왔다. [14]는 데이터가 희박한 B^+ 트리 색인을 온-라인 재조직하는 기법에 대해서 연구하였고, [15]은 온-라인 재조직 중에 이차 색인의 갱신을 늦추는 기법을 제안하였다. [1]은 비공유 시스템에서 부하의 불균형이 발생되었을 때 이동되어야 할 데이터의 크기에 대해서 다루었다. 최근의 연구에 의하면, [5]는 비공유 시스템에서 B^+색인에 기반 한 데이터의 이동을 빠르고 효율적으로 수행하는 온-라인 재조직 기법을 제안하였다. 기존의 온-라인 재조직 기법은 고정된 개수의 프로세싱 노드(Processing Node)[1]들 중에서 하나의 노드에 과부하가 생긴 경우, 과부하 노드의 데이터를 인접 노드로 빠르게 이동시킴으로써 과부하 문제를 해결한다. 그러나 클러스터 안에 두개 이상의 다중 노드에 과부하가 집중된 경우, 부하 분배를 위해 많은 횟수의 데이터 이동이 필요하고, 전체 시스템의 질의에 대한 평균 응답 시간을 느리게 만드는 한계점을 갖는다.

본 논문에서는 확장 가능한 데이터베이스 클러스터에서 데이터 재조직을 위한 빠르고 효율적인 재조직 기법을 제안한다. 제안된 기법은 기존의 기법과 비교해서 다음과 같은 차이점을 갖는다.

1) 클러스터를 구성하는 활성화 상태의 노드에서만 데이터 재조직을 수행하는 것이 아니고 온라인 확장을 통해 새롭게 추가된 노드를 포함하여 데이터 재조직을 수행한다. 즉, 특정 노드에 집중되는 작업량을 근접 노드나 온라인 확장을 통해 추가된 노드에 분배하는 방식으로, 하나의 노드 부하 뿐만 아니라 기존 문제점으로 제기된 두 개 이상의 노드에서 과부하가 발생한 경우, 기존 기법에 비해 작은 양의 데이터 이동만으로 빠르고 효율적인 온-라인 재조직을 수행하도록 한다.

2) 기존의 기법에서는 병렬 데이터베이스 시스템 환경에서의 B^+-트리 색인의 재조직을 다루었지만 제안된 기법에서는 클러스터를 구성하는 각 노드의 데이터베이스가 빠른 데이터 접근과 동시성 보장을 위해 메모리 상주 데이터베이스 이므로 캐시를 고려한 CSB^+-트리(Cache Sensitive B^+-Tree) 색인의 재조직을 수행한다. 소스(Source) 노드에서 목적(Target)노드로 데이터를 이동시킬 때마다 캐시의 특성을 반영한 색인을 유지함으로써 실시간 검색 질의 뿐만 아니라 갱신 질의에 대해서 빠른 응답 시간을 보이도록 한다.

본 논문에서는 확장 가능한 고 가용성 데이터베이스 클러스터 시스템으로 개발된 *Ultra Fault-Tolerant Database Cluster* 라는 시스템을 실험 환경으로 해서 제안된 기법이 기존 기법에 비해 빠르고 효율적인 온-라인 재조직 기법임을 성능평가를 통해 보인다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구를 다루고, 3장에서는 제안된 기법의 시스템 환경과 배경에 대해 언급한다. 4장에서는 제안된 온-라인 재조직 기법에 대해 구체적으로 설명하고 5장에서 성능 평가를 보이고 6장에서 결론을 맺는다.

2. 관련 연구

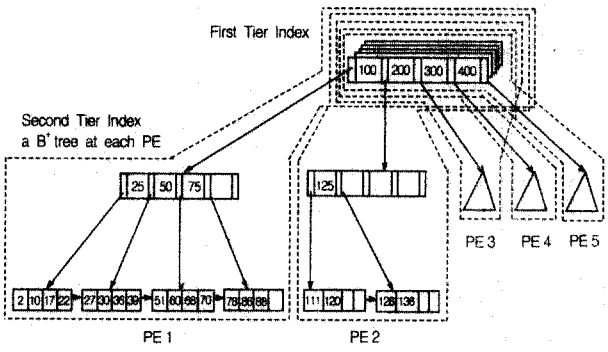
본 장에서는 관련연구로 기존의 대표적인 온-라인 재조직 기법과 본 논문의 4장에서 제안하는 재조직 기법에서 사용되는 메모리 상주 데이터베이스에서의 B^+트리 색인에 대해서 설명한다.

2.1 병렬 데이터베이스에서 B^+-트리 색인에 의한 온라인 재조직 기법

비공유 병렬 데이터베이스에서 데이터가 초기에 모든 프로세싱 노드에 레코드의 특정 속성값의 범위 값에 의해 분할되어 있는 상태에서 클라이언트 질의의 패턴이 특정 노드의 데이터만을 검색하는 경우나 특정 노드에 데이터 삽입이 집중되는 경우, 해당 노드에 과부하 또는 불균형 상태(Data Skew)가 발생된다. 이러한 불균형 상태를 제거하여 전체 시스템의 생산량(Throughput)을 늘리고 시스템의 응답속도를 빠르게 하기 위하여 각 노드간 색인에 기초한 데이터의 이동을 수행한다. 색인의 구성은 (그림 1)과 같이 2 계층 트리의 데이터 구조를 구성하는데 첫번째 트리 계층은 데이터가 저장되어 있는 프로세싱 노드를 탐색하는 데 사용되고 이 부분은 모든 노드에 복사되어 메모리에서 관리된다. 두 번째 트리 계층은 B^+-트리 색인 구조와 동일하고 각 노드에 저장된 데이터에 대한 색인을 구성한다. 이러한 구성에서 온-라인 재조직을 수행하는 과정은 다음과 같다[4, 5, 7, 8].

1) 프로세싱 노드(Processing Node)는 클러스터를 구성하는 각 Processing element로 독립된 메모리와 디스크를 갖는다.

1) 각 노드의 작업 큐에 한계값 이상의 클라이언트 질의가 집중된 경우나 응답 속도가 느려진 경우 데이터에 대한 온-라인 재조직을 결정한다.
2) 인접한 왼쪽이나 오른쪽 노드 중 부하가 적은 노드를 목적 노드(Target Node)로 선택하고 과부하가 발생된 소스 노드(Source Node)의 B+트리 색인 중에서 이동할 데이터의 부분을 결정하고 데이터 이동(Data Migration)을 수행한다.
3) 목적 노드의 기존의 색인과 이동된 데이터와의 통합을 수행한다. 이때 이동된 데이터만을 가지고 벌크 로딩(Bulk Loading) 방법을 사용하여 새로운 색인을 구성하고 해당 노드의 색인과 높이를 고려해서 목적 노드의 색인과 통합을 수행한다.



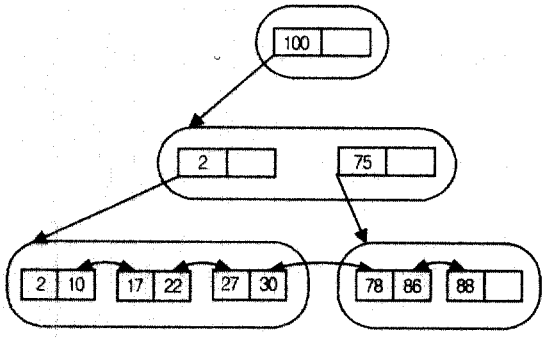
(그림1) 병렬 데이터베이스에서 온라인 재조직을 위한 색인 구조

위에서 언급한 B+-트리 색인에 기반한 온라인 재조직 기법은 다음과 같은 문제점을 갖는다. 첫째, 클러스터를 구성하는 있는 프로세싱 노드들 중에서 오직 하나의 노드에 부하가 집중되는 경우만을 가정하고 있다. 즉, 실제로 발생 빈도가 높은 인접한 노드들이 동시에 부하가 집중되는 경우에 대해서, 효율적인 온라인 재조직 방법이 제시되지 못하였다. 둘째, 온라인 재조직 기법에 온라인 확장 개념을 고려치 않았다. 고정된 노드 내에서만 데이터의 이동을 고려함으로써, 경우에 따라 반복적인 데이터의 이동을 발생시켜 온라인 재구성 트랜잭션의 수행 시간을 길게 하고 다른 일반 트랜잭션 수행을 방해하는 문제점을 갖는다.

본 논문에서는 기존의 B+-트리 색인 기반의 온라인 재조직 기법에서 문제점으로 제기된 두 개 이상의 노드에서 과부하가 발생한 경우, 클러스터를 구성하는 활성화 상태의 노드에서만 데이터 재조직을 수행하는 것이 아니고 온라인 확장을 통해 새롭게 추가된 노드를 포함하여 데이터 재조직을 수행함으로써, 기존 기법에 비해 반복적인 데이터의 이동을 없애고 작은 양의 데이터 이동만으로 빠르고 효율적인 온-라인 재조직을 수행하도록 한다.

2.2 메모리 상주 데이터베이스에서의 B+-트리 색인

메모리 상주 데이터베이스의 색인 구조에 있어서 캐시의 작용이 중요한 부분을 차지함으로 캐시를 고려한 색인 기법이 많이 연구되어 왔다[6, 9, 10]. CSB+-트리 색인(Cache Sensitive B+-Trees)은 이러한 특성을 고려하여 제안된 B+-트리 색인 중 하나로써 빠른 검색뿐만 아니라 갱신에도 큰 부하를 갖지 않는 색인이다. 기본적으로 기존의 B+-트리 구조를 기본 바탕으로 삼는다. 그러나 디스크 기반의 시스템에서 페이지 구조와 같이 메모리 기반의 시스템에서는 기본 전송 단위가 캐시 라인이다. 또한 기존의 B+-트리가 모든 자식노드의 포인터들을 명시적으로 저장한 것과는 달리 캐시를 고려한 B+-트리는 (그림 2)와 같이 같은 레벨에 위치한 자식 노드들을 그룹으로 묶고 그 첫번째 자식 노드의 주소 값만을 명시적으로 표현하여 저장한다. 이와 같은 구조는 기존의 B+-트리 색인 보다 캐시 라인에 저장할 수 있는 키 값의 개수를 증가시킴으로써 접근 속도를 보다 빠르게 한다.



(그림 2) CSB+-트리 색인의 데이터 구조

본 논문에서 제안하고자 하는 온라인 재조직 기법의 적용 환경은 이미 앞 절에서 설명한 병렬 데이터베이스 시스템 환경에서의 B+-트리 색인의 재조직 기법과 다소 차이가 있다. 즉, 제안된 기법의 환경은 클러스터를 구성하는 각 노드의 데이터베이스가 빠른 데이터 접근과 동시성 보장을 위한 메모리 상주 데이터베이스이다. 따라서, 본 논문에서는 본 절에서 설명된 캐시를 고려한 CSB+-트리(Cache Sensitive B+-Tree) 색인에 기반한 온라인 재조직 기법을 제안한다. 소스(Source) 노드에서 목적(Target)노드로 데이터를 이동시킬 때마다 캐시의 특성을 반영한 색인을 유지함으로써, 재조직 트랜잭션뿐만 아니라 일반 검색 질의 및 갱신 질의에 대해서 빠른 응답 시간을 보이도록 하는데 그 목적이 있다.

3. 확장 가능한 데이터베이스 클러스터 시스템의 아키텍처

본 장에서는 본 논문의 전개상 미리 언급되어야 할 확장

가능한 고가용 데이터베이스 클러스터 시스템인 최대 결함허용 보장 데이터베이스 클러스터 시스템(*Ultra Fault-Tolerant Database Cluster*)의 전체 구조와 환경에 대해서 설명한다.

3.1 비공유 구조

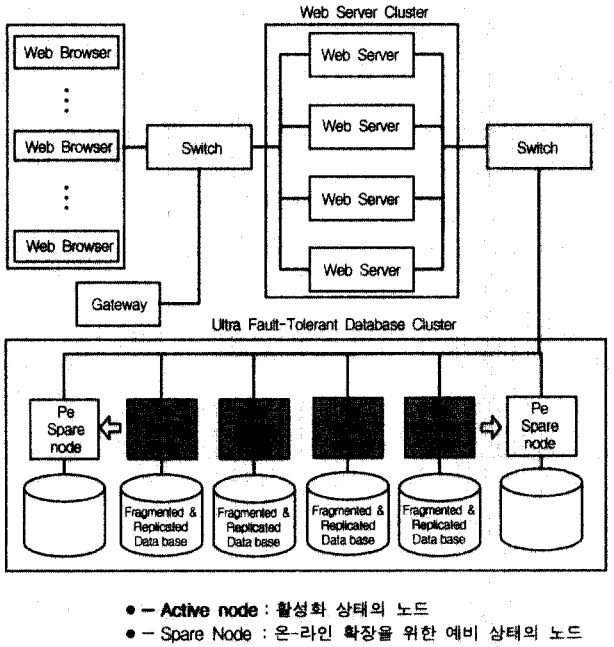
본 시스템의 구조는 비공유 환경을 가정한다. 즉 높은 대역폭과 네트워크 지연이 거의 없는 고속의 랜(LAN) 망을 통해 지리적으로 가까운 여러 개의 노드가 서로 연결되어 클러스터를 구성하고 각 노드는 독립적인 주기억장치와 메모리, 저장장치를 갖는 워크스테이션급으로 구성된다.

3.2 데이터 분할

본 시스템의 데이터는 분할(Fragmentation)과 복제(Replication)라는 두 가지 정책을 혼합 사용하여 각 프로세싱 노드의 독립된 저장장치에서 관리된다. 분할의 방법으로 테이블의 특정 키 값을 통해 범위에 의하거나 해쉬 함수를 통해 각 노드에 저장한다. 또한 테이블의 크기(레코드 수) 여부에 따라서 분할에 참여하는 노드 그룹이 결정되고, 노드 그룹을 구성하는 각 노드에 분할된 데이터가 저장된다. 복제는 시스템의 가용성을 높이기 위한 정책으로 분할된 데이터를 다른 노드에 복사본을 저장하는 방법이다. 복사본의 경우 두개가 가장 이상적이라는 연구 결과를 따라서 하나의 마스터(Master)에 대해서 하나의 백업(Backup)을 구성한다[3].

3.3 노드의 온라인 확장

시스템의 가용성을 높이기 위한 방법으로 온라인 확장을 지원한다. 즉, 클러스터를 구성하고 있는 여러 노드들을 역할에 따라서 두 가지 타입으로 나누는데 실시간 클라이언트 질의를 수행하고 있는 노드를 활동 노드(Active Node)라 하고 시스템의 확장을 위해서 사전에 등록되어 확장을 준비하고 있는 노드를 예비 노드(Spare Node)라 한다. 활동 노드 중에서 특정 노드가 고장을 일으킨 경우나 부하가 집중되는 경우, 예비 노드를 활성화 상태로 만들고 예비노드에 기존의 분할 및 복제되어 있던 데이터에 대한 온-라인 재조직을 수행한다. 이로써 전체 시스템의 가용성을 높이고 병목현상을 제거하여 전체 트랜잭션 처리량을 증가시키고 평균 응답 시간을 빠르게 한다(그림 3).

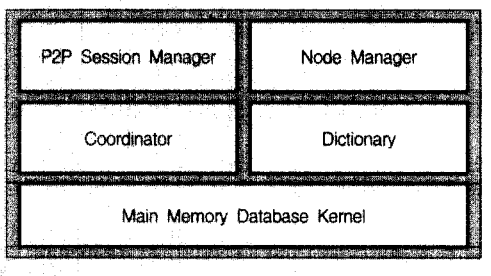


● – **Active node** : 활성화 상태의 노드
● – Spare Node : 온-라인 확장을 위한 예비 상태의 노드

(그림 3) 최대 결함허용 보장 데이터베이스 클러스터(Ultra Fault-Tolerant Database Cluster)의 아키텍처

3.4 주요 컴포넌트

각 프로세싱 노드의 세부 컴포넌트는 (그림 4)와 같이 피어투피어 세션 관리자(Peer-To-Peer Session Manager), 노드 관리자(Node Manager), 조정자(Coordinator), 데이터사전(Dictionary), 메모리상주 DBMS 커널(Main-memory DBMS kernel)로 나눌 수 있다. 피어투피어 세션 관리자는 다시 각 노드간의 데이터와 로그정보, 제어 메시지 전송을 위한 피어 채널(Peer Channel)과 클라이언트의 직접적인 질의를 처리하는 클라이언트 채널(Client Channel)로 구성된다. 노드관리자는 각 노드의 시스템 정보, 질의 패턴, 접근되는 데이터에 대한 통계정보를 관리함으로써 자신의 노드에 걸리는 부하를 체크하고 온라인 확장의 실행 여부를 결정한다. 조정자는 두 개 이상의 노드를 참조하는 트랜잭션을 처리하면서 임시적 데이터 공간을 관리한다. 데이터사전은 데이터의 분할과 복제 정보를 저장 관리하면서 트랜잭션 처리시에 참조된다. 각 트랜잭션은 디스크기반 DBMS에 비해 보다 빠른 질의 응답 시간을 갖는 메모리 상주 DBMS 커널을 통해 처리되며 빠른 데이터 접근을 위한 메모리 기반 색인으로 CSB^+트리가 내부적으로 구성된다.



(그림 4) 각 프로세싱 노드의 주요 컴포넌트

4. 온라인 확장을 고려한 CSB^+트리 색인에 기반 한 재조직 기법

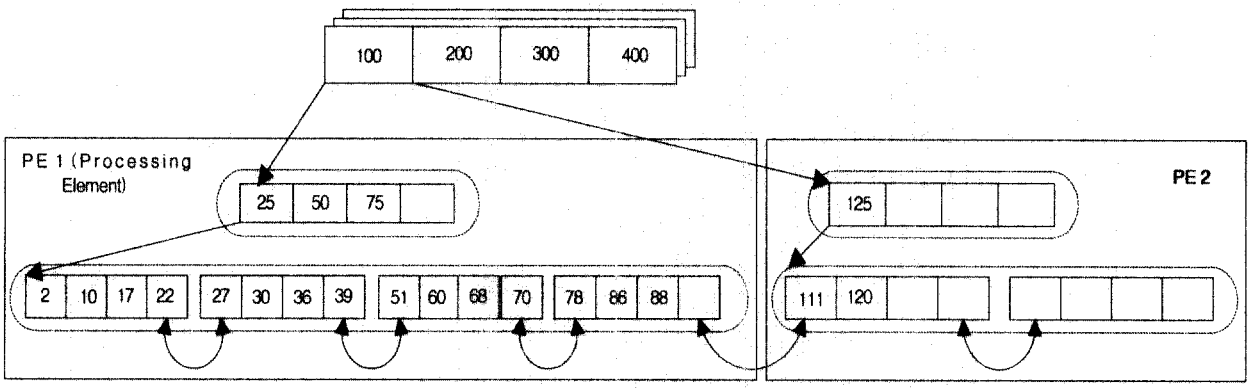
4.1 재조직 예

본 절에서는 기존의 기법과 제안된 기법을 예를 통해 비교

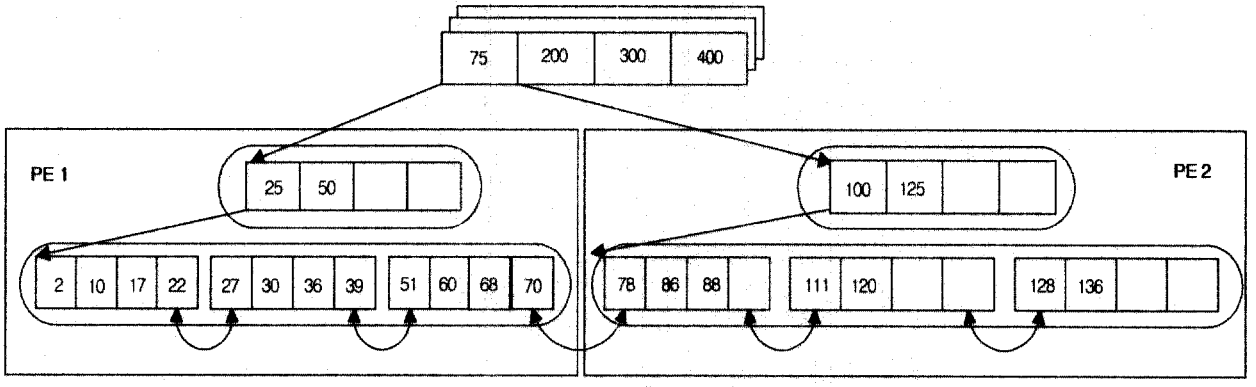
설명한다. 데이터는 범위 분할에 의해 각 노드에 분산되어 있기 때문에 하나의 프로세싱 노드에서, 인접 노드 또는 새로운 노드로 데이터를 이동한다. 각 노드의 데이터는 B+트리 색인으로 중복이 없고 이동되는 데이터는 해당 노드의 앞부분 또는 뒷부분이 된다. (그림 5)에서, 노드 PE 1에는 노드 PE 2보다 많은 데이터가 집중되어 있다. 따라서 PE 1은 PE 2 보다 일반적인 질의 요청이 많아 질 것이다. 이 때 PE 1에 저장된 데이터의 일부를 인접 노드로 이동시킴으로써 작업량의 집중으로 인한 성능의 저하를 막을 수 있다. (그림 6)은 (그림 5)에서 노드 PE 1의 병목현상을 막기 위해 두개의 노드를 재조직 한 것이다.

또한, (그림 6)의 상태에서 클라이언트 질의의 패턴이 변경되어 다수의 클라이언트 질의가 0~75 범위의 정수 값만을 검색한다면 PE 1에 그 범위의 데이터들이 대부분 존재하므로 PE 1에 과부하가 발생하고 다시 인접 노드로 데이터의 이동을 수행해야 한다. 위의 두 가지 경우 모두 기존의 재조직 기법으로 해결이 가능하다.

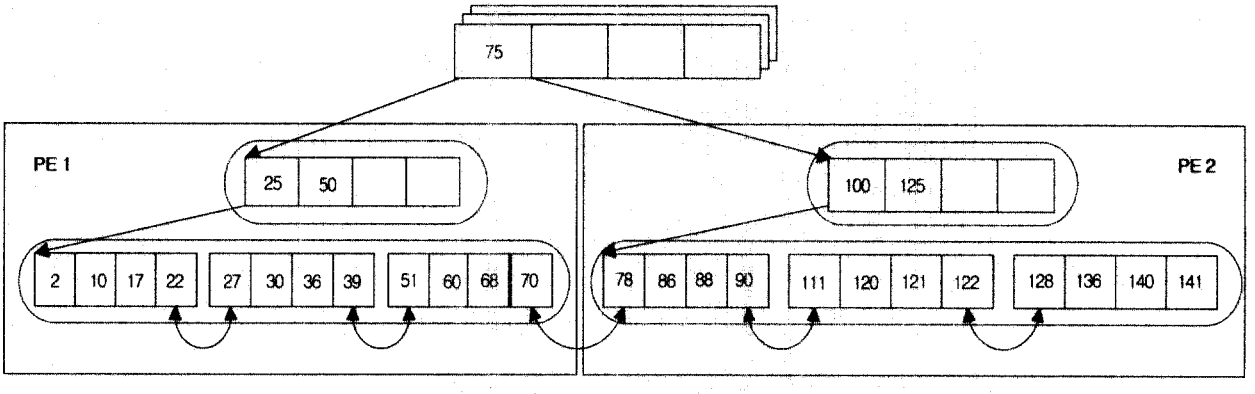
그러나, (그림 7)과 같이 두 개의 인접 노드에 저장된 데이



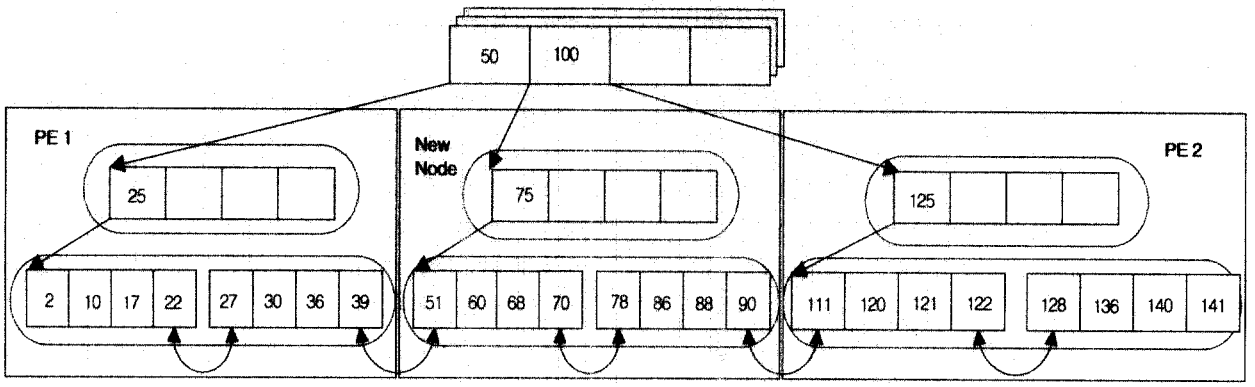
(그림 5) 온-라인 재조직을 수행하기 이전의 색인 구조



(그림 6) 온-라인 재조직을 수행한 이후 색인 구조



(그림 7) 두 개 이상의 인접 노드에 작업량이 집중된 경우의 색인 구조



(그림 8) 온-라인 확장을 통한 새로운 노드로의 데이터 이동

터의 양이 비슷하고, 두 개의 노드에서 동시에 과부하가 발생된 경우, 인접 노드들간의 데이터 재조직은 무의미하다. 즉, 온라인 증권거래와 같이 빠른 응답시간이 요구되거나 여러 대의 인접한 노드에 걸쳐 저장된 데이터에 대하여 질의가 폭주한다면 인접 노드간의 데이터 재배치 만으로는 노드들 간의 반복적인 이동만 발생되고 재조직 시간이 길어져서 다른 실시간 질의에 대한 응답시간이 지연되는 결과를 가져온다. 이를 해결하기 위해서는 인접 노드로 데이터를 분배하는 것이 아니라 온라인 확장을 통한 데이터의 재조직으로 작업량의 급격한 증가를 (그림 8)과 같이 해결할 수 있다.

4.2 재조직 정책

데이터에 대한 접근 빈도 수의 급격한 증가와 접근 패턴의 변화에 대응한 색인의 재조직 기법은 기존의 방법과 마찬가지로 다음의 세 단계로 구성된다.

1) 온-라인 재조직의 수행 결정
2) 소스 노드의 이동할 데이터의 양과 목적 노드의 결정
3) 이동된 데이터와 목적 노드의 데이터 간의 통합

1 단계는 기존의 방법과 동일하게 각 노드의 작업 큐에 한계값 이상의 클라이언트 질의가 집중된 경우나 응답 속도가 느려진 경우 온-라인 재조직의 수행을 결정한다. 2 단계는 두 가지 경우로 나누어 볼 수 있다. 첫째는, 기존의 가동중인 활동 노드(Active Node)들 안에서 색인을 재조직하는 경우이고 둘째는, 온라인 확장을 통한 예비노드(Spare Node)를 포함해서 각 노드의 색인을 재조직하는 경우이다. 이미 앞에서 언급한 대로 전자의 경우는 한번에 하나의 노드에 작업량이 집중된 문제는 해결할 수 있지만 두 개 이상의 노드에 한꺼번에 부하가 집중된 경우는 앞 절의 예처럼 인접노드로의 반복적인 데이터의 이동만 발생되고 근본적인 부하를 제거하지 못하는 한계점을 갖는다. 후자의 경우, 두 개 이상의 노드들의 작업량이 특정 한계 값을 초과한 경우 온라인 확장을 수행하여 기존의 노드와는 별도의 새로운 노드에 이동할 데이터의 양을 결정한다. 또한 3 단계에서는 기존의 기법은 B^+트리 색인을 구성하지만 제안된 기법에서는 CSB^+트리 색인을 구성함으로 데이터 이동 과정 중에 색인의 데이터 구조를 유지하기 위한 트리 재조직 과정이 필요하다.

4.3 확장된 CSB^+-트리 색인을 이용한 온라인 재조직 기법

4.1절에서 언급한 바와 같이 온라인 상태에서의 데이터 재배치는 각 서버에서의 색인 재조직에 따른 비용을 줄이는 것이 중요하다. 이 때 각 노드간의 트리의 높이 차이가 클 경우 재조직의 비용은 증가하게 된다. 이를 해결하기 위해 제안된 기법은 루트 노드의 저장 능력을 높여 트리의 높이가 높아지는 것을 억제 시킴으로써 재조직의 비용을 줄인다. 확장된 CSB^+-트리 색인의 노드 구성은 다음과 같다.

- *nKey* : 노드 내에 있는 키의 개수
- *firstChild* : 자식 노드 그룹에 있는 첫 번째 자식노드의 주소값
- *keyList*[2*d*] : key 값

명시적인 자식 노드의 포인터가 하나 뿐이므로 나머지 자식 노드들을 탐색하기 위해서는 오프셋 값을 그 firstChild에 더하여 자식 노드들의 주소를 알아 낼 수 있다. 이를 위해는 같은 레벨에 있는 노드 그룹이 모두 물리적으로 인접한 공간에 저장되어 있어야 하는데 이는 삽입 연산 발생 시 해당 노드의 분할이 필요한 경우 노드 그룹의 크기보다 노드 하나 크기가 덧붙여진 공간에 삽입하고자 하는 레코드와 해당 노드의 레코드를 저장한 후 다시 메모리에 그 크기 만큼의 연속적인 공간을 재할당 받아야 한다. 또한 다른 서버로의 데이터 분배 시 트리의 오른 쪽 부분이 이동되어야 할 경우에는 공간의 재활용이 가능하지만 왼쪽 부분의 이동이 일어날 경우에는 앞에서 언급한 것처럼 공간의 재할당이 필요하다.

(알고리즘 1)과 (알고리즘 2)는 클러스터를 구성하는 노드들

사이에 부하 불균형이 발생된 경우에 소스노드(Source Node)와 목적 노드(Target Node) 사이에 이동시킬 데이터에 대한 알고리즘이다. (알고리즘 1)은 데이터 이동을 위한 소스노드와 목적 노드를 선택하는 알고리즘이다. 우선 과부하가 발생된 노드를 찾아서 extract_key()라는 함수를 통해 옮길 데이터를 선택하고 transmit()라는 함수를 통해 데이터를 목적 노드로 전송한다. 또한 옮긴 데이터에 대해서는 delete_branch()라는 함수를 통해 소스 노드에서 제거하고 클러스터를 구성하는 예비노드(Spare Node)를 활성화 시키기 위해 online_scaling()라는 함수를 호출한다.

```
Algoritm Remove_Branch ()
 PE : an array that records load and index information in each PE
      (Processing Element) ;
 THRESHOLD : minimum value of overload PE ;   // 과부하 결정
              을 위한 한계 값
 THRESHOLD_GAP : threshold of gap between source and destina-
                  tion load
                  // 과부하 차의 한계 값
 Source_PE : an array that records No of overloaded PE ;   // 과부
             하가 발생한 노드의 번호 배열
 NUM_Source_PE : number of overloaded PE ;   // 과부하가 발생한
                 노드의 수

 NUM_Source_PE = 0 ;
 /* Determine the source PE */
 for (int i = 1 ; i < NUM_PE ; i++) {
    if ( ( PE[i].Load > THRESHOLD ) {
       Source_PE[NUM_Source_PE] = i ;
       NUM_Source_PE++ ;
    }
 }
 if (NUM_Source_PE == 0) exit ;   // 과부하가 발생한 노드가 없
    는 경우 종료
 for ( int j = 0 ; j < NUM_Source_PE ; j++ ) {   // 과부하가 발생된
    노드만큼 반복 수행
   source 1 = Source_PE[j] ;
   if ( PE[source 1 +1].Load > PE[source1 -1].Load )
   // 부하가 적은 인접 노드를 목적 노드로 선택
      destination = source 1 - 1 ;
   else
    destination = source1 + 1 ;

   // 인접노드에도 동시에 과부하가 걸린 경우
   if ( | PE [source 1].Load - PE [destination].Load | < THRESHOLD
      _GAP ) {
      if ( PE [source 1 + 1].Load > PE [source 1 -1].Load )
        source 2 = source 1 + 1 ;
      else
        source 2 = source 1 - 1 ;

      if ( source 1 > source 2 ) {   // source 1과 source 2를 오름차순
         으로 결정
        temp = source 2 ;
        source 2 = source 1 ;
        source 1 = temp ;
      }
      destination = online_scaling() ;   // 온라인 확장을 통한 새로
                    운 목적 노드 결정
   }
   else source 2 = source 1 ;   // 인접 노드에 과부하가 발생하지 않은
                  경우
   /* two to two Reorganization */   // 기존의 기법과 동일하게
      수행
   if (source 1 == source 2) {
      Keys = extract_keys ( PE [source 1].Root ) ;   // 소스 노드에서
             옮길 데이터를 추출한다.
      // 데이터를 목적 노드로 옮겨서 목적노드의 색인에 삽입한다.
      transmit ( destination, add_branch ( Keys ) ;
      // 소스 노드에서 옮긴 데이터를 삭제한다.
      delete_branch( PE [source 1].Root ) ;
    }
   /* two to three Reorganization */
   else {   // 제안된 기법에 의해 수행
      Keys 1 = extract_keys ( PE [source 1].Root ) ;   // 소스 노드 1
               에서 옮길 데이터를 추출한다.
      Keys 2 = extract_keys ( PE [source 2].Root ) ;   // 소스 노드 2
               에서 옮길 데이터를 추출한다.
      // 데이터를 목적 노드로 옮겨서 목적노드의 색인에 삽입한다.
      transmit (destination, add_branch ( Keys1 ) ;
      transmit (destination, add_branch ( Keys2 ) ;

      // 소스 노드 1에서 데이터를 삭제하기 전에 색인을 재조직한다.
      rebuild_CSBTree ( PE [source 1] ) ;
      // 소스 노드 2에서 데이터를 삭제하기 전에 색인을 재조직한다.
      rebuild_CSBTree ( PE [source 2] ) ;

      delete_branch ( PE [source 1].Root ) ;   // 소스 노드 1에서 옮긴
                  데이터를 삭제한다.
      delete_branch ( PE [source 2].Root ) ;   // 소스 노드 2에서 옮긴
                  데이터를 삭제한다.
   }
 } // end of for
```

(알고리즘 1) 소스노드의 색인에서 옮길 데이터를 찾아서 제거하는 알고리즘

(알고리즘 1)을 보다 자세히 살펴보면 먼저 과부하가 발생한 노드의 개수를 구하고 각 노드의 번호를 구한다. 다음은 과부하가 발생된 노드 만큼 반복 수행을 하는데, 과부하가 발생한 소스 노드에 대해서 목적 노드를 결정한다. 이때 인접 노드의 부하를 검사해서 인접 노드에 부하가 발생하지 않은 경우 이를 목적 노드로 결정하고 기존의 기법과 동일하게 두 개 노드의 데이터를 재조직하는 과정을 수행한다(two to two Reorganization). 그러나, 인접 노드에도 부하가 발생한 경우 온-라인 확장을 통해 새롭게 할당된 노드를 목적노드로 결정하고 처음의 과부하가 발생한 소스노드와 인접한 노드를 두 개의 소스 노드로 선택해서 세 개 노드의 데이터를 재조직하는 과정을 수행한다(Two to Three Reorganization).

```
Algoritm Add_Branch ( Keys )
   n : the number of index entries in the root node of index at
       destination
   Keys : a set of keys transmitted from source PE

   // 목적 노드의 색인과 통합하기 전에 벌크 로딩을 통해 새로운 색
      인을 생성한다.
   Pnew = bulk_load ( Keys ) ;
   /* Integrate new index with the index of destination PE */
   if ( IsRight ( Pnew, PE [destination].Root ) ) // 이동된 데이터가 기
   존 노드의 오른쪽 부분이 되는 경우
```

```
    PE [destination].Root -> P_{n+1} = PE[destination].Root -> P_n ;
    for ( i = n-1 ; i >= 0 ; i-- ) {
        PE [destination].Root -> P_{i+1} = PE[destination].Root -> P_i ;
        PE [destination].Root -> K_{i+1} = PE[destination].Root -> K_i ;
    }
    PE [destination].Root -> P_0 = P_new ;
    PE [destination].Root -> K_0 = find_separator ( ) ;
}
else {   // 이동된 데이터가 기존 노드의 왼쪽 부분이 되는 경우
    PE [destination].Root -> K_n = find_separator ( ) ;
    PE [destination].Root -> P_{n+1} = P_new ;
}
rebuild_CSBTree ( PE [destination] ) ;   // 색인의 재조직
```

(알고리즘 2) 목적 노드의 색인에 이동된 데이터를 합치는 알고리즘

(알고리즘 2)는 소스노드로부터 이동된 데이터를 먼저 bulk_load( )라는 함수를 통해 새로운 색인을 생성한다. 이때 CSB$^+$-트리 색인의 속성을 유지하면서 벌크 로딩을 수행한다. 다음은, 생성된 색인을 목적 노드에 존재하는 기존의 색인과 통합하고 색인의 식별자를 적절하게 수정한다.

```
Algoritm rebuild_CSBTree ( PE [Target] )
  /* addData : 이동시킬 데이터가 있는 노드의 최상위 노드 주소 */
  If ( side == right ) {
     if ( nKey == 1 )
        addData = PE[Target].root -> firstChild ;
     else
        addData = S.root -> firstChild + PE [Target].root -> nKey *
                  offset ;
     PE [Target].root -> firstChild -> nKey = PE [Target].root ->
                  firstChild -> nKey - 1 ;
  }
  else {
     addData = PE [Target].root -> firstChild ;
     new node = malloc ( 2 * order * offset + sizeof (nKey)
              + sizeof (firstChild) ) ;
     new node = PE [Target].root -> firstChild를 제외한 모든 노드들
     PE [Target].root -> firstChild -> nKey = PE [Target].root ->
                  firstChild -> nKey - 1 ;
     PE [Target].root -> firstChild = new node ;
  }
```

(알고리즘 3) CSB$^+$-트리 색인을 재조직하는 알고리즘

(알고리즘 3)는 CSB$^+$ 트리 재조직을 보여주기 위한 것이다. 트리 내부에서 이동시킬 데이터의 양이 결정된 경우 작업량이 특정 서버에 집중될 경우 데이터를 오른쪽 서버로 이동할 것인지 왼쪽으로 이동할 것인지에 대한 판단을 한다. 이때 전체 트리 구조에서 왼쪽 부분의 노드를 이동시킬 경우 물리적으로 인접 공간의 위치에 노드 그룹이 위치해야 하는 CSB$^+$ 트리의 특징으로 인해 새로 연속적인 공간을 할당 받아 재 저장해야 하고 부모 노드의 firstChild 값을 변경해 주어야 한다.

5. 성능 평가

본 장에서는 제안된 온라인 재조직 기법의 성능을 평가한다. 성능 평가를 위해 3장에서 제시한 적용 시스템 환경과 동일한 성능평가 모형에 의한 모의 실험과 실제 구현 시스템에서의 성능을 병행 수행한다. 모의 실험은 구현 시스템에서의 제한된 프로세싱 노드 수에 제한을 받지 않는다는 점에서 MCC에서 개발한 CSIM 언어[11, 13]를 이용하여 수행되었다.

성능 평가의 비교 대상으로는 2장에서 설명한 병렬 데이터베이스에서 B$^+$-트리 색인에 의한 온라인 재조직 기법과 비교하는데, 기존 기법과 제안된 기법에서의 기반 색인이 서로 다르므로 기존 기법의 색인을 B$^+$트리에서 CSB$^+$트리로 변경한 후, 재조직 기법의 성능을 비교 평가한다. 모의 실험에 사용된 성능 평가 지수는 온라인 재조직을 완료하는데 필요한 각 노드간 데이터 이동 회수와 전체 트랜잭션의 응답시간에 미치는 영향이다. 즉, 온라인 재조직 트랜잭션이 시작되어 완료되기 전까지 클러스터를 구성하고 있는 노드들에서 발생되는 데이터 이동 회수를 측정하여 온라인 재조직 비용을 평가한다. 또한 온라인 재구성 중에 발생되는 모든 트랜잭션의 수행 응답 시간을 측정하여 온라인 재조직이 시스템 전체에 미치는 영향을 평가한다. 모의 실험 결과의 신뢰도를 높이기 위해 배치평균기법(Batch mean method)을 사용하여 50개의 다른 seed로 산출된 결과들의 평균값을 측정한다. 각각의 실험은 10000개의 트랜잭션 중 초기 100개가 완료가 될 때 까지의 결과들은 무시함으로써 산출된 결과가 90%의 신뢰수준을 만족하도록 하였다.

제안된 기법의 성능 평가를 위해 본 모의 실험에서 사용한 시스템 환경으로 32개의 활동노드와 8개의 가용 노드를 갖는 확장 가능한 비공유 구조의 메모리 상주 데이터베이스 클러스터를 가정한다. 각 노드의 프로세스는 1Gbps의 내부 네트워크 망을 통해 메시지와 데이터를 서로 교환한다. <표 1>은

〈표 1〉 실험환경

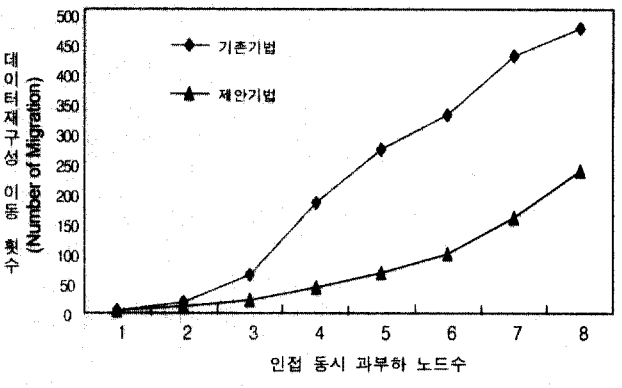
시스템 속성	
네트워크 속도	1G bps
클러스터에 속한 프로세싱 노드 수	활동 노드 32, 가용노드 16
프로세싱 노드의 시스템 사양	PentiumIII 700 CPU, 1GB Memory
캐시크기, 캐시라인(C)	16KByte, 64Byte
데이터베이스 속성	
색인의 노드 크기	64Byte (캐시라인과 동일한 크기)
레코드의 수	10,000,000
키의 크기(K)	4Byte
질의 수(Q)	10,000
색인의 노드당 가지 수(m)	14
인접 동시 부하 집중 노드 수	1, 2, 3, 4, 5, 6, 7, 8

세부적인 실험 환경을 요약한다. 각 매개 변수의 구체적인 값은 [5]와 [10]을 참조하여 기존 기법에서 수행한 실험과 비슷한 조건을 구성하였다. 그 이유는 기존 기법이 Zipf 분산에 의해 질의를 발생시키는데 반해 제안된 기법에서는 인접 동시 과부하 노드가 발생되도록 질의를 생성시킨다는데 차이가 있고 나머지는 동일하기 때문이다.

위 표에서 캐시라인의 크기가 64Byte이므로 캐시를 고려한 B+색인의 노드 크기를 64Byte로 하고 레코드의 크기 값이 4Byte이므로 색인의 각 노드에는 14개의 데이터 키 값과 1개의 데이터 개수, 서브 자식노드에 대한 포인터 한 개를 갖는다. 본 모의 실험을 위해 다음의 두 단계를 수행한다. 먼저 1단계로, 초기의 CSB+트리를 생성한다. 키의 크기로 4byte를 갖는 10,000,000개의 레코드를 클러스터에 포함된 활동 노드에 고르게 무작위로 분포 시킨다. 그리고 나서 10000개의 좁은 범위의 검색 질의를 발생시켜 특정 노드에 부하가 집중되도록 한다. 특정 노드에 부하가 집중되면 온라인 재조직 트랜잭션이 수행되고 부하가 집중된 노드의 데이터를 인접 노드로 이동시킨다. 실제로, 주어진 수 만큼의 노드에 트리를 구성하고 주어진 레코드로 테이블을 구성함으로써 과부하시 이동되는 데이터의 개수와 범위를 산출할 수 있는데, 이러한 정보를 사용하여 두번째 단계를 수행한다. 즉, 두번째 단계로, 1단계에서 구해진 정보와 CSIM 언어를 사용하여 모의 실험 모델을 구현한다. 이때, 각 노드들은 자원(Resource)으로, 질의는 엔터티(Entity)로 모델링 한다. 그리고 구현된 모의 실험 모델에서 위에서 언급한 온라인 재조직을 완료하는데 필요한 각 노드간 데이터 이동 회수와 전체 트랜잭션의 응답시간에 미치는 영향을 측정한다.

첫 번째 실험으로 32개의 활성화 상태의 활성노드(Active Node)와 확장을 위한 16개의 여분 상태의 가용노드(Spare Node)로 구성된 클러스터에 10000개의 범위 질의를 수행하는데, 기존의 기법에서는 평균부하[2]의 15%이상이 되는 과부하 노드가 1개가 되도록 질의를 생성시켰지만 본 실험에서는 다중노드의 과부하 문제를 모델링하기 위해 동시 과부하 노드 수가 1개, 2개, 3개, …, 8개가 되도록 질의를 발생시키고, 이때 제안된 기법과 기존 기법의 데이터 이동 횟수(데이터 이동에 따른 I/O 발생 페이지 수)를 측정하였다. 데이터 이동의 경우 기존 연구의 결과를 반영하여 데이터의 크기를 고정적으로 하기 보다는 유동적으로 적용하였다.

(그림 9)는 인접 동시 과부하 노드 수를 변화시킬 때, 각 기법의 재구성 비용의 변화를 나타낸다. 인접 동시 과부하 노드 수가 증가할수록 제안된 기법이 기존 기법에 비해 적은 횟수의 데이터 이동만으로 과부하 문제를 해결함을 보인다. 그 이유는 인접 동시 과부하 노드 수가 증가할수록 부하 분산을 위해 수행하는 데이터 이동이 과부하 노드에서 인접 노드로 이동될 때, 목적 노드 또한 과부하 상태의 노드가 될 확률이 커지기 때문이다. 기존 기법의 경우, 과부하 발생 노드에서 왼쪽 또는 오른쪽의 인접 노드 중에서 부하가 적은 노드를 목적 노드로 선택하여 색인의 일정 부분을 이동시킨다. 그 결과, 과부하 노드간의 데이터의 반복적인 이동이 불가피하고 재조직 비용이 커지게 된다. 본 논문에서 제안된 기법의 경우, 인접 노드가 과부하 상태의 경우, 온라인 확장을 통해 가용노드를 목적 노드로 해서 데이터를 이동시킴으로써 반복적인 데이터의 이동을 방지한다. (그림 9)에 나타나듯이 인접 동시 과부하 노드 수가 3개인 경우에서부터 두 기법간의 비용의 차이가 발생하여 노드 수가 늘어날수록 더 많은 비용의 차이를 보임을 알 수 있다.

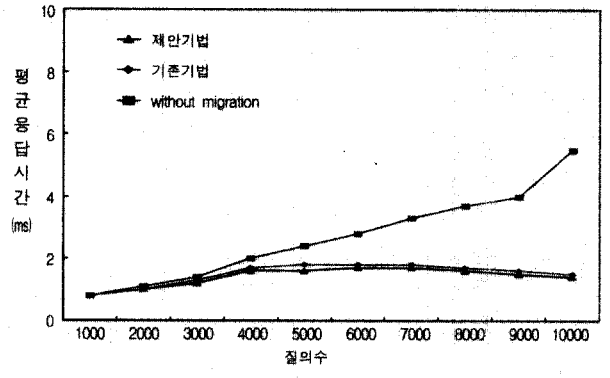


(그림 9) 동시 과부하 노드 수에 대한 데이터 재조직 횟수비교

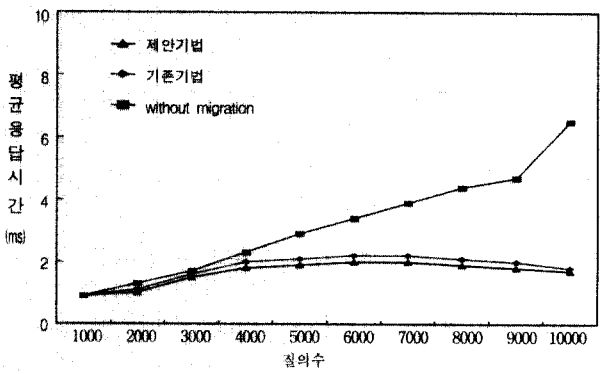
두 번째 실험으로, 인접하여 동시에 과부하가 발생한 노드의 개수를 2개, 3개, …, 8개로 늘려 가면서 온라인 재조직을 발생시키고 이때 수행되는 10000개의 전체 질의에 대한 시스템의 평균적인 응답 시간을 측정하였다. 즉, 데이터 이동이 질의 응답 시간에 미치는 영향을 분석하였다. 데이터 이동은 각각의 모든 프로세싱 노드의 작업 큐에 10개 이상의 질의가 기다리고 있기 전에는 발생하지 않도록 했다. 그렇지 않고 가장 많은 10개 이상의 질의가 작업 큐에 쌓인 노드는 과부하가 발생된 소스노드로 판단하고 인접한 노드 또는 새로운 확장 노드로 색인의 일정부분의 데이터를 이동 시킴으로써 시스템의 응답시간을 높이고 전체 트랜잭션의 생산량을 늘인다.

(그림 10)은 인접 동시 과부하 노드 수를 3, 4, 5, 6, 7, 8로 두었을 때, 10000개의 트랜잭션이 완료되는 동안 각 트랜잭션의 평균 응답 시간을 나타낸다. 기존 기법은 데이터 이동을 하지 않는 경우에 비해 선행 연구[5]의 결과와 일치하는, 약 60%의 향상된 결과를 보이고, 제안된 기법은 기존 기법에 비해 약 20% 향상된 결과를 보인다. 또한 인접 동시 과부하 노드 수가 많아질수록 각 기법간에 성능 차이가 더 크게 발생함을 알 수 있다. 그 이유는 10000개의 질의가 수행되면서 특

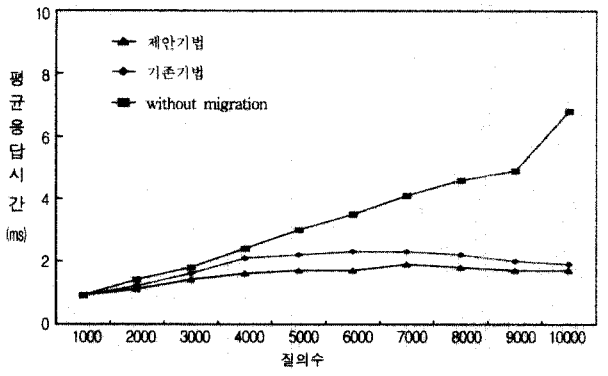
2) 평균부하는 클러스터 시스템에서 전체 질의의 수를 활성화 상태의 프로세싱 노드의 개수로 나누었을 때의 부하를 말한다.



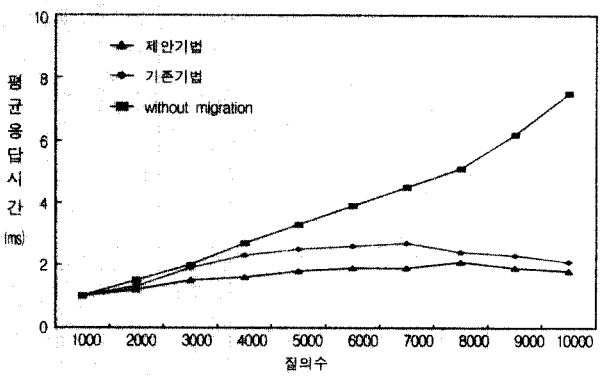
(a) 인접 동시 과부하 노드 수 = 3



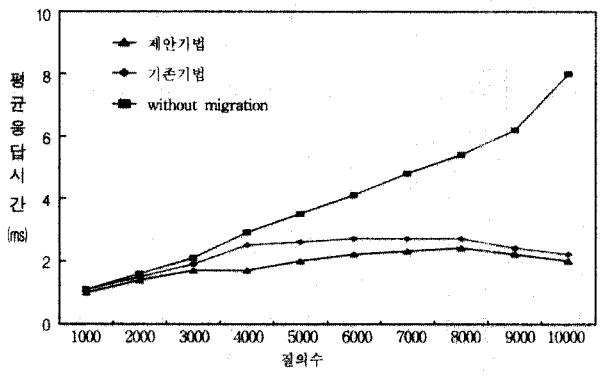
(b) 인접 동시 과부하 노드 수 = 4



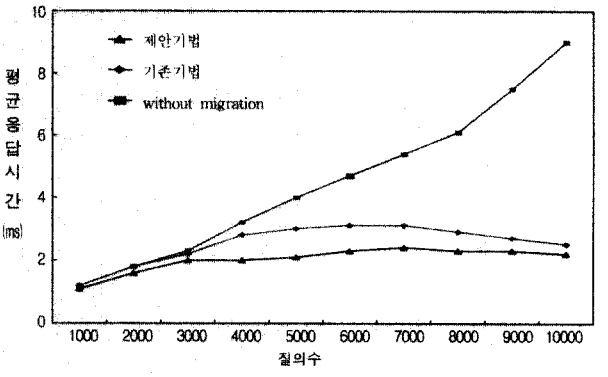
(c) 인접 동시 과부하 노드 수 = 5



(d) 인접 동시 과부하 노드 수 = 6



(e) 인접 동시 과부하 노드 수 = 7



(f) 인접 동시 과부하 노드 수 = 8

(그림 10) 최대 과부하 노드의 질의에 대한 평균응답시간

정 노드에 과부하를 발생시키고, 과부하 노드에서 처리되는 트랜잭션 처리 시간은 일반 노드에서 처리되는 시간보다 길기 때문에 전체 트랜잭션의 평균 응답 시간도 길어지게 된다. 온라인 재조직을 수행하지 않는 경우, 질의량이 많아질수록 응답시간은 점점 길어지게 되고, 온라인 재조직을 수행하는 경우, 과부하 노드의 부하를 줄여줌으로써 과부하 노드에서의 응답시간을 줄이고 전체 트랜잭션의 응답시간을 줄이게 된다. 또한 온라인 재조직 비용이 클수록 데이터 이동에 따른 추가 트랜잭션을 많이 발생시키므로, 동시 과부하 노드의 부하 분산시 비용이 적게 드는 제안된 재조직 기법이 기존 기법에 비해 평균응답시간이 더 빠르다. (그림 10)에 나타나듯이 인접 동시 과부하 노드 수가 많아 질수록 평균응답시간의 차가 커지는 것은 부하 분산에 필요한 비용 차가 점점 커지기 때문이다.

6. 결 론

본 연구는 비공유 데이터베이스 클러스터 환경에서 색인을 기반으로 한 시스템의 성능을 위한 튜닝 기술로서, 클러스터 내에 과부하가 발생된 노드들로부터 인접 노드나 온-라인 확장을 통해 새롭게 추가된 노드에 이동시킬 데이터의 양을 빠르게 결정하고 벌크로딩에 의해 색인을 통합함으로써 데이터

를 재조직하는 기법을 제안하였다. 제안된 기법은 클라이언트 질의의 급격한 증가나 질의 패턴의 변경에 따른 시스템의 성능 및 가용성의 저하에 대해서 데이터와 색인의 갱신을 최소화하면서 빠르게 재조직을 수행하도록 하였다.

본 논문에서는 기존의 재조직 기법에서의 문제점인 클러스터 안에 두개 이상의 다중 노드에 과부하가 집중된 경우, 부하 분배를 위해 많은 횟수의 데이터 이동이 필요하고, 전체 시스템의 질의에 대한 평균 응답 시간이 느려진다는 단점을 개선하였다. 즉, 클러스터를 구성하는 활성화 상태의 노드에서만 데이터 재조직을 수행하는 것이 아니고 온라인 확장을 통해 새롭게 추가된 노드를 포함하여 데이터 재조직을 수행하였다. 특정 노드에 집중되는 작업량을 인접 노드나 온라인 확장을 통해 추가된 노드에 분배하는 방식으로, 하나의 노드 부하 뿐만 아니라 기존 문제점으로 제기된 두 개 이상의 노드에서 과부하가 발생한 경우, 기존 기법에 비해 작은 양의 데이터 이동만으로 빠르고 효율적인 온-라인 재조직을 수행하도록 하였다. 또한, 기존의 기법에서는 병렬 데이터베이스 시스템 환경에서의 B^+-트리 색인의 재조직을 다루었지만 제안된 기법에서는 클러스터를 구성하는 각 노드의 데이터베이스가 빠른 데이터 접근과 동시성 보장을 위해 메모리 상주 데이터베이스 이므로 캐시를 고려한 CSB^+-트리(Cache Sensitive B^+-Tree) 색인의 재조직을 수행하였다. 소스(Source) 노드에서 목적(Target)노드로 데이터를 이동시킬 때마다 캐시의 특성을 반영한 색인을 유지함으로써 실시간 검색 질의 뿐만 아니라 갱신 질의에 대해서 빠른 응답 시간을 보이도록 하였다.

본 논문에서는 제안된 기법의 성능 평가를 위해서 확장 가능한 고 가용성 데이터베이스 클러스터 시스템으로 개발된 Ultra Fault-Tolerant Database Cluster라는 시스템을 실험 환경으로 해서 온-라인 재조직을 완료하는 시간(데이터 이동 회수)과 트랜잭션의 평균 응답시간을 정량적으로 평가하였다. 실험 결과에 의하면 제안된 기법이 기존 기법에 비해 적은 횟수의 데이터 이동만으로 과부하 문제를 해결하였고 클라이언트 질의에 대한 평균 응답 시간도 향상됨을 볼 수 있었다. 특히, 인접한 동시 과부하 노드 수가 많을수록 제안된 기법이 더 좋은 성능을 보였다. 그러나 제안된 기법은 라운드 로빈 방식이나 해싱 방법에 의해 분할된 테이블에 대해서는 적용이 될 수 없고 각 노드에 B-트리와 같이 범위 분할에 의해 분산되어 있는 테이블에 적용 가능한 재구성 기법이었다.

향후 연구로는 라운드 로빈 방식이나 해싱 방법에 의해 분할된 테이블에 대한 재구성 기법을 연구하고, 제안된 기법을 다양한 응용 시스템에 적용할 수 있도록 확장하는 것과 메모리 상주 데이터베이스의 다양한 색인에 본 기법을 적용해서 성능을 비교 평가함으로써 최적의 데이터베이스 클러스터를 구축하는데 있다.

참 고 문 헌

[1] K. J. Achyutuni, "Two Techniques for On-Line Index Modification in Shared Nothing Parallel Databases," Proceedings of the 1996 ACM SIGMOD, 1996.

[2] Roger Bamford, Rafiul Ahad and Angelo Pruscino, "A Scalable and Highly Available Networked Database Architecture," Proceedings of the 25th VLDB Conference, 1999.

[3] Svein Erik Brastsberg and Rune Humborstad, "Online Scaling in Highly Available Database," Proceedings of the 27th VLDB Conference, 2001.

[4] Mohana H. Lakhamraju, Rajeev Rastogi, S. Seshadri and S. Sudarshan, "On-line reorganization in object databases," Proceedings of 2000 ACM SIGMOD, pp.58-69, 2000.

[5] Mong Li Lee and Masaru Kitsuregawa, "Towards Self-Tuning Data Placement in Parallel Database Systems," Proceedings of the 2000 ACM SIGMOD, 2000.

[6] Tobin J. Lehman, "A study of index structures for main memory database management systems," Proceedings of the 12th VLDB, 1996.

[7] D. Lomet, "Replicatd indexes for distributed data," Proceedings of Conference on Parallel and Distributed Information Systems, pp.108-119, 1996.

[8] Nagavamsi Ponnekanti and Hanuma Kodavalla, "Online Index Rebuild," Proceedings of the 2000 ACM SIGMOD, 2000.

[9] Jun Rao and Kenneth A. Ross, "Cache conscious indexing for decision-support in main memory," Proceedings of the 25th VLDB, 1999.

[10] Jun Rao and Kenneth A. Ross, "Making B^+-Trees Cache Conscious in Main Memory," Proceedings of the 2000 ACM SIGMOD, 2000.

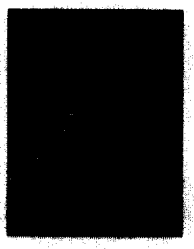
[11] H. Schwtman, CSIM User Guide for use with CSIM Revision 16, MCC, 1992.

[12] R. Vingralek, Y. Breitbart and G. Weikum, "Snow-ball : Scalable storage on networks of workstations," Distributed and Parallel Databases, 6(2), 1998.

[13] K. Watkins. Discrete event simulation in c. McGraw-Hill, 1993.

[14] C. Zou and B. Salzberg, "On-line reorganization of sparsely-populated b^+ trees," Proceedings of the 1996 ACM SIGMOD, 1996.

[15] C. Zou and B. Salzberg, "Safely and Efficiently Updating References during Online Reorganization," Proceedings of the 24th VLDB Conference, 1998.



이 충 호

e-mail : chlee@dblab.inha.ac.kr

1997년 인하대학교 전자계산공학과 졸업 (공학사)

1999년 인하대학교 대학원 전자계산공학과 졸업(공학석사)

1999~현재 인하대학교 대학원 전자계산공학과 박사과정

관심분야 : 데이터베이스 클러스터, 분산 데이터베이스, 공간 데이터베이스 등

배 해 영

e-mail : hybae@inha.ac.kr

1974년 인하대학교 응용물리학과 졸업 (공학사)

1978년 연세대학교 대학원 전자계산학과 졸업(공학석사)

1989년 숭실대학교 대학원 전자계산학과 졸업(공학박사)

1985년 Univ. of Houston 객원 교수

1992년~1994년 인하대학교 전자계산소 소장

1982년~현재 인하대학교 전자계산공학과 교수

1999년~현재 지능형 GIS 연구센터 소장

2000년~현재 중국 중경우전대학교 대학원 명예 교수

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