클러스터링 데이터베이스에서 온라인 확장을 고려한 CSB$^+$ 트리 색인의 온라인 재구성 기법

°심태정* 이충호* 이순조** 배해영*
*인하대학교 전자계산공학과
**서원대학교 컴퓨터 교육과
taing78@hanmail.net

Online Scaling Consious Online Reorganization of CSB$^+$ tree Index in a Database Cluster

°Tai-Jung Sim* Chung-Ho Lee* Sun-Jo Lee** Hae-Young Bae*
*Dept. of Computer Science and Engineering, Inha University
**Dept. of Computer Education, Seowon University

요 약

클러스터링 데이터베이스는 높은 가용성과 확장성을 갖으며, 예상치 못한 클라이언트 질의의 증가나 질의 패턴의 변경에 따른 작업부하의 편중에 효율적으로 대처할 수 있는 구조이다. 특히 온라인 확장 기법은 트랜잭션 처리를 중지하지 않고 새로운 노드를 클러스터에 추가하여 데이터를 재구성함으로써 임의의 노드에 질의가 집중되는 문제를 해결할 수 있다. 정적으로 구성된 시스템만으로는 두 대 이상의 서버에 작업량이 집중될 경우 재배치 시 서버 간의 데이터 이동의 반복 현상이 발생되며, 이로 인해 네트웍의 부하와 함께 실시간 트랜잭션의 처리에 있어서 응답 시간이 지연되는 문제점이 발생한다.

따라서 본 논문에서는 데이터 이동의 반복 현상을 해결하기 위해 클러스터링 데이터베이스에서 온라인 확장을 고려한 CSB+ 트리 색인의 온라인 재구성 기법을 제안한다. 제안된 기법은 온라인 확장을 통한 동적 노드의 확장으로 데이터 이동의 반복을 막고 새롭게 추가된 노드를 통해 빠르고 효율적인 데이터의 분산을 수행한다. 또한 각 시스템의 내부를 CSB$^+$ 트리로 구성하여 데이터의 재구성시에도 실시간 트랜잭션에 대한 빠른 응답 시간을 보장한다.

1. 서 론

인터넷 사용자의 증가로 대량의 데이터의 관리와 데이터 요청에 대한 빠른 응답 시간이 중요한 문제로 대두 되었다. 인터넷과 같은 동적인 환경에서는 특정 데이터에 대한 요청을 하는 사용자의 수가 단시간 내에 급격하게 증가할 수도 있고 예측할 수 없는 패턴으로 데이터를 요청할 수도 있다.

비공유(Shared-Nothing)구조의 데이터베이스 시스템은 데이터에 대한 질의 요청에 대해 효율적인 처리를 하기 위해 질의가 집중되지 않도록 배치가 되어야 한다. 그러나, 질의 패턴의 변화에 의해 초기의 데이터 분배 정책은 문제가 생길 수 있다. 이를 위해 온라인 상에서 유동적으로 확장할 수 있는 시스템이 요구된다.

각 서버에서는 효과적인 질의 처리와 갱신을 위해 데이터의 인덱싱이 필요하다. 메인 메모리의 대형화 추세에 의해 클러스터링 데이터베이스의 전체 인덱스의 메모리 적재가 가능하게 되었으며, 메모리를 효율적으로 활용하기 위해 캐시를 고려한 B$^+$ tree(Cache Sensitive B$^+$-Tree)가 사용된다. CSB$^+$ tree는 모든 자식 노드를 물리적으로 인접한 위치에 저장 시켜 그 첫번째 자식 노드의 포인터만을 명시적으로 노드에 저장하므로 공간적인 측면에서 효율적이며, 이러한 특징을 클러스터링 데이터베이스에 이용해 효율적인 질의의 분산을 가능케 한다.

그러나, 사용자 질의의 집중 현상이 단일 노드가 아닌 2개 이상의 노드에 발생할 경우 각각의 프로세스에서 수행하는 데이터 복사 과정에 의해 데이터 이동의 반복 현상이 나타난다.

본 논문에서는 클러스터링 데이터베이스에서 온라인 확장을 고려한 확장된 CSB$^+$ tree색인의 온라인 재구성 기법을 제안한다. 이 기법에서는 한 서버에 집중되는 작업량을 근접 서버나 온라인 상에서 확장된 새로운 서버에 분배하는 방식으로 특정 서버의 부하를 막는 동시에 각 서버 내부 데이터를 확장된 CSB$^+$ 트리 방식으로 구성하여 갱신에 대한 비용을 감소시키고 사용자의 질의에 빠르게 응답할 수 있도록 하며, 문제점으로 제기된 2개 이상의 서버에 집중되는 작업량에 대해서는 온라인 확장을 이용한 재구성을 하여 데이터 이동 횟수를 감소시키고, 전체 노드의 확장을 통해 안정적인 질의의 분산을 수행한다.

본 논문의 구성은 2장에서 이에 관한 관련연구를 다루고, 3장에서는 제안된 기법의 시스템 환경과 배경에 대해 언급한다. 4장에서는 제안된 재구성기법에 대해 설명하고 5장에서 성능 평가를 하고 마지막 6장에서 결론을 맺는다.

2. 관련 연구

2.1 병렬 데이터베이스에서의 온라인 재구성 기법

데이터는 초기에 모든 프로세스 단위에 걸쳐 분산되어 있다. 그러나 다른 데이터들보다 특정 데이터에 사용자의 접근이 집중적으로 이루어질 경우 데이터를 저장하고 있는 프로세스 단위에 급격한 작업량의 증가로 병목현상을 초래할 수 있다. 이러한 경우 특정 시스템에 집중되는 작업량을 다른 서버로 분배한다면 병목현상을 해결하여 사용자의 데이터 요청 질의에 대한 응답시간을 줄일 수 있다.

특정 서버에 몰린 작업량을 다른 서버로 분배하기 위해서는 각 서버에 있는 데이터의 재구성을 필요로 한다. 이를 살펴보면

첫째, 대량의 데이터가 특정 서버에 집중하여 저장되어 있는 경우

둘째, 특정 서버에 다수의 사용자 접속이 급속히 증가하는 경우

로 볼 수 있다. 이를 self tuning 방법을 사용해 부하를 저하시킨다.

2.2 메모리 상주 데이터베이스에서의 B$^+$ 트리 색인

CSB$^+$ tree는 기본적으로 기존의 B$^+$ tree 구조를 기본 바탕으로 삼는다 그러나 디스크 기반의 시스템에서 페이지 구조와 같이 메모리 기반의 시스템에서는 기본 전송 단위가 캐시 라인이다. 또한 기존의 B$^+$ tree가 모든 자식노드의 포인터들을 명시적으로 저장한 것과는 달리 캐시 sensitive B$^+$ tree는 같은 레벨에 위치한 자식 노드들을 그룹으로 묶고 그 첫번째 자식 노드의 주소값 만을 명시적으로 표현하여 저장한다. 이와 같은 구조는 기존의 B$^+$ tree 보다 캐시 라인에 저장할 수 있는 키 값의 개수를 증가시키고 노드의 분할이 발생할 경우의 모든 자식 노드 들의 포인터를 이동시켜야만 하는 오버헤드를 감소시킬 수 있다. 그러나 삽입 연산으로 인해 발생 되는 갱신의 경우 그룹으로 묶여진 노드들을 물리적으로 인

접한 위치에 저장시키기 위해 노드 그룹보다 노드 하나만큼의 크기가 더 큰 버퍼에 갱신된 데이터를 이동시킨 후 다시 인접한 물리적 주소들을 할당 받아 저장해야 하는 오버헤드가 발생하는 문제점이 있다.

3. 시스템 개요

본 장에서는 본 논문의 전개상 미리 언급되어야 할 전체 시스템 구조와 환경 및 배경에 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.

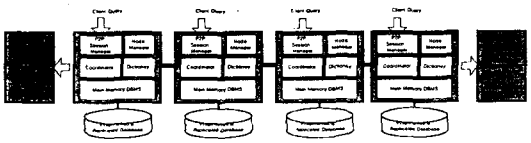
3.2 데이터 분할

본 시스템의 데이터는 분할(Fragmentation)과 복제(Replication)라는 두 가지 정책을 혼합 사용하여 각 노드의 독립된 저장장치에서 관리된다. 분할의 방법으로 테이블의 기본키를 통해 범위에 의하거나 해쉬 함수를 통해 각 노드에 저장한다. 또한 테이블의 레코드 수에 따라서 분할에 참여하는 노드의 수가 결정되고 각 노드별로 해당되는 부분만이 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 노드의 온라인 확장

시스템의 가용성을 높이기 위한 방법으로 온라인 확장을 지원한다. 즉, 시스템이 가동 중에 특정 노드가 고장을 일으킨 경우나 부하가 집중되는 경우 예비로 등록되어 있던 새로운 노드가 클러스터에 추가되어 활성화 상태가 되고 기존에 분할 및 복제되어 있던 데이터를 재 배치하고 인덱스를 재구성함으로써 시스템의 전체 트랜잭션 처리량을 증가시키고 평균 응답 시간을 줄이도록 한다.

3.4 주요 컴포넌트



[그림 1] 시스템 아키텍쳐

[그림 1]과 같이 본 시스템을 구성하는 주요 컴포넌트로 피어투피어 세션 관리자(P2P Session Manager), 노드 관리자(Node Manager), 조정자(Coordinator), 데이터사전(Dictionary), 메모리상주 DBMS(Main-memory DBMS)로 나눌 수 있다. 피어투피어 세션 관리자는 각 노드간의 데이터와 로그정보, 제어 메시지 전송을 위한 피어채널(Peer Channel)과 클라이언트의 직접적인 질의를 처리하는 클라이언트 채널로 구성된다. 노드관리자는 각 노드의 시스템 정보, 질의 패턴, 접근되는 데이터에 대한 통계정보를 관리함으로써 자신의 노드에 걸리는 부하를 체크하고 온라인 확장의 실행 여부를 결정한다. 조정자는 두개 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 DBMS를 통해 처리되며 빠른 데이터 접근을 위한 메모리 기반 색인으로 CSB$^+$트리가 내부적으로 구성된다.

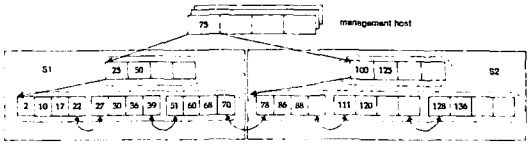
4. 온라인 확장을 고려한 CSB$^+$ 트리 재구성 기법

4.1 재구성 예

[그림 2]에서 서버 S1에는 S2보다 많은 데이터가 집중되어 있다. 따라서 S1은 S2 보다 일반적인 질의 요청이 많아 질 것이다. 이 때 S1에 저장된 데이터의 일부를 인접 서버나 새로운 노드에 이동시킴으로써 작업량의 집중으로 인한 성능의 저하를 막을 수 있다. [그림 2]와 같은 구성에서는 사용자의 데이터에 대한 질의 요청 패턴이 보편적인 경우 인접 서버로 데이터를 이동하여 문제점을 해결 할 수 있다.

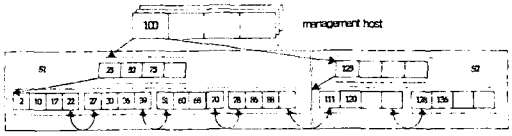
[그림 2] 시스템의 초기 기본 구성

[그림 3]은 S1의 시스템 부하를 막기 위해 각 노드를 재구성 한 것이다.

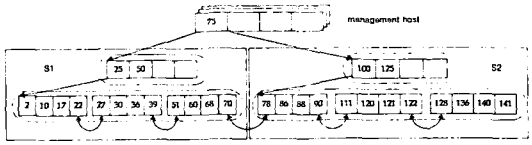


[그림 3] [그림 2]를 재구성한 구조

그러나 이 경우에 만일 다수의 사용자가 0~75 범위의 정수값을 검색하

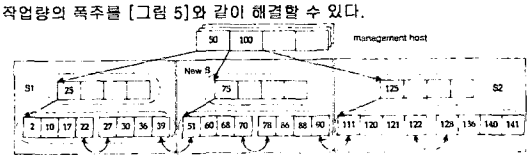


고자 하는 질의를 계속적으로 요청할 경우 S1에 그 범위의 데이터들이 대부분 존재하므로 다시 S1에 작업량이 집중하는 현상이 발생하고 또 다시 서버 간의 데이터 재배치를 필요로 하게 된다.



[그림 4] 인접 서버의 작업량이 비슷한 경우의 구조

위의 방법의 데이터 재구성은 모두 인접 서버에 데이터를 이동 시킴으로써 시스템의 부하를 해결하였다. 그러나 [그림 4]와 같이 인접 서버에 저장된 데이터의 양이 비슷한 경우, 인접 서버들간의 데이터 재배치는 무의미하다. 또한 온라인 상에서의 증권거래와 같이 빠른 응답시간이 요구되거나 여러 대의 인접한 서버에 걸쳐 저장된 데이터에 동시에 사용자의 접속이 폭주한다면 인접 서버간의 재배치 만으로는 서버들 간의 반복적인 데이터 재배치가 발생할 수 있고 그에 따른 서버 내의 재배치로 인한 응답시간의 지연을 가져올 수 있다. 이를 해결하기 위해서는 하나의 서버에 데이터를 분배하는 것이 아니라 온라인 상에서의 시스템 확장으로 작업량의 폭주를 [그림 5]와 같이 해결할 수 있다.



[그림 5] 온라인 상에서 확장된 서버로의 데이터 이동

4.2 재구성 정책

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

1단계 - 데이터 이동의 시작
2단계 - 이동할 데이터의 양과 목적지 노드의 결정
3단계 - 이동된 데이터와 목적지 노드의 데이터 통합

이때, 2단계는 다시 두 가지 경우로 나누어 볼 수 있다. 첫째는, 기존의 가동중인 노드(Active Node)들 안에서 색인을 재구성하는 경우이고 둘째는, 온라인 확장이 결정되어 예비노드(Spare Node) 상태에서 클러스터에 참여한 새로운 노드(New Node)들을 포함해서 각 노드의 색인을 재구성하는 경우이다. 이미 서론에서 언급한 대로 전자의 경우는 한번에 두개 이하의 노드에 작업량이 집중된 문제는 해결할 수 있지만 세 개 이상의 노드에 한꺼번에 부하가 집중된 경우는 앞 절의 예처럼 인접노드로의 반복적인 데이터의 이동만 발생되고 근본적인 부하를 제거하지 못하는 한계점을 갖는다. 후자의 경우, 세 개 이상의 노드들의 작업량이 특정 한계값을 초과한 경우 온라인 확장을 수행하여 기존의 노드와는 별도의 새로운 노드에 이동할 데이터의 양을 결정한다

4.3 확장된 CSB$^+$ tree

4.1에서 언급한 것처럼 온라인 상에서의 데이터 재배치는 각 서버에서의 인덱스 재구성에 따른 오버헤드를 줄이는 것이 중요하다. 이 때 각 노드간의 tree의 높이차이가 클 경우 재구성의 비용은 증가하게 된다. 이를 해결하기 위해 본 시스템은 기존의 adaptive B$^+$ tree를 이용한다. 이는 *management host*의 저장 능력을 높여 트리의 높이가 높아지는 것을 억제시켜 재구성의 비용을 줄인다.

확장된 CSB B$^+$의 노드 구성은 다음과 같다.

nKey : 노드 내에 있는 키의 개수
firstChile : 자식 노드 그룹에 있는 첫번째 자식노드의 주소값

keyList[2d] : key 값

명시적인 자식 노드의 포인터가 하나 뿐이므로 나머지 자식 노드들을 탐색하기 위해서는 옵셋값을 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.이는 온라인 상에서의 쇼핑몰이나 전자도서관 같이 빠른 탐색 시간이 요구되는 환경에서는 효율적이지만 저장공간의 제약이 따를 경우에는 각 노드 그룹을 세그먼트 단위로 나누어 저장하는 방법을 이용하는 것이 더 효율적이다.

4.3 알고리즘

```
begin
    /*S_NUM : 작업량이 집중된 서버의 개수 */
    if( S_NUM != 1){
        /*이들 서버가 인접한 곳에 위치 해 있는가?*/
        if( yes ) {
            nearS_NUM = 인접 서버의 개수;
            if( nearS_NUM == 짝수 )  newS_NUM = nearS_NUM / 2;
            else newS_NUM = nearS_NUM - 1;
            데이터 재분배;
        }
        else
            각 서버에 대해 S_NUM = 1인 경우와 동일하게 처리;
    }
    else {
        /*S: 작업량이 집중된 서버, S_MOVE : 데이터를 이동시킬 서버*/
        if( S == 노드 그룹의 첫번째에 위치 ) {
            S_MOVE = S_RIGHT;
            if( S.데이터 양 ≒ S_MOVE 데이터 양 ) 새로운 노드 추가;
        }
        else if( S == 노드 그룹의 맨 마지막에 위치 ) {
            S_MOVE = S_LEFT;
            if( S.데이터 양 ≒ S_MOVE 데이터 양 ) 새로운 노드 추가;
        }
        else {
            if( S_RIGHT.데이터 양 > S_LEFT.데이터 양 ) S_MOVE = S_LEFT;
            else S_MOVE = S_LEFT;
            if( S.데이터 양 ≒ S_MOVE.데이터 양 ) 새로운 노드 추가;
        }
    }
end
```

[알고리즘 1] 데이터 재분배 알고리즘

[알고리즘 1]을 살펴보면 먼저 작업량이 폭주한 서버의 개수를 검사한다. 개수가 짝수일 경우 (서버 개수 / 2)개, 홀수일 경우 (서버 개수 - 1)개의 시스템을 온라인 상에서 확장시켜 데이터를 재분배한다. 반면 작업량이 한 대의 서버에 집중되었을 경우에는 인접 서버의 데이터 양을 조사하고 데이터 차이가 적을 경우 시스템 한 대를 추가하여 새로운 노드로 데이터를 이동시키고 그렇지 않을 때에는 인접 서버 중 데이터의 양이 적은 쪽을 선택해 데이터를 이동한다.

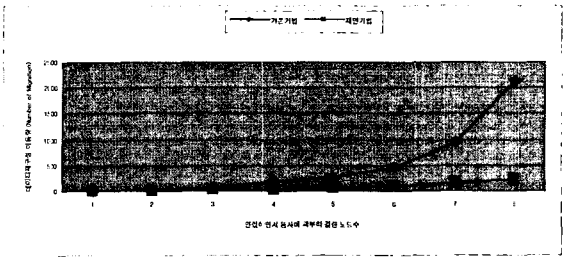
```
/*인접 서버간 작업량의 많은 차이가 있는 경우*/
begin
    /*S : 데이터의 이동을 필요로 하는 서버*/
    if( S_RIGHT.작업량 < S_LEFT.작업량 ) {
        /*addData : 이동시킬 데이터가 있는 노드의 최상위 노드 주소*/
        if( nKey == 1 ) addData = S.root -> firstChild;
        else addData = S.root -> firstChild + S.root-> nKey * offset;
        transfer( addData ); //주소값으로 노드에 접근한 후 데이터 이동
        S.root->firstChild->nKey =  S.root->firstChild->nKey - 1;
    }
    else {
        addData = S.root->firstChild
        transfer(addData); //주소값으로 노드에 접근한 후 데이터 이동
        new node = mememory allocate( 2 * order * offset + sizeof(nKey) + sizeof(firsChild) )
        new node = S.root->firstChild를 제외한 모든 노드들
        S.root->firstChild->nKey =  S.root->firstChild->nKey - 1
        S.root->firstChild = new node
    }
end
```

[알고리즘 2] 인접 서버로 데이터 이동 시 트리 재구성 알고리즘

[알고리즘 2]에서는 CSB$^+$ 트리 재구성을 보여주기 위한 것이다. 트리 내부에서 이동시킬 데이터의 양이 결정되었다고 가정하면 작업량이 특정 서버에 집중될 경우 데이터를 오른쪽 서버로 이동할 것인지 왼쪽으로 이동할 것인지에 대한 판단을 한다. 이 때 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 재저장해야 하고 부모 노드의 firstChild 값을 변경해 주어야 한다. 여기서는 작업량이 한곳에 집중되었을 경우만을 보여주고 있으나 여러 대의 서버의 데이터에서 재배치가 필요한 경우에도 기본적인 알고리즘은 비슷하다.

5. 성능 평가

본 장에서는 제안된 기법의 성능 평가를 측정하기 위해서 16개의 활동노드와 8개의 가용 노드를 갖고 동시 과부하 노드의 수를 1 ~ 8개의 경우에 데이터 재구성 횟수를 비교평가 하였다. 실험 결과 [그림 6]과 같이 제안된 기법이 기존 기법에 비해 적은 횟수의 데이터 이동만으로 과부하 문제를 해결하는 것을 볼 수 있다.



[그림 6] 동시 과부하 노드 수에 대한 데이터 재구성 횟수비교

6. 결론

본 연구는 기존의 온라인 스케일링 시스템의 문제점 보완과 향상된 성능에 초점을 맞추었다. 기존 시스템에서의 문제점들은 본 논문에서 제시하는 클러스터링 데이터베이스에서 온라인 확장을 고려한 CSB$^+$ 트리 색인의 온라인 재구성 기법이 해결하고 있다.

확장된 CSB$^+$ 트리는 온라인 환경에서 요구되는 빠른 응답 시간을 만족시켜 줄 수 있고 전체적 구조로 보았을 때 제안된 재구성 기법은 특정 서버에 작업량이 집중되어 발생하는 성능저하를 막기 위해 데이터를 재분배 가능하도록 한다. 이때 두 개 이상의 서버에 질의가 집중되는 경우 유휴 서버에 데이터를 분배시키는 방법으로 온라인 확장을 사용할 수 있는데 이 방법은 기존 서버에 데이터를 분배 했을 경우보다 내부 트리의 구성시간이 단축되고 데이터의 이동 횟수를 감소시킨다.

7. 참고문헌

[1] Kiran J. Achyutuni. " Two Techniques for On-Line Index Modification in Shared Nothing Parallel Databases" , In Proceedings SIGMOD ' 96 6/96 Montreal, Canada

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

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

[4] C.Zou, B.Salzberg. " Safely and Efficiently Updating References during Online Reorganization" , In Proceedings VLDB, 1998

[5] Jun Rao, Kenneth A.Ross. " Making B+-Trees Cache Conscious in Main Memory" , In MOD 2000, Dallas, TX USA

[6] Uwe Röhm, Klemens Böhm, Hans-Jörg Schek. " Cache-Aware Query Routing in a Cluster of Databases" , In Proceedings 17th ICDE, 2001

[7] Mong Li Lee, Masaru Kitsuregawa. " Towards Self-Tuning Data Placement in Parallel Database Systems" , In SIGMOD ' 2000 Dallas, Texas USA