단일 무선 채널에서 브로드캐스트 디스크 프로그램을 위한 지수 인덱스 기법

(An Exponential Indexing Scheme for Broadcast Disk Program in a Single Wireless Channel)

박 기 영 † 정 성 원 ††
(Kiyoung Park) (Sungwon Jung)

요 약 모바일 환경에서는 상향링크와 하향링크의 대역폭이 비대칭적이며 전력이 한정되어 있기 때문에 보다 효율적인 데이타 전송기술인 브로드캐스팅 방법이 연구되어 왔다. 브로드캐스트에서 인덱스를 사용하면 원하는 데이타가 언제 방송되는지를 알 수 있어 튜닝 시간을 줄이고, 전력의 소비를 줄일 수 있다. 지금까지 연구된 싱글 채널 인덱스 기법들은 모든 데이타 아이템이 동일한 확률로 접근되는 flat 브로드캐스트에 적합한 인덱스 기법들이다. 데이타 아이템에 대한 접근 확률이 편향되는 경우에는 멀티디스크 방송 기법을 사용해야 효과적인 전송이 가능하나, 기존의 인덱스 기법들은 인덱스가 한 방송 주기 내에서 반복되어 방송되는 데이타 아이템을 가리킬 수 없기 때문에 멀티디스크 방송 기법에는 효과적이지 않다. 본 논문에서는 싱글 채널 인덱스 기법으로서 멀티디스크 방송에 적용되는 인덱스 기법인 MDEI (Multi-disk Exponential Index) 기법을 제안한다. 본 논문에서 제안 하는 MDEI 기법은 각 디스크 별로 인덱스를 구성하기 때문에, 데이타에 대한 접근확률이 편향되는 경우에 flat 브로드캐스트를 사용하는 다른 인덱스 기법들보다 평균 접근 지연시간 시간을 크게 줄일 수 있다. 실험 결과는 데이타에 대한 접근 확률이 편향된 환경에서 MDEI가 평균 접근 지연시간에 있어서 매우 좋은 성능을 갖는 것을 보여준다.

키워드 : 모바일 데이타베이스, 무선 방송, 브로드캐스트 디스크, 지수 인덱스

Abstract Broadcast scheme has been widely researched for efficient data delivery in the mobile environment because the downlink capacity of a mobile client is much greater than the uplink capacity, and the power of a mobile client is limited. In the proposed scheme, the index lets the client know when data items would be broadcasted and enables the client to minimize the tuning time and power consumption. Single channel index schemes are fit to flat broadcast that performs well when all the broadcasted data items are accessed with the same probability whereas the multi-disk broadcast scheme is proper when the data access distribution is skewed. The existing index schemes, however, cannot work on the Multi-disk broadcast scheme because they cannot point the replicating data items in a broadcast cycle. This paper proposes a Multi-disk Exponential Index (MDEI) which is a single channel index scheme fit to Multi-disk broadcast scheme. Because MDEI scheme organizes a separate index for each disk, it functions with multi-disk broadcast, resulting in a greater reduction of average access latency than that of other flat-broadcast index schemes when the data access distribution is skewed. The performance evaluation showed that MDEI has a good performance when data access distribution is skewed. MDEI has short average access latency and not much average tuning time when the data access distribution is skewed.

Key words : Mobile Databases, Wireless Broadcast, Broadcast Disk, Exponential Index

· 본 연구는 국토해양부 첨단도시기술개발사업-지능형국토정보기술혁신 사업과제의 연구비지원(07국토정보C03)에 의해 수행되었습니다.

† 학생회원 : 서강대학교 컴퓨터공학과
joy710@sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
jungsung@sogang.ac.kr

논문접수 : 2008년 2월 27일
심사완료 : 2008년 8월 21일

정보과학회논문지: 데이타베이스 제35권 제6호(2008.12)

1. 서 론

빠르게 발전하고 있는 컴퓨터 하드웨어와 무선 네트워크 기술은 모바일 컴퓨팅 분야의 발전을 가속화 하고 있으며 이에 따라 모바일 기기의 사용이 급증하고 있다. 모바일 환경에서의 낮은 대역폭과 이동 단말기들이 가지게 되는 휴대성으로 인한 전력 소모 문제 때문에 모바일 유저들에게 어떻게 데이타를 효과적으로 전달할 수 있는가의 문제가 대두된다[1,2].

이 문제를 해결하기 위한 연구 중 하나로 다수의 클라이언트들이 필요로 하는 데이타를 서버에서 전송해 주는 방법인 브로드 캐스트가 연구되어 왔다. 이 방법은 서버가 계속하여 데이타들을 브로드캐스트 하고, 클라이언트는 필요로 하는 데이타를 그 중에서 선택적으로 받는 방법이다. 이러한 브로드캐스트의 성능은 클라이언트가 원하는 데이타를 얻기 위해 채널을 청취(listen)하는 시간인 튜닝 시간(tuning time)과, 클라이언트가 데이타에 대한 요청을 보낸 후부터 그 데이타를 얻을 때까지 걸린 시간인 접근 지연시간(access latency)으로 평가할 수 있다. 튜닝 시간 동안 클라이언트는 활성 모드(Active mode)로 동작해야 하기 때문에 이 시간은 단말기의 전력 소모량과 비례하며, 접근 지연시간은 클라이언트에게 제공되는 브로드캐스트 서비스의 질(Quality of Service)을 나타내는 척도가 된다.

브로드캐스팅에서 인덱스를 사용하면 원하는 데이타가 언제 방송되는지를 알 수 있어 튜닝 시간을 줄이고, 전력의 소비를 줄이는 효과가 있다. 이에 따라 브로드캐스팅에 대한 다양한 인덱스 방법들이 개발되어 왔다. Lee는 시그니처를 사용하는 인덱스 기법을 제안했다[3]. 이 기법은 방송 주기를 몇 개의 프레임으로 나누고, 각각의 프레임 앞에 프레임 내의 데이타 아이템들에 대한 시그니처를 더하여 방송한다. 클라이언트는 시그니처를 확인해봄으로써 원하는 데이타가 현재 프레임 내에서 방송되는지를 알아볼 수 있다. 하지만 시그니처는 데이타들이 프레임 내에서 언제 방송되는지에 대한 정보를 담고 있지 않기 때문에 시그니처를 통해 원하는 데이타를 찾더라도 해당 프레임을 순차적으로 모두 읽어보아야 한다. 그리고 시그니처가 일치하여 해당 프레임을 모두 읽더라도 원하는 데이타가 프레임 내에 없을 수도 있다. 그 뿐만 아니라 시그니처는 방송 주기 전체에 대한 전역적인 정보를 가지고 있지 않기 때문에 시그니처들을 순차적으로 모두 읽어야 한다. Imielinski는 해시 함수를 사용할 것을 제안하였다[4]. 이 기법에서는 해시 함수를 사용하여 데이타 아이템을 슬롯에 할당한다. 해시 값의 충돌이 일어날 경우에는 충돌된 데이타 아이템을 한 슬롯 뒤에 할당하고 이후의 데이타들을 한 슬롯씩 이동시킨다. 해시 함수는 브로드캐스트 프로그램을 통해서 방송되므로 클라이언트는 이 함수를 통해 원하는 데이타 아이템이 어떤 슬롯에서 방송되는지 알 수 있다.

트리를 이용한 기법도 여러 가지로 제안되었다. Imielinski는 또한 기존의 디스크 저장장치들을 위해 사용되던 트리를 기반 인덱스 기법을 무선 브로드캐스트 환경에 적용할 것을 제안했다[5]. 그것은 트리의 인덱스 노드들을 방송 스케줄의 데이타 아이템 사이에 함께 방송하는 것이다. 이 기법을 이용하면 클라이언트는 방송 중에서 루트 인덱스 노드로부터 트리의 링크를 따라 원하는 데이타를 가리키는 인덱스 노드에 선택적으로 접근할 수 있다. Shivakumar는 데이타의 접근 확률이 상이할 경우에는 Alphabetic Huffman Tree를 사용하면 평균 튜닝 시간을 더 줄일 수 있다는 것을 보였다[6]. 이 기법에서는 접근 빈도가 높은 데이타 아이템을 인덱스 트리의 루트에 가깝게 위치시킴으로써 인덱스 노드에 대한 접근 횟수를 줄이는 방법을 사용하였다. Xu는 트리 기반의 인덱스 기법을 확장하여 링크를 공유하는 여러 개의 인덱스 트리를 구성하도록 했다[7][8]. 지수 인덱스라고 불리는 이 인덱스는 인덱스가 위치한 버켓을 기준으로 인덱스가 구성되기 때문에 어떤 인덱스를 읽더라도 그 인덱스가 루트 인덱스가 된다. 그렇기 때문에 이 기법을 사용하면 방송 주기 내의 어떤 위치에서든지 바로 검색을 시작할 수 있다.

위의 기법들은 모든 데이타 아이템이 동일한 빈도로 방송되는 flat 브로드캐스트 방송에 적합한 인덱스 기법들이다. 데이타 아이템에 대한 접근 확률이 편향되는 경우에는 접근 확률이 높은 데이타 아이템을 더 자주 방송하는 멀티디스크 방송[9]을 해야 효과적이다. 하지만 멀티디스크 방송 기법에 적용하는 데 있어서 기존의 인덱스 기법에는 심각한 문제점이 있다. 왜냐하면 멀티디스크에서는 인기가 높은 데이타아이템이 여러 번 반복되어 방송되는데 반하여, 포인트 할 수 있는 위치가 하나뿐인 기존의 인덱스 기법을 사용해서는 반복되는 데이타 아이템의 위치를 가리킬 수 없기 때문이다.

MHash 기법[10]은 두 개의 인수를 갖는 해시 함수를 사용하여 반복되는 데이타 아이템의 위치를 가리킬 수 있게 하였고, 그 결과 데이타에 대한 접근 확률이 편향되는 경우에 flat 브로드캐스트 방송에 대한 인덱스 기법보다 좋은 성능을 갖는다. 그렇지만 MHash 기법은 두 가지 이유에서 멀티디스크 방송에 적합하지 않다. 첫째, 멀티디스크 방송 프로그램의 모든 데이타 아이템을 정확하게 가리킬 수 있는 해시 함수는 없다. 두 번째 이유는 MHash 기법이 데이타 아이템의 최대 반복 횟수를 일정 숫자 이하로 제한한다는 것이다. 이는 데이타

접근 확률이 편향되는 경우에 멀티디스크 방송 프로그램에서 요구하는 반복 횟수가 MHash 기법의 최대 반복 횟수보다 클 수 있다는 것을 의미한다.

본 논문에서는 멀티디스크 방송에 적합한 싱글 채널 인덱스 기법인 MDEI(Multi-disk Exponential Index)를 제안한다. MDEI는 멀티디스크에서의 동작을 위해 인덱스가 한 방송주기 내에서 여러 번 방송되는 데이타 아이템을 가리킬 수 있도록 지수 인덱스 기법을 확장한 기법이다. 기존의 지수인덱스는 하나의 키에 대해 하나의 위치만을 지정할 수 있기 때문에 멀티디스크에서의 동작이 불가능하지만, MDEI는 이를 멀티디스크의 각 디스크마다 별도의 인덱스를 구성하도록 확장했기 때문에 멀티디스크에서의 동작이 가능하다. 그러므로 MDEI 기법은 데이타 아이템 접근 확률이 편향될 때 멀티디스크 기법과 함께 사용하면 평균 접근 지연시간을 줄일 수 있다.

특별히 지수인덱스를 사용한 것은 멀티디스크에의 확장에서 몇 가지 장점을 갖기 때문이다. 본 논문에서처럼 디스크별로 인덱스를 구성할 경우에는 디스크 내의 데이타 아이템 개수가 인덱스의 크기에 영향을 준다. 그런데 멀티디스크에서 각각의 디스크가 포함하는 데이타 아이템의 개수는 데이타에 대한 접근 확률의 편향성 정도에 따라 달라진다. 트리 기반의 인덱스 기법들은 인덱스 트리가 하나의 루트를 갖는 트리로 구성되기 때문에 데이타 아이템의 개수 변화가 인덱스 트리의 크기 변화에 큰 영향을 주지만, 지수인덱스는 서로 다른 루트를 갖는 인덱스 트리들이 방송 전체에 분산되어 나타나기 때문에 데이타 아이템의 개수 변화가 인덱스 테이블의 크기 변화에 미치는 영향이 작아서 멀티디스크에의 확장에 적당하다. 또한 지수인덱스는 인덱스 테이블이 짧은 간격으로 자주 방송되기 때문에 상대적으로 초기 탐색 시간이 짧아서 평균 접근 지연시간을 줄일 수 있다는 장점이 있다.

다만 MDEI는 지수인덱스를 확장한 것으로, 지수인덱스 기법보다 인덱스 버킷의 사이즈가 커지게 되어 실제 데이타와 비교하여 과도한 인덱스의 사이즈 크기의 우려가 생길 수 있다. 하지만, 기존의 지수인덱스 기법과 비교하여 추가적으로 수 개의 컨트롤 정보만이 추가될 뿐이고, 이는 각각 수 비트의 크기만으로도 충분한 인덱싱이 가능하므로 실제 전송되는 데이타의 사이즈가 수십 바이트 이상임을 감안하면 줄어드는 평균 접근 지연시간에 비해 영향을 덜 받음을 알 수 있다. 추후 본 논문의 4장에서 실험을 통하여 이를 증명할 것이다.

마지막으로 본 논문에서는 clustered 브로드캐스트를 가정한다. clustered 브로드캐스트는 똑같은 키값을 갖는 반복적인 데이타들이 연속적으로 나타남을 가정하며 따라서 반복되는 첫 데이타만 찾으면 연속되는 모든 데이타를 받을 수 있다. 반대로 non-clustered 브로드캐스트는 똑같은 킷값을 갖는 반복적인 데이타들이 비연속적으로 나타남을 가정하며 따라서 하나의 데이타를 찾았다고 할지라도 나머지 데이타를 찾기 위한 추가적인 검색이 필요하다. [5,7]에서는 non-clustered 브로드캐스트 환경에서의 인덱싱 기법을 설명하고 있으나, 여전히 접근 확률의 편향성을 반영하지 못한다. MDEI의 경우 별다른 수정 없이 non-clustered 브로드캐스트에 적용이 가능하므로 non-clustered 브로드캐스트 환경에서도 접근 확률의 편향성을 반영할 수 있다. 본 논문에서는 기본적으로 clustered 브로드캐스트를 가정한 MDEI기법을 제시한 후에 non-clustered 브로드캐스트에 적용하는 방법에 대해 설명할 것이다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련된 연구를 살펴보고, 3장에서는 MDEI 브로드캐스트 프로그램의 구성 방법과 이에 대한 클라이언트의 접근 프로토콜을 설명할 것이다. 4장에서는 이 기법의 실험 환경과 실험 결과를 설명하고, 5장에서 이 논문의 결론을 맺을 것이다.

2. 관련 연구

앞에서 설명한 것처럼 무선 브로드캐스트 환경에서 인덱스를 사용하면 튜닝 시간을 줄일 수 있어 전력을 효과적으로 사용할 수 있게 된다. 이 장에서는 가장 최근의 연구 결과인 지수 인덱스와 MHash 기법에 대해 자세히 설명한다.

그림 1은 16개의 데이타 아이템으로 브로드캐스트 프로그램과 이에 대한 지수 인덱스를 구성한 예이다. 브로드캐스트 프로그램은 키의 오름차순으로 정렬되어 방송된다. 각각의 버켓은 테이블 형태의 인덱스를 갖는데, {*distInt*, *maxKey*}의 튜플로 이루어져 있다. *distInt*는 현재 버켓으로부터의 거리이며, *maxKey*는 *distInt*만큼 떨어진 버켓의 가장 큰 키 값이다. *i*번째 행의 *distInt*의 값은 2^i-1이다. 그러므로 버켓 1의 인덱스 테이블에서 *distInt*는 1, 2, 3, 4행에서 각각 1, 3, 7, 15의 값을 갖고, maxKey값은 IBM, MOT, SUNW, YHOO가 된다.

클라이언트는 이 인덱스 테이블을 따라 원하는 데이타 아이템을 찾는다. 데이타 아이템 NOK에 접근하고자 하는 클라이언트가 버켓 1에서 초기 탐색을 하여 버켓 1의 끝에서 방송되는 인덱스 테이블을 읽었다고 가정하자. 버켓 1의 인덱스 테이블에서 NOK는 세 번째 버켓인 MOT와 일곱 번째 버켓인 SUNW 사이에 위치할 것임을 알 수 있으므로 클라이언트는 네 번째 버켓인 버켓 5까지 대기 모드로 진행한다. 버켓 5를 읽은 클라이언트는 인덱스 테이블에서 NOK이 바로 다음 버켓에

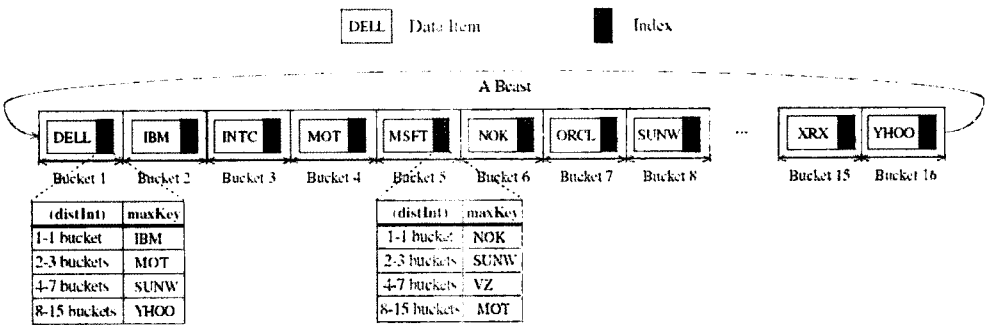

그림 1 Exponential Index

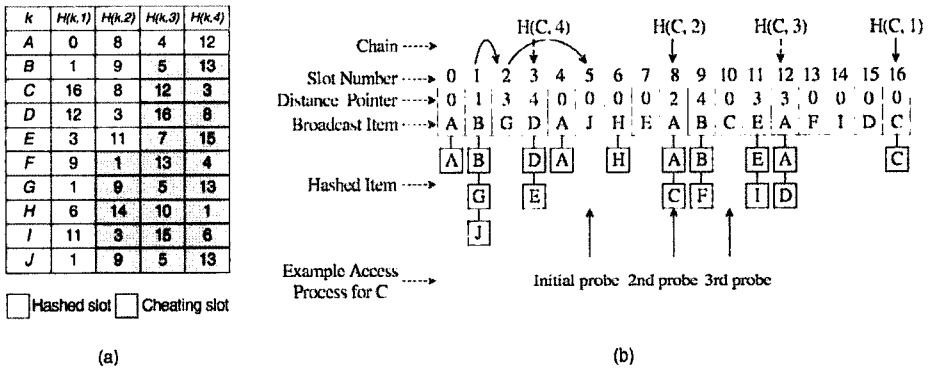

그림 2 MHash 기법 (a)해시 테이블 (b)방송 프로그램

서 방송됨을 알 수 있고, 다음 버켓에서 NOK에 접근한다. 이와 같은 인덱스 구조와 접근 프로토콜 때문에 지수인덱스에서는 방송 주기 내에서의 어떤 위치에서라도 탐색을 시작하는 것이 가능하며 평균 접근 지연시간도 짧고 평균 튜닝 시간도 길지 않다. 그렇지만 지수 인덱스는 1장에서 설명한 것처럼 flat 브로드캐스트를 기반으로 하는 인덱스 기법이기 때문에 멀티디스크 방송에는 사용할 수 없다.

MHash 기법은 non flat 브로드캐스트에 대한 인덱스 기법이다[10]. MHash 기법은 기본적으로 [4]와 같은 해시 기반의 기법이지만 [4]의 기법과는 달리 두 개의 인수를 갖는 해시 함수를 사용하여 non-flat 방송 프로그램에 적합한 기법으로 제안되었다. 그림 2의 테이블에서 볼 수 있듯이, 두 개의 인수를 갖는 해시 함수를 사용하면 하나의 데이타 아이템 키가 여러 개의 해시 값을 가질 수 있게 되어, 한 방송 주기 내에서 여러 번 방송되는 데이타 아이템을 가리킬 수 있게 된다.

MHash는 non-flat 브로드캐스트를 기반으로 동작하기 때문에 데이타 아이템에 대한 접근 확률이 편향되는 경우 flat 브로드캐스트를 기반으로 하는 기존의 인덱스 기법들보다 접근 지연시간을 많이 줄일 수 있다. 하지만 역시 멀티디스크에 적용할 수는 없는 기법이다. 3장에서는 멀티디스크를 기반으로 동작하는 새로운 인덱스 기법을 제안한다.

3. Multi-disk Exponential Index

이 장에서는 Multi-disk Exponential Index (MDEI)의 구성 방법을 설명하고, MDEI 인덱스를 이용한 방송 프로그램에 클라이언트가 접근하는 프로토콜을 설명한다.

3.1 MDEI 인덱스 구성

MDEI는 멀티디스크 방송기법[9]에 대한 인덱스 기법이므로 방송 프로그램은 멀티디스크로 구성한다. 여기서 각각의 버켓은 하나 이상의 데이타 아이템과 컨트롤 정보, 그리고 인덱스 테이블을 갖는다. 본 장에서는 간단한 논의를 위해 하나의 버켓은 컨트롤 정보와 인덱스 테이블과 더불어 단 하나의 데이타 아이템만 갖는다고 가정한다.

멀티디스크를 구성하기 위해서는 주어진 데이타 집합을 주어진 개수의 디스크로 나눈다. 주어진 디스크의 개수가 *numDsk*라고 하면 disk 1, disk 2, ···, disk *numDsk*가 존재하며, 낮은 번호의 디스크가 높은 번호의 디스크보다 더 큰 방송 빈도를 갖는다. 그러므로 상

대적으로 접근 확률이 높은 데이타 아이템들이 낮은 번호의 디스크에 포함되며, 접근 확률이 낮은 데이타 아이템들은 높은 번호의 디스크에 포함된다. 각각의 디스크들은 몇 개의 chunk로 나누어지고, 이 chunk들이 서로 인터리빙 되어 멀티디스크 방송 프로그램으로 구성된다. 이 때, 각각의 디스크는 데이타 아이템의 키 값에 의해 정렬시켜야 MDEI 인덱스를 구성할 수 있다.

예를 들어 그림 3의 데이타 아이템들을 멀티디스크로 구성하여 방송하는 경우를 생각해보자. 방송되는 데이타 아이템은 모두 40개이며, 그 중 8개의 데이타가 나머지 32개의 데이타보다 자주 접근된다. 자주 접근되는 데이타 아이템들을 나머지 데이타 아이템보다 두 배 더 자주 방송되도록 하여 멀티디스크를 구성하면, 그림 4의 방송 프로그램이 만들어진다. D33, D35 등의 인기 있는 8개 데이타 아이템이 포함된 disk 1은 한 방송 주기에 두 번 방송되며, 나머지 인기 없는 데이타 아이템이 포함된 disk 2는 한 방송 주기에 한 번 방송된다. Disk 1과 disk 2는 각각 데이타 키에 의해 정렬되어 있다.

이제 프로그램에 대한 인덱스를 구성해야 한다. 인덱스는 테이블 형태로 구성되며 각 버켓의 뒷부분에 저장된다. 인덱스 테이블의 엔트리 수는 디스크마다 다르다. 디스크 k에 데이타 아이템이 n_k개 있다고 할 때 디스크 k에서 방송되는 버켓의 인덱스 테이블의 엔트리 수는 $\lceil \log n_k \rceil$ 이다. 테이블은 {*distInt*, *maxKey*}의 튜플로 이루어진다. 여기서 *distInt*는 현재 버켓으로부터의 거리, *maxKey*는 *distInt*만큼 떨어진 버켓의 최대 키 값이다. i 번째 엔트리의 *distInt*값을 $distInt_i$라고 할 때, $distInt_i$는 2^i-1이다. 그러므로 임의의 버켓에서 $distInt_1$은 1, $distInt_2$는 3, $distInt_3$은 7이 된다.

MDEI는 멀티디스크에서 동작하기 위하여 인덱스 테이블을 디스크별로 따로 구성한다. 즉, 임의의 디스크에서 방송되는 인덱스 테이블은 그 디스크에 포함된 데이타만을 대상으로 구성된다. 그러므로 *distInt*를 가지고 *maxKey*를 구할 때 *distInt*의 거리는 현재 디스크만을 고려한다. 그림 4의 bucket 1에서 방송되는 인덱스 테이블을 예로 들어보자. bucket 1은 disk 1에 포함되고 disk 1에서는 8개의 데이타 아이템이 방송되므로 bucket 1의 인덱스 테이블의 엔트리 수는 3개가 된다. 세 개의 엔트리에서 *distInt*는 각각 1, 3, 7의 값을 갖는다. i번째 엔트리의 *maxKey*값을 $maxKey_i$라고 할 때,

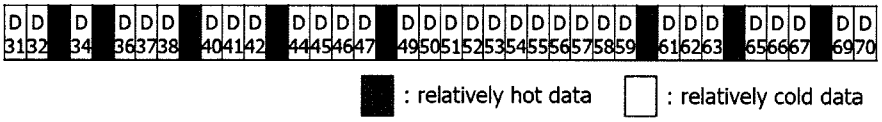

그림 3 방송할 데이타 아이템의 집합

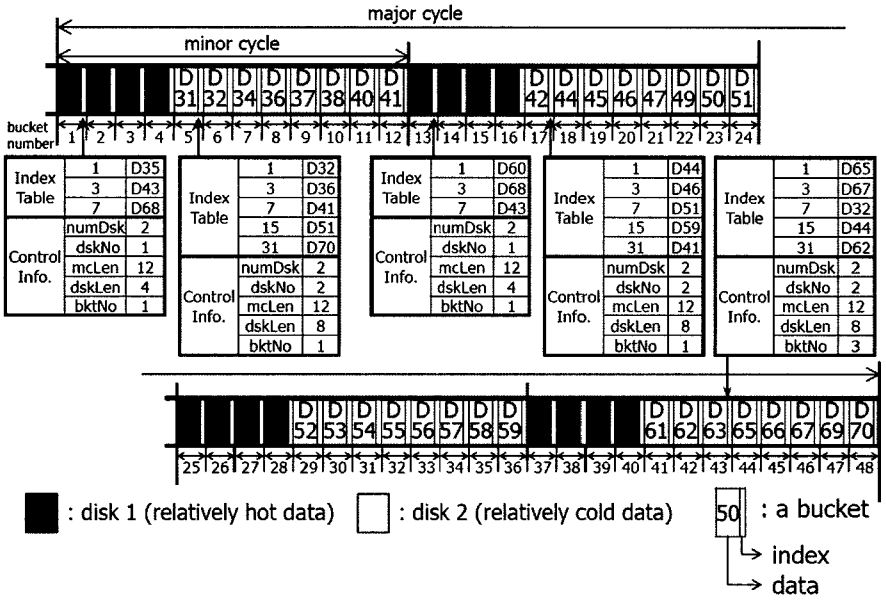

그림 4 MDEI 프로그램 구성

*distInt*가 1인 첫 번째 엔트리에서 $maxKey_1$은 bucket 2의 D35가 되며, *distInt*가 3인 두 번째 엔트리에서 $maxKey_2$는 bucket 4의 D43이 된다. *distInt*가 7인 세 번째 엔트리에서 $maxKey_3$은 bucket 8의 데이타 아이템 D36이 아니라 bucket 16의 D68이 된다. disk 1만을 생각하면 bucket 1로부터 7번째 버켓은 bucket 16이기 때문이다. 마찬가지로 bucket 5의 인덱스 테이블은 disk 2에 속해 있기 때문에 disk 2만을 고려하여 인덱스 테이블을 구성하고, 그 결과 네 번째 엔트리의 $maxKey_4$는 bucket 24의 D51, 다섯 번째 엔트리의 $maxKey_5$는 bucket 48의 D70이 된다.

여기서 한 가지 문제가 발생한다. 임의의 버켓에 있는 인덱스 테이블은 자신이 포함된 디스크만을 고려하여 구성되었기 때문에 인덱스 테이블만 가지고서는 여러 디스크들이 인터리빙되어 방송되는 프로그램 내에서 원하는 데이타의 방송 위치를 찾을 수 없다. 예를 들면, 데이타 아이템 D68을 찾는 클라이언트가 현재 bucket 1에 접근하여 인덱스 테이블을 읽었다고 가정할 때, 현재 위치와 D68이 방송되는 bucket 16 사이에 disk 2가 방송되기 때문에 D68의 방송 위치를 찾을 수 없다는 것이다. 그래서 각 버켓에서는 컨트롤 정보도 함께 방송된다. 컨트롤 정보는 방송 내의 디스크 개수를 나타내는 *numDsk*, 현재 버켓의 디스크 번호를 나타내는 *dskNo*, 마이너 주기의 길이를 나타내는 *mcLen*, 하나의 마이너 주기 내에서 현재 디스크가 차지하는 길이를 나타내는 *dskLen*, 현재 버켓의 상대적인 위치를 나타내는 *bktNo*, 현재 디스크의 그룹 크기를 나타내는 *gSize*로 구성된다. *bktNo*는 현재 마이너 주기 내에서 현재 디스크가 시작되는 버켓을 1번으로 하여 현재 버켓의 상대적인 위치를 의미한다. 표 1은 컨트롤 정보를 정리한 것이다.

bucket 43의 컨트롤 정보를 예로 들면, 현재 방송에서 디스크는 두 개가 있으므로 *numDsk*는 2이며, bucket 43은 disk 2에 포함되어 있으므로 *dskNo*는 2가 된다. 현재 방송에서 하나의 마이너 주기는 bucket 37에서 bucket 48까지의 12개 버켓으로 되어 있으므로 mcLen은 12이고, 그 중 disk 2는 8개 버켓으로 이루어져 있으므로 dskLen은 8이 된다. 현재 마이너 주기에서 disk 2는 bucket 41에서 시작하므로 현재 버켓의 *bktNo*는 3이 된다. 그룹 크기를 나타내는 *gSize*는 그룹을 사용하는 4장에서 다시 설명하기로 한다.

표 1 컨트롤 정보

Notation	Description
numDsk	디스크의 개수
dskNo	현재 방송되는 디스크의 번호
mcLen	하나의 마이너 주기의 길이
dskLen	하나의 마이너 주기 내에서 현재 디스크의 길이
bktNo	현재 디스크에서 현재 버켓의 상대적인 위치
gSize	현재 디스크의 그룹 크기

이러한 인덱스 구성 방법은 기존의 지수 인덱스 기법에 비해 컨트롤 정보가 추가되어 인덱스 크기가 늘어남에 따라 인덱스의 비율이 상승하게 된다. 그림 4에서의 예제를 살펴보면 컨트롤 정보의 크기는 2 byte 정도가 된다. 컨트롤 정보는 인덱스 테이블 당 하나만 추가되므로 인덱스 엔트리 하나의 크기가 4 byte를 가정하면 디스크 1에서의 인덱스의 크기는 (인덱스 엔트리의 크기 x 인덱스 엔트리의 개수 + 컨트롤 정보의 크기)이고 실제로 계산해보면 4 byte x 3 + 2 byte = 14 byte 가 된다. (컨트롤 정보가 없을 때의 인덱스 크기는 12 byte.) 같은 방법으로 디스크 2에서의 인덱스의 크기는 4 byte x 5 + 2 byte = 22 byte가 된다. 따라서 인덱스의 크기는 각각 16%, 10% 정도 커짐을 알 수 있다. 다만 실제 데이타 버킷의 크기를 1 KB 라고 감안하면 데이타 대비 인덱스의 비율은 각각 0.2% 정도가 늘어날 뿐이어서 거의 변하지 않는다고 할 수 있다. 또한 데이타 및 인덱스가 N배 늘어난다고 하여도 컨트롤 정보의 크기는 $\log N$ 배 만큼만 증가하게 되므로, 실제 방송환경에서 인덱스 크기 증가로 인한 접근 지연 시간의 손해는 미미하다. 5장에서의 성능 비교를 통해 이를 확인해 볼 것이다.

또한 이와 같은 인덱스 구성 방법은 clustered 브로드캐스트를 가정한 것으로, 한 가지의 고려사항을 추가하면 non-clustered 브로드캐스트에 적용이 가능하다. [5,7]에서 사용하고 있는 메타 세그먼트를 보면, 각각의 메타 세그먼트 내에서는 기존의 인덱싱 기법이 유효하며 따라서 기존의 지수 인덱스를 적용하여 사용하고 있다. 따라서 본 MDEI의 경우에도 메타 세그먼트 내에서는 동일하게 적용이 가능할 것이다. 한편 본 논문의 MDEI는 각 디스크별로 마이너 주기를 유지하고 지수인덱스를 마이너 주기별로 적용하고 있다. 따라서 이 마이너 주기를 메타 세그먼트와 동일한 길이로 구성한다면, non-clustered 브로드캐스트 환경에서도 MDEI를 쉽게 적용할 수 있음을 알 수 있다. 다만 하나의 데이타를 발견한 이후에도 다음 마이너 주기에 대한 검색이 계속 이루어져야 할 것이다.

데이타 아이템 수가 N일 때 MDEI 인덱스 구성은 O(N) 시간에 가능하다. MDEI 인덱스를 구성하기 위해서는 인덱스 테이블과 컨트롤 정보를 구성해야 한다. 먼저 컨트롤 정보 구성은 별도의 계산이 필요하지 않고 정해진 값들과 버켓 번호만 넣어주면 되기 때문에 O(N) 시간에 구성이 가능하다. 인덱스 테이블도 역시 O(N) 시간에 구성이 가능하다. 인덱스 테이블이 디스크별로

따로 구성되기 때문에 하나의 디스크만 고려하여 *distInt* 값과 *maxKey*만을 계산하면 인덱스를 구성할 수 있기 때문이다. 인덱스 테이블의 구성과 컨트롤 정보 구성은 별도의 작업으로 이루어지므로 전체 MDEI 인덱스 구성의 시간복잡도는 O(N)이 된다.

이로써 우리는 MDEI 인덱스의 구성 방법을 모두 알아보았다. 3.2절에서는 클라이언트가 원하는 데이타를 찾기 위해 구성된 인덱스 테이블과 컨트롤 정보를 어떻게 사용하는지를 설명한다.

3.2 데이타 접근 프로토콜

클라이언트는 방송으로부터 인덱스 테이블과 컨트롤 정보를 읽어서 원하는 데이타 아이템이 방송되는 시점을 찾아야 한다. 이 절에서는 3.1절에서 구성한 브로드캐스트 프로그램에서 클라이언트가 데이타 아이템에 접근하는 방법을 설명한다.

멀티디스크 방송 프로그램을 기반으로 하는 MDEI 인덱스를 이용하기 위해 클라이언트는 데이타를 디스크별로 따로 찾아야 한다. 원하는 데이타가 어떤 디스크에서 방송될지 모르기 때문에 클라이언트는 모든 디스크를 찾아보아야 한다. 각 디스크를 순차적으로 탐색하면 원하는 데이타에 접근하는 시간이 길어지므로 각 디스크를 병렬적으로 탐색한다. 병렬적인 탐색을 위해 클라이언트는 각 디스크에서의 다음 튜닝 위치 정보를 유지해야 한다. 이 정보를 $tuningPos_i$ ($1 \le i \le numDsk$) 라고 한다. $tuningPos_i$는 disk i에서의 다음 튜닝 위치이며 처음에는 모두 -1로 초기화한다.

클라이언트는 처음 접근한 버켓에서 인덱스 테이블을 읽는다. 현재 접근한 버켓이 disk k라고 하면 클라이언트는 disk k에서 원하는 데이타 아이템이 방송될 위치를 계산하여 $tuningPos_k$에 저장한다. 현재 접근한 디스크에서 데이타 아이템이 방송될 위치를 찾아 저장한다. 그리고 disk $((k+1)\%numDsk)$에 대해 동일한 작업을 반복한다. disk $((k-1)\%numDsk)$까지 동일한 작업을 반복하고 나면 $tuningPos_i$는 모든 디스크에 대해 다음 튜닝 위치를 저장한 상태가 된다. 이후로는 $tuningPos_i$에서 가장 가까운 튜닝 위치를 찾아가서 탐색을 계속하게 된다.

이제 문제는 인덱스 테이블에서 데이타 아이템이 방송될 위치를 구하는 방법이다. 인덱스 테이블은 인터리빙 되어 있는 다른 디스크를 고려하지 않고 현재 디스크만을 고려하여 구성되었기 때문에 인덱스 테이블만 참조해서는 정확한 다음 튜닝 위치를 계산할 수 없다. 다음 튜닝 위치의 정확한 계산에 대한 설명을 위해 다음과 같이 *logical_offset*과 *physical_offset*을 정의한다.

정의 1. $maxKey_k < targetKey < maxKey_{k+1}$인 *targetKey*에 대해 *logical_offset*은 다음과 같이 정의한다.

$$logical\_offset = distInt_k + 1, \text{ such that} \quad (1)$$

정의 2. 임의의 *logical_offset*에 대해 *physical_offset*은 다음과 같이 정의된다.

$$physical\_offset = q*mcLen - bktNo + r + 1 \quad (2)$$

where

$$q = \lfloor (logical\_offset + bktNo - 1)/\text{dskLen} \rfloor \quad (3)$$

$$r = \begin{cases} logical\_offset & \text{if } q = 0 \\ \lfloor (logical\_offset + bktNo - 1)\%dskLen \rfloor & \text{if } q > 0 \end{cases} \quad (4)$$

q는 현재 위치에서 *logical_offset*만큼을 진행하기 위하여 몇 번의 마이너 주기를 지나가야 하는지를 의미한다. r은 q만큼의 마이너 주기를 진행한 후에 몇 개의 버켓을 더 지나가야 하는지를 의미한다. 그러므로 식 (2)의 의미는 q회 만큼의 마이너 주기를 진행하고 r개의 버켓을 더 진행하는 만큼이 *physical_offset*이라는 의미이다. *mcLen*, *bktNo*, *dskLen* 등은 모두 컨트롤 정보를 통해 알 수 있는 정보들이고, q와 r은 인덱스 테이블로부터 계산되는 *logical_offset*을 이용해 계산되므로 클라이언트는 방송에서 *physical_offset*의 계산에 필요한 모든 정보를 얻을 수 있다. 그림 5는 클라이언트가 인덱스와 컨트롤 정보를 통해 원하는 데이타에 접근하는 알고리즘이다. disk k의 데이타 아이템 수를 n_k라고 할 때, 루프를 반복하는 최대 횟수는 $numDsk*\log n_k$이다. 루프 안에서는 상수번의 비교와 연산이 이루어지므로 알고리즘 전체의 시간 복잡도는 $O(numDsk*\log n_k)$가 된다.

예를 들어 그림 4에서 버켓 1이 방송되기 직전에 클라이언트가 데이타 아이템 D44를 요청했다고 가정하자. 클라이언트는 bucket 1에서 방송되는 인덱스 테이블을 보게 된다. D44은 D43보다 크고 D68보다 작기 때문에 만약 D44가 disk 1에서 방송된다면 D44는 현재 위치에서 네 버켓에서 일곱 버켓 만큼 떨어진 곳 사이에 존재

```
Input: Key k
Output: location of data item k
Set tuningPos_i=∞ for all 1≤i≤numDsk
while(true) {
    read a bucket currently broadcasted
    if ( The bucket contains data item k ) {
        receive data item k; return;
    }
    else {
        if (The disk of the bucket does not have the data item k)
            tuningPos_dskNo = -1;
        else
            tuningPos_dskNo = q*mcLen - bktNo + r + 1;
    }
    if ( tuningPos_(dskNo+1)%numDsk == ∞)
        tuningPos_(dskNo+1)%numDsk = dskLen - bktNo + 1;
    if ( tuningPos_i== -1 for all 1≤i≤numDsk) {
        search failure; return;
    }
    Go into the doze mode and tune in after min_(1≤i≤numDsk) tuningPos_i
}
```

그림 5 데이타 접근 알고리즘

한다. 그러므로 disk 1에서는 네 버켓 만큼 뒤에서 튜닝해야 하며, 디스크 1만 고려했을 때 네 버켓 만큼 떨어진 곳은 bucket 13이다. 그러므로 disk 1에서의 다음 튜닝 위치는 bucket 13이라는 것을 저장해두고 절전 모드로 disk 2로 간다.

disk 2는 bucket 5에서부터 시작된다. 클라이언트는 bucket 5의 데이타 아이템을 확인한다. 찾고자 하는 데이타 D44가 bucket 5에서 발견되지 않기 때문에 클라이언트는 bucket 5의 인덱스 테이블을 확인한다. D44는 D41보다 크고 D51보다 작으므로 클라이언트는 8번째 버켓까지 절전 모드로 기다린다. 이 때 디스크 2의 8번째 버켓은 bucket 17이므로 다음 튜닝 위치는 bucket 17이며, 이것을 저장해둔다.

disk 1과 disk 2의 다음 튜닝 위치가 각각 bucket 13과 bucket 17이므로, 클라이언트는 둘 중 먼저 방송되는 bucket 13까지 절전 모드로 진행한다. bucket 13에서 D44가 발견되지 않고, 인덱스 테이블을 확인해봤을 때, D44가 존재할 버켓이 없으므로 disk 1에서는 D44가 방송되지 않음을 알 수 있다. 그러므로 disk 1에서는 더 이상 찾아볼 필요가 없다. disk 2의 다음 튜닝 위치인 bucket 17까지 절전 모드로 진행한다. bucket 17에서는 D44가 발견되지 않지만, 인덱스 테이블에서 D44를 발견할 수 있다. 따라서 클라이언트는 하나 뒤의 버켓에서 원하는 데이타 D44에 접근할 수 있다.

지금까지 MDEI 인덱스의 구성과 데이타에의 접근 프로토콜을 설명했다. 이 기법에서 인덱스를 모든 버켓마다 방송하지 않고 몇 개의 버켓을 하나의 그룹으로 묶어서 그룹마다 인덱스를 한 번씩 방송하면 전체 방송의 길이를 줄일 수 있어서 평균 접근 지연시간을 줄일 수 있다. 다음 장에서는 그룹 단위로 인덱스를 방송하는 기법을 제시한다.

4. 그룹을 사용한 MDEI 인덱스

이 장에서는 그룹을 사용한 MDEI 기법을 설명한다. 먼저 그룹을 사용한 MDEI 인덱스의 구성 방법과 데이타에의 접근 프로토콜을 설명하고, 그룹 크기를 결정하는 문제를 설명한다.

4.1 그룹을 사용한 MDEI 인덱스 구성

그룹을 사용하여 MDEI 인덱스를 구성하는 방법은 기본적으로 그룹을 사용하지 않는 경우와 동일하다. 다른 점은 몇 개의 버켓을 묶어서 하나의 그룹을 구성하며, 인덱스의 구성하는 기본 단위가 버켓 대신 그룹이 된다는 것이다. 먼저 그림 6의 예를 들어 설명한다.

그림 6은 그림 3의 데이타 아이템들을 가지고 그룹을 사용하여 MDEI 프로그램을 구성한 예이다. disk 1은 2개의 버켓을 하나의 그룹으로, disk 2는 4개의 버켓을 하나의 그룹으로 묶어서 인덱스를 구성했으며 이 정보는 컨트롤 정보에서 방송된다. 인덱스 모든 버켓에서 방

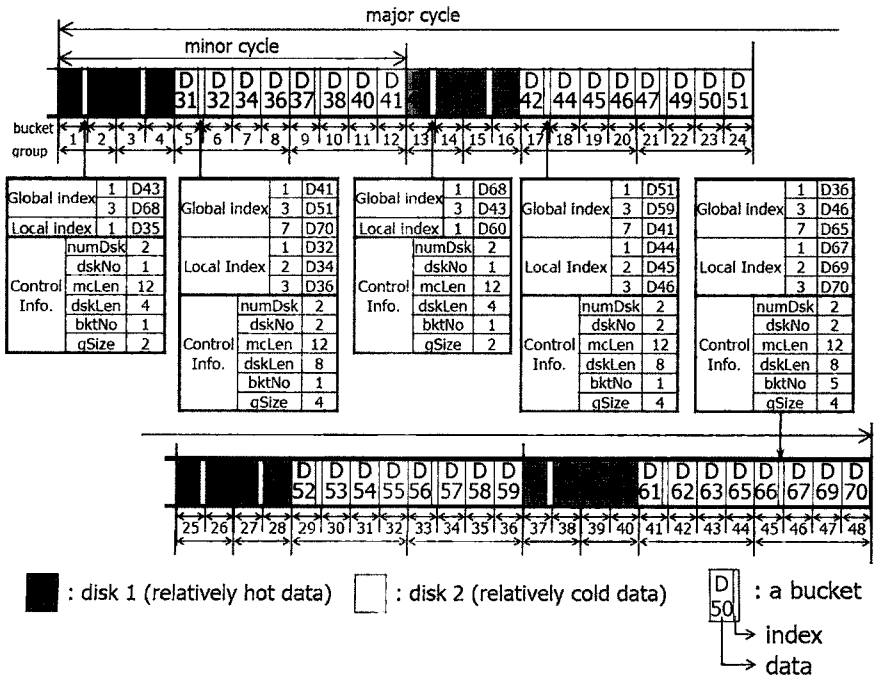

그림 6 그룹을 사용한 MDEI 프로그램 구성

송되는 대신 그룹의 첫 번째 버켓의 뒤쪽에서 방송된다. 그룹을 사용할 때는 인덱스가 전역 인덱스(global index)와 지역 인덱스(local index)의 두 가지 종류로 나뉘어 방송되는데, 전역 인덱스는 그룹 단위의 인덱스를, 지역 인덱스는 그룹 내에서의 버켓 단위의 인덱스를 의미한다. 예를 들어서 bucket 5에서 방송되는 인덱스 테이블에서 전역 인덱스는 세 개의 엔트리를 갖는데, 한 그룹 뒤의 *maxKey*가 D41, 세 그룹 뒤의 *maxKey*가 D51, 일곱 그룹 뒤의 *maxKey*가 D70임을 나타낸다. 지역 인덱스 역시 세 개의 엔트리를 가지며, 그룹 내에서 첫 번째 버켓의 *maxKey*는 D32, 두 번째 버켓의 *maxKey*는 D34, 세 번째 버켓의 *maxKey*는 D36임을 나타낸다. 전역 인덱스의 *distInt*는 그룹을 사용하지 않을 때의 인덱스와 같이 지수 함수의 속도로 증가하여 $distInt_i$는 2^i-1의 값을 갖지만, 지역 인덱스의 *distInt*는 선형으로 증가하여 $distInt_i$는 i의 값을 갖는다.

클라이언트는 데이타 아이템에 접근하기 위해 지역 인덱스와 전역 인덱스, 두 개의 테이블을 살펴보아야 한다. 클라이언트는 인덱스 테이블을 읽으면 먼저 지역 인덱스를 살펴본다. 지역 인덱스를 살펴보면 원하는 데이타 아이템이 현재 그룹 내에 있는지를 알 수 있기 때문이다. 원하는 데이타 아이템이 현재 그룹 내에 있으면 해당 위치를 다음 튜닝 위치로 저장해두고 대기 모드로 기다리고, 그렇지 않다면 전역 인덱스를 살펴본다. 전역 인덱스를 살펴볼 경우에는 그룹을 사용하지 않을 때와 같은 방법으로 데이타 아이템을 찾아가게 된다. 단, physical_offset을 계산함에 있어서 q와 r 값의 계산 방법이 식 (5)와 식 (6)과 같이 그룹 크기를 고려하도록 바뀌게 된다.

$$q = \lfloor (logical\_offset * gSize + bktNo - 1)/\mathrm{dskLen} \rfloor \quad (5)$$

$$r = \begin{cases} logical\_offset & \text{if } q = 0 \\ \lfloor (logical\_offset * gSize + bktNo - 1)\%dskLen \rfloor & \text{if } q > 0 \end{cases} \quad (6)$$

그림 7은 그룹을 사용했을 때의 데이타 접근 알고리즘이다.

4.2 그룹 크기의 결정

그룹 크기를 결정하기 위해 평균 접근 지연시간과 평균 튜닝 시간을 고려할 수 있다. 우리는 평균 접근 지연시간과 평균 튜닝 시간을 최소화하는 그룹 크기를 결정해야 한다.

평균 접근 지연시간은 방송 프로그램의 길이와 밀접한 관련이 있다. 그룹의 크기가 작을수록 삽입되는 인덱스 테이블의 수가 많아져서 방송 프로그램의 길이는 길어진다. 동시에 그룹의 크기가 클수록 인덱스 테이블 하나의 크기가 커져서 방송 프로그램의 길이가 길어진다. 또한 그룹의 크기가 클수록 인덱스의 방송 주기가 길어

```
Input: Key k
Output: location of data item k
Set tuningPos_i = ∞ for all 1≤i≤numDsk
while(true) {
    read a bucket currently broadcasted
    if ( The bucket contains data item k )
        receive data item k and
    else {
        if (The disk of the bucket does not have the data item k)
            tuningPos_dskNo = -1;
        else if ( The data item k is in this group )
            tuningPos_dskNo = Logical_offset;
        else
            tuningPos_dskNo = q*mcLen - bktNo + r + 1;
    }
    if ( tuningPos_(dskNo+1)%numDsk == ∞)
        tuningPos_(dskNo+1)%numDsk = dskLen - bktNo + 1;
    if ( tuningPos_i== -1 for all 1≤i≤numDsk)
        search failure and return;
    Go into the doze mode and tune in after min_(1≤i≤numDsk) tuningPos_i
}
```

그림 7 그룹을 사용했을 때의 데이타 접근 알고리즘

지면서 초기 탐색 시간이 커져서 오히려 평균 접근 지연 시간이 길어질 수 있다. 그러므로 평균 접근 지연시간을 고려하여 그룹 크기를 결정하면 양 극단의 사이에서 그룹의 크기를 적당히 결정할 수 있다.

그렇지만 평균 튜닝 시간만은 그룹 크기를 결정하는 요소로 사용하기 어렵다. 그룹 크기가 커질수록 평균 튜닝 시간이 작아지기 때문에 튜닝 시간을 고려하면 그룹 크기를 무조건 최대한 크게 결정하면 된다. 튜닝 시간은 방송 프로그램의 길이와 상관없이 프로그램에 대한 튜닝 횟수에만 관계가 있기 때문에 튜닝 시간을 최소화하는 방법은 인덱스를 최대한 많이 늘려서 하나의 인덱스를 보면 원하는 데이타의 위치를 바로 알 수 있도록 하는 것이다. 예를 들어, 트리 인덱스의 경우에는 데이타 아이템 사이사이에 인덱스 트리 전체를 방송하면 튜닝 시간을 최소화할 수 있다. MDEI의 경우에는 그룹 크기를 디스크의 크기와 같게 하면, 인덱스 테이블에 전역 인덱스 대신 지역 인덱스만을 사용하게 되고, 지역 인덱스의 엔트리 수는 디스크의 크기와 같아질 것이다. 이렇게 되면 인덱스 테이블의 크기가 매우 커져서 방송 프로그램의 길이가 제어할 수 없을 만큼 커지며, 그 내용의 대부분이 데이타가 아니라 인덱스로 채워지게 된다. 튜닝 시간을 최소화하는 그룹 크기는 한 극단에 위치하기 때문에 튜닝 시간은 그룹 크기를 결정하는 요소로 사용하기에 부적절하다.

그러므로 그룹 크기의 결정은 평균 접근 지연시간만을 고려하되, 평균 접근 지연시간이 동일한 경우에는 큰 그룹 크기를 선택하는 방법으로 튜닝 시간을 반영할 것이다.

그룹 크기 결정 방법에 대한 설명에 앞서 이 절에서 사용할 용어들을 정리한다. 시스템에는 총 N개의 데이타 아이템이 있다. 데이타 아이템들은 $numDsk$개의 디스크로 나누어져 방송될 것이며, disk k에서는 n_k개의

데이타가 방송되고, $gSize_k$의 그룹 크기를 갖는다. $pDsk_k$는 disk k의 모든 데이타 아이템들의 접근 확률의 합으로써, 클라이언트가 disk k에 접근할 확률을 의미한다. *nChk(k)*와 *freq(k)*는 각각 disk k의 chunk 개수와 상대적인 방송 빈도를 의미하는데, 멀티디스크를 구성하기 위해 주어져야 할 값들이다. 하나의 버켓은 B *bytes*의 용량을 가지며, 인덱스를 함께 방송하는 그룹 내 첫 번째 버켓은 인덱스 부분을 제외하고 B' *bytes*만큼의 데이타를 저장할 수 있다. 하나의 데이타 아이템의 크기는 d이며, 하나의 인덱스 엔트리의 크기는 e이다. 표 2는 이를 정리한 것이다.

표 2 그룹 크기 결정과 관련된 파라미터 및 용어

Notation	Description
N	데이타 아이템의 개수
nk	disk k의 데이타 아이템 개수
$gSizek$	disk k의 그룹 크기 (buckets)
$pDskk$	disk k의 데이타 아이템 접근 확률의 합
B	버켓 size (bytes)
B'	인덱스 테이블을 제외한 버켓 size (bytes)
d	데이타 아이템의 크기 (bytes)
e	인덱스 엔트리의 크기 (bytes)
nChk(k)	disk k의 chunk 수
freq(k)	disk k의 상대적인 방송 빈도
dskLenk	방송 프로그램에서 disk k의 길이 (buckets)
bcLen	방송 프로그램 전체의 길이

4.2.1 최적화된 그룹 크기 결정 방법

우리는 평균 접근 지연시간을 최소화하기 위한 그룹 크기를 결정하고자 한다. 이를 위해 몇 가지 정리를 통해 평균 접근 지연시간과 그룹 크기의 관계식을 얻어낸다. 정리 1은 방송 프로그램의 길이에 대한 것이다.

정리 1. 디스크가 *numDsk*개 존재하며 disk k에는 n_k개의 데이타 아이템이 방송되고, 각각의 디스크마다 그룹 크기 *gSize*는 서로 다르며 disk k의 그룹 크기는 $gSize_k$라고 하자. 또한 disk k의 chunk 수는 *nChk(k)*이고, 상대적인 방송 빈도는 *freq(k)*라고 할 때 한 방송 주기 내에서 disk k의 길이 $dskLen_k$와 방송주기 전체의 길이는 다음과 같다.

$$dskLen_k = \left\lceil \frac{n_k}{\lfloor B'/s \rfloor + (gSize_k - 1) * \lfloor B/s \rfloor} / nChk(k) \right\rceil * gSize_k * nChk(k) * freq(k) \quad (7)$$

$$bcLen = \sum_{k=1}^{numDsk} dskLen_k \quad (8)$$

증명. 인덱스를 함께 방송하는 그룹 내 첫 번째 버켓은 인덱스 부분을 제외하고 B' *bytes*만큼의 데이타를 저장할 수 있다고 할 때, B'는 다음과 같다.

$$B' = B - e * \lceil \log n_k \rceil \quad (9)$$

disk k의 한 그룹에는 $\lfloor B'/d \rfloor + (gSize_k - 1) * \lfloor B/d \rfloor$개의 데이타 아이템이 존재하며, 따라서 disk k에는 $\left\lceil \frac{n_k}{\lfloor B'/d \rfloor + (gSize_k - 1) * \lfloor B/d \rfloor} \right\rceil$개의 그룹이 존재한다. 멀티디스크를 구성하기 위해 이 그룹들을 *nChk(k)*개의 chunk로 나눠야 한다. 그 결과 disk k는 $\left\lceil \left\lceil \frac{n_k}{\lfloor B'/s \rfloor + (gSize_k - 1) * \lfloor B/s \rfloor} \right\rceil / nChk(k) \right\rceil$개의 chunk로 나눠지며, 하나의 chunk의 크기는 $\left\lceil \left\lceil \frac{n_k}{\lfloor B'/s \rfloor + (gSize_k - 1) * \lfloor B/s \rfloor} \right\rceil / nChk(k) \right\rceil * gSize_k$가 된다.

멀티디스크 방송 프로그램에서 disk k의 길이는 disk k의 모든 chunk의 크기의 합에 disk k의 방송 빈도를 곱한 것이다. 그러므로 $dskLen_k$는 chunk 하나의 크기 $\left\lceil \left\lceil \frac{n_k}{\lfloor B'/s \rfloor + (gSize_k - 1) * \lfloor B/s \rfloor} \right\rceil / nChk(k) \right\rceil * gSize_k$에 disk k의 chunk 수 *nChk(k)*와 방송 빈도 *freq(k)*를 곱하면 식 (7)과 같이 계산된다.

또한 디스크의 길이를 모두 합하면 방송 프로그램 전체의 길이를 구할 수 있다. 그러므로 방송 프로그램 전체의 길이 *bcLen*은 $dskLen_k$을 사용하여 식 (8)과 같이 정리할 수 있다. □

평균 접근 지연시간을 구하기 위해서는 초기 탐색 시간을 계산해야 한다. 정리 2는 초기 탐색 시간을 계산하는 방법이다.

정리 2. *numDsk*개의 디스크가 존재하고, disk k의 그룹 크기는 $gSize_k$이고, disk k의 길이를 $dskLen_k$이며, 방송 주기의 길이는 *bcLen*인 방송 프로그램에서 초기 탐색 시간은 다음과 같다

$$(initial\ probe\ time) = \sum_{k=1}^{numDsk} \left(\frac{gSize_k}{2} * \frac{dskLen_k}{bcLen} \right) \quad (10)$$

증명. 디스크마다 그룹 크기가 다르기 때문에 초기 탐색 시간도 디스크마다 서로 다르다. 인덱스는 그룹마다 한 번씩 나타나므로 disk k의 초기 탐색 시간은 $\frac{gSize_k}{2}$가 된다. 초기 탐색이 디스크 k에서 일어날 확률은 디스크 k의 길이와 비례하므로, 이 확률은 $\frac{dskLen_k}{bcLen}$이다.

초기 탐색 시간은 모든 디스크에 대해 초기 탐색시간과 그 확률의 곱에 대한 합으로 계산되며, 식 (10)이 이에 해당한다. □

정리 3. disk 1의 그룹 크기는 $gSize_1$, disk 2의 그룹 크기는 $gSize_2$, disk k의 그룹 크기는 $gSize_k$일 때의 평균 접근 지연시간 avgLatency$_{\{gSize1,\ gSize2,\ \cdots,\ gSizenumDsk\}}$ 다음과 같다.

$$avgLatency_{\{gSize_1, gSize_2, \cdots, gSize_{numDsk}\}} = \sum_{k=1}^{numDsk}\left(\frac{gSize_k}{2} * \frac{dskLen_k}{bcLen}\right) + \sum_{k=1}^{numDsk}\frac{pDsk_k * bcLen}{2 * freq(k)} \quad (11)$$

증명. 평균 접근 지연시간은 초기 탐색 시간과 이후 데이타까지의 접근에 걸리는 시간의 합이다. 초기 탐색 시간은 정리 3에서 구해진다. 이후 데이타까지의 접근에 걸리는 시간은 $1 \le k \le numDsk$의 모든 disk k에 대하여, disk k의 접근 지연시간과 disk k의 접근 확률의 곱을 더한 값과 같다. 여기서 disk k에의 접근 확률은 $pDsk_k$이며, disk k의 접근 지연시간은 $\frac{bcLen}{2 * freq(k)}$이므로, 초기 탐색 이후 접근 지연시간은 다음과 같다.

$$\sum_{k=1}^{numDsk}\frac{pDsk_k * bcLen}{2 * freq(k)} \quad (12)$$

평균 접근 지연시간은 초기 탐색 시간과 이후 데이타까지의 접근에 걸리는 시간의 합이며, 이는 각각 식 (10)와 식 (12)에 해당한다. 그러므로 식 (10)와 식 (12)의 합인 식 (11)은 평균 접근 지연시간을 계산한다. □

식 (11)은 그룹 크기와 평균 접근 지연시간의 관계식이다. 그러므로 우리는 다음과 같이 평균 접근 지연시간을 최소화하는 그룹 크기를 결정할 수 있다.

$$\{gSize_1, gSize_2, \ldots, gSize_{numDsk}\} = \{g_1, g_2, \ldots, g_{numDsk}\}$$
such that $avgLatency_{\{gSize_1, gSize_2, \ldots, gSize_{numDsk}\}} \le avgLatency_{\{g_1, g_2, \ldots, g_{numDsk}\}}$
for all $gSize_k \neq g_k$, $1 \le gSize_k \le n_k$, $1 \le g_k \le n_k$
where n_k is number of data items in disk k (13)

4.2.2 제안하는 그룹 크기 결정 방법

앞에서 설명한 그룹 크기 결정 방법은 평균 접근 지연시간을 최소화할 수 있지만 $O(n^{numDsk})$의 복잡도를 갖는 계산을 거쳐야 한다. 이 계산이 복잡해지는 중요한 요인은 방송 프로그램 전체를 고려하는 계산에 있다. 방송 프로그램 전체를 고려하기 위해서는 $bcLen$을 계산하고, 방송 전체에 비례하는 초기 탐색 시간을 계산해야 하지만 이를 계산하기 위해서는 $gSize_k$값이 먼저 결정되어야 한다. 하지만 $gSize_k$값은 우리가 최종적으로 구해내야 할 값이다.

그러므로 우리는 그룹 크기를 결정하는 데 있어서 기존의 최적화된 방법이 갖는 의미만을 고려한 차선책을 선택한다. 기존의 최적화된 방법은 방송 프로그램 전체에서 평균 접근시간을 줄이는 그룹 크기를 구하자는 것이다. 우리는 방송 프로그램 전체를 고려하는 대신 각각의 디스크별로 평균 접근 지연시간을 최소화하는 $gSize_k$를 계산하는 방법을 선택한다. 각각의 디스크별로 평균 접근 지연시간을 최소화 하면 각각의 디스크는 접근 지연시간을 줄이기 위해 $dskLen_k$를 줄이게 되고, 초기 탐색 시간도 고려 대상이 되어 전체 프로그램의 길이를 줄이고 초기 탐색 시간을 줄이는 효과가 있다. 그리고 클라이언트가 인덱스를 따라 탐색을 할 때도 디스크별로 따로 탐색을 하기 때문에 디스크별로 접근 지연시간을 줄이는 것이 의미가 있다.

이제 디스크별로 평균 접근 지연시간을 최소화하는 방법을 설명한다. 정리 4는 임의의 디스크로 만든 방송 프로그램의 접근 지연시간을 계산하는 방법이다.

정리 4. disk k의 디스크 길이가 $dskLen_k$이고, 방송 빈도는 $freq(k)$이며 그룹 크기는 $gSize_k$일 때, disk k의 데이타만을 별도의 방송 프로그램으로 만든다면 평균 접근 지연시간은 다음과 같다.

$$(average\ access\ latency\ of\ disk\ k) = \frac{gSize_k}{2} + \frac{dskLen_k}{2 * freq(k)} \quad (14)$$

증명. 평균 접근 지연시간은 초기 탐색 시간과 이후 데이타까지의 접근에 걸리는 시간의 합이다. 초기 탐색 시간은 그룹 크기의 반이므로 $\frac{gSize_k}{2}$이 된다. 디스크 길이는 $dskLen_k$이며 그 길이 내에서 데이타는 $freq(k)$번 방송되므로, 모든 데이타는 $\frac{dskLen_k}{freq(k)}$ 마다 한 번씩 방송된다. 그러므로 초기 탐색 이후의 접근 지연시간은 $\frac{dskLen_k}{2 * freq(k)}$이 된다. 평균 접근 지연시간은 초기 탐색 시간과 이후 접근 지연시간을 합하여 식 (21)과 같이 계산된다. □

우리는 disk k의 평균 접근 지연시간, 즉 식 (14)을 최소화하도록 disk k의 그룹 크기 $gSize_k$를 결정한다. 그러므로 $gSize_k$는 다음과 같이 구할 수 있다.

$$gSize_k = g,$$
such that $\frac{g}{2} + \frac{dskLen_{k,g}}{2 * freq(k)} \le \frac{h}{2} + \frac{dskLen_{k,h}}{2 * freq(k)}$
for all $g \neq h$, $1 \le g \le n_k$, $1 \le h \le n_k$
where $dskLen_{k,g} =$
$$\left\lceil \frac{n_1}{\lfloor B/s \rfloor + (g-1) * \lfloor B/s \rfloor} / nChk(k) \right\rceil * g * nChk(k) * freq(k) \quad (15)$$

$dskLen_{k,g}$는 그룹 크기가 g일 때의 disk k의 디스크

길이를 의미한다. 결국 식 (15)에 의해 disk k의 그룹 크기 $gSize_k$는 disk k의 평균 접근 지연시간을 최소화하는 그룹 크기로 결정된다. 이에 따르면 $gSize_k$는 O(n) 시간 내에 계산이 가능하다. 우리는 5장에서 이 장에서 제안하는 그룹 크기 결정 방법의 우수한 성능을 실험을 통해 보일 것이다.

5. 성능 분석

이 장에서는 MHash 인덱스 기법과 지수 인덱스 기법과의 비교를 통해서 본 논문에서 제안하는 MDEI 기법의 성능을 분석한다. 제안하는 MDEI 기법에 대해서도 그룹을 사용하는 경우와 사용하지 않는 경우를 함께 비교하여 그룹을 사용하는 것이 효과적인지 알아본다. 그룹을 사용하는 경우는 MDEI로, 사용하지 않는 경우는 MDEI-noGrp로 표기하겠다. 지수 인덱스는 Exponential로 표기한다.

실험에서는 데이타 접근 확률을 모델링하기 위해서 매개 변수 θ를 갖는 Zipf 분포를 사용하였다. Zipf 분포는 무선 브로드캐스트 환경에서 균일하지 않은(non uniform) 접근 패턴을 모델링하기 위해서 널리 사용된다. 즉, θ의 값이 증가함에 따라 데이타 아이템의 접근 확률은 편향된 분포를 갖게 된다. 별도의 언급이 없을 경우 θ는 1.0의 값을 갖는다.

실험에서는 총 5000개의 데이타 아이템이 방송되며, 버켓의 크기는 1KB이다. 하나의 데이타 아이템은 120 바이트이고, 하나의 인덱스 엔트리의 크기는 4 바이트이다. 표 3은 매개변수를 정리한 것이며, 각 실험에서 별도의 언급이 없을 경우 매개 변수들은 이와 같은 값을 갖는다.

표 3 실험 매개 변수

parameter	description	default value
n	number of data items	5000
bktSize	size of a bucket	1 KB
dataSize	size of a data item	128 bytes
indexSize	size of an index entry	4 bytes

5.1 그룹 크기 결정 알고리즘의 성능

본 논문에서는 평균 접근 지연시간을 줄이기 위해서 그룹을 사용할 것을 제안하였다. 그리고 그룹 크기를 결정하는 최적화된 알고리즘은 계산 복잡도가 너무나 크기 때문에 계산이 간단한 다른 알고리즘을 제안했다. 이 장에서는 이 알고리즘의 결과로 생성된 그룹 크기가 최적화된 알고리즘에 얼마나 가까운 성능을 내는지 알아본다.

그림 8은 최적화된 그룹 크기와 본 논문에서 제시하는 방법으로 구한 그룹 크기를 이용하여 MDEI를 구성했

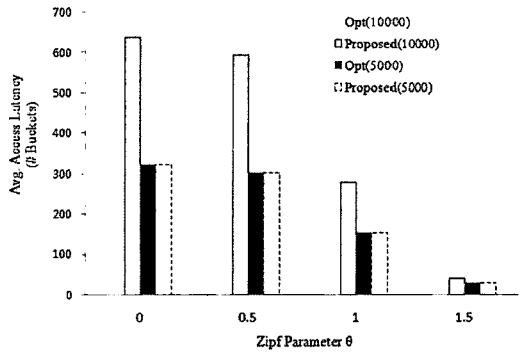

그림 8 그룹 크기 결정 방법에 따른 평균 접근 지연시간

을 때 평균 접근 지연시간을 비교한 것이다. Opt(10000)은 데이타 아이템이 10000개일 때 최적화된 그룹 크기를 이용한 것이고, Proposed(10000)은 역시 데이타 아이템이 10000개일 때 본 논문에서 제시한 방법을 이용하여 그룹 크기를 구한 경우이다.

그림 8을 보면 데이타 아이템 수나 데이타 접근 확률의 편향성 정도에 상관없이, 본 논문에서 제안하는 그룹 크기 결정 알고리즘이 최적화된 알고리즘과 동일한 결과를 나타내는 것을 볼 수 있다. 이는 그룹 크기 결정 알고리즘을 제안할 때 설명한 것과 같이, 각각의 디스크별로 평균 접근 지연시간을 줄이는 것이 전체 디스크에 대한 접근 지연시간을 줄이는 것과 동일선상에 있기 때문이다. 또한 어느 정도 근접한 결과가 아니라 아주 동일한 결과가 나온 것은 그룹 크기 결정의 문제가 이산적인(discrete) 문제라는 이유도 있다.

결과적으로 제안하는 그룹 크기 결정 방법이 계산 시간을 크게 줄이면서도 최적화된 값과 거의 같은 결과를 나타내는 것을 알 수 있다.

5.2 데이타에 대한 접근 확률 분포 변화에 따른 성능

이 장에서는 데이타에 대한 접근 확률 분포 변화에 따라 제안하는 기법이 어떤 성능을 보이는지를 알아본다. 그림 9와 그림 10은 제안하는 기법이 접근 확률 분포 변화에 따라 어떤 성능을 보이는지 알아보기 위하여 zipf factor θ를 변화시키면서 평균 접근 지연시간과 평균 튜닝 시간을 측정한 것이다.

MHash 기법이 θ값에 따라 방송 프로그램이 새로 구성되듯이, 이 실험에서 MDEI 기법도 θ값에 따라 디스크의 수를 달리하여 인덱스를 구성하였다. [9]에서 알 수 있듯이 멀티디스크는 데이타의 접근 편향성이 커질수록 디스크의 수가 많아져야 한다. 즉, θ가 0.0일 때는 한 개의 디스크로 구성되고, θ가 커짐에 따라 디스크의 수가 많아진다. 이 실험에서는 θ가 0.0일 때 1개, 0.5일 때 4개, 1.0일 때 6개, 1.5일 때 8개의 디스크로 방송 프로그램을 구성하였으며, 디스크의 수가 *numDsk*일 때

disk k의 상대적인 방송 빈도는 $2^{numDsk-k}$로 설정하였다. 그러므로 θ가 0.5일 때 디스크들의 상대적인 방송 빈도는 8, 4, 2, 1이 되도록 멀티디스크를 구성하였다.

그림 9는 본 논문에서 제안하는 MDEI 기법이 MHash 기법이나 지수 인덱스 기법에 비해 짧은 접근 지연시간을 갖는 것을 보여준다. 데이타에 대한 접근 확률을 전혀 반영하지 못하는 지수 인덱스와는 매우 큰 성능 차이를 보이며, 접근 확률이 편향됨에 따라 접근 지연시간이 줄어드는 MHash 기법에 비해서도 매우 좋은 성능을 나타낸다. 이런 차이는 데이타 접근 확률에 대한 편향성 정도가 커질수록 더 커진다. MDEI의 평균 접근 지연시간은 θ가 1.0일 때는 MHash 기법의 3/4 정도, 1.5일 때는 1/4 정도에 불과하다. MDEI 기법이 데이타 접근 확률이 편향된 환경을 위한 기법이라는 것을 생각하면 이러한 결과는 매우 의미가 있다고 할 수 있다. 이는 편향성이 커지면 데이타 아이템의 최대 방송 횟수가 커져야 하는데, MDEI는 최대 방송 횟수가 커지지만 MHash 기법에서 최대 반복횟수를 제한하기 때문이다.

그림 10에서 볼 수 있듯이 평균 튜닝 시간은 MHash 기법에 비해 MDEI가 조금 더 크다. MDEI는 디스크별로 별도의 검색을 해야 하기 때문에 튜닝 시간이 길어지는 것으로 생각된다. 하지만 그 차이가 크지 않다. 튜닝 시간에서의 손실은 평균 접근 지연시간에서 얻을 수 있는 이익에 비해 낮다고 할 수 있다.

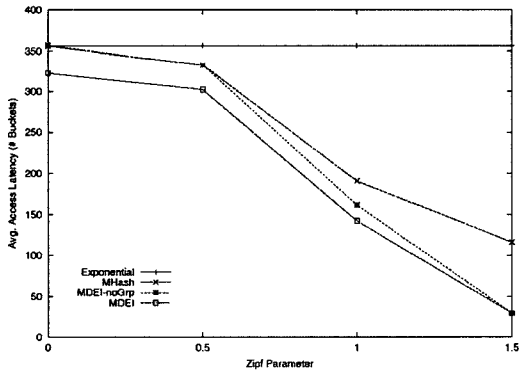

그림 9 Zipf factor에 따른 평균 접근 지연시간

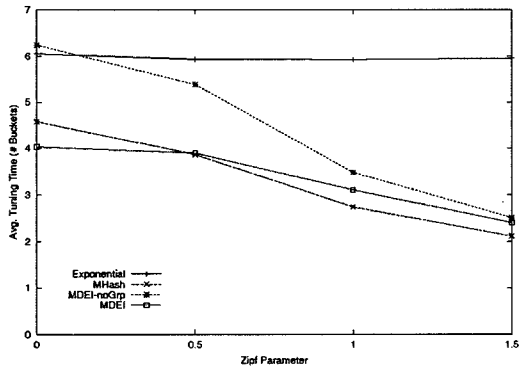

그림 10 Zipf factor에 따른 평균 튜닝 시간

한편 이 실험을 통해 그룹 사용의 효과에 대해서도 살펴볼 수 있다. 그림 9에서 볼 수 있듯이 그룹을 사용하면 평균 접근 지연시간을 줄일 수 있다. 그룹을 사용하면 방송 프로그램에서 인덱스의 비중이 낮아져서 방송 프로그램의 길이가 짧아지기 때문이다. 하지만 편향성이 커지면 디스크가 많아짐에 따라 하나의 디스크에 포함된 데이타 아이템 수가 적어지고, 이에 따라 그룹을 적용하기가 힘들어진다. 튜닝 시간에도 동일한 결과를 볼 수 있다. 그림 10에서 θ가 작을 때는 그룹을 사용하는 것이 큰 효과를 보이며, θ가 클 때는 그룹을 사용한 경우와 사용하지 않은 경우의 차이가 크지 않다. 이를 통해 그룹을 사용하면 접근 지연시간과 튜닝 시간 측면에서 효과가 있으며, 편향성이 작은 경우에 상대적으로 성능이 좋지 않은 특성에 대한 보완책이 될 수 있음을 알 수 있다.

5.3 데이타 아이템 수에 따른 성능

이 장에서는 데이타 아이템 수에 따른 성능을 알아본다. 우리는 데이타 아이템 수를 2500개에서 10000개까지 늘려가면서 접근 지연시간과 튜닝 시간이 어떻게 변하는지 살펴보았다. 그림 11과 그림 12에서 볼 수 있는 것처럼 데이타 아이템 수가 늘어남에 따라 접근 지연시간과 튜닝시간이 모두 증가함을 볼 수 있다. 데이타 아이템 수에 상관없이 접근 지연시간에 있어서는 MDEI가 가장 좋은 성능을 보이고, 튜닝 시간에 있어서는 MHash가 가장 좋은 성능을 보인다. 하지만 튜닝 시간은 데이타 아이템 수가 만 개일 때도 그 차이가 1 버켓 시간이 채 되지 않지만, 접근 지연시간은 그 차이가 100 버켓 시간이 넘는다. 데이타 아이템 수가 많거나 적은 경우에도 MDEI가 효과적인 인덱스 방법임을 알 수 있다.

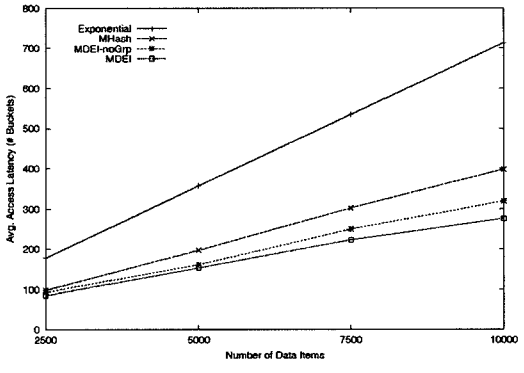

그림 11 데이타 아이템 수에 대한 평균 접근 지연시간

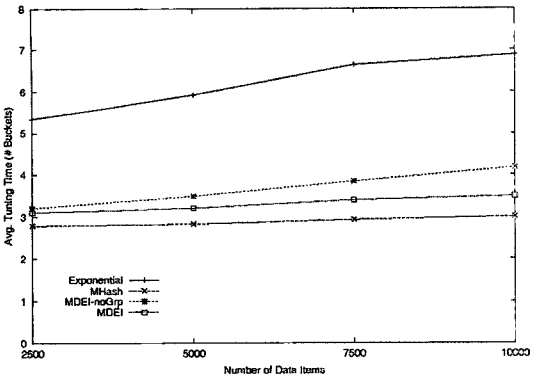

그림 12 데이타 아이템 수에 대한 평균 튜닝 시간

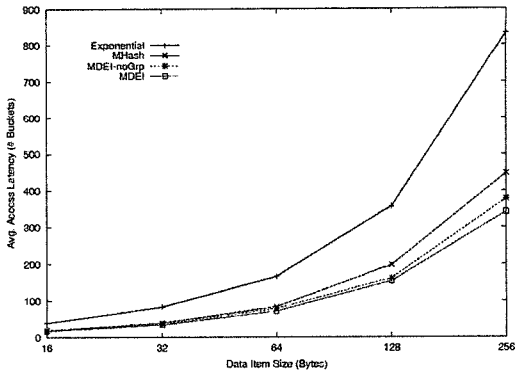

그림 13 데이타 아이템 크기에 대한 평균 접근 지연시간

5.4 데이타 아이템 크기에 따른 성능

다음으로 데이타 아이템 크기를 변화시키면서 제안하는 알고리즘의 성능을 평가한다. 데이타 아이템 크기가 16, 32, 64, 128, 256인 경우에 대해 접근 지연시간과 튜닝 시간의 변화를 시뮬레이션 하였다. 그림 13에서 볼 수 있는 것처럼 접근 지연시간은 데이타 아이템의 크기가 증가함에 따라 함께 증가하였다. 데이타 아이템의 크기가 커지면 방송하는데 걸리는 시간이 길어지기 때문이다. 여러 기법들 중에서는 MDEI가 데이타 아이템의 크기와 상관없이 가장 좋은 성능을 보인다. 기존의 기법들보다 인덱스의 사이즈가 더 길어졌음에도 이와 같은 성능을 보이는 것은, 이러한 인덱스를 이용하여 평균 접근 지연시간을 크게 단축할 수 있기 때문임을 알 수 있다. 다만 데이타 아이템 크기가 클 때는 기법간의 성능 차이가 크지만, 데이타 아이템 크기가 작을 때는 전체적으로 방송 주기 자체가 짧아지고 길어진 인덱스 사이즈의 영향으로 상대적으로 성능 차이가 작음을 볼 수 있다.

그림 14에서는 튜닝 시간을 볼 수 있다. MDEI의 튜닝 시간은 MHash보다 조금 크지만 그 차이가 크지는 않다. 한 가지 특기할만한 점은 지수 인덱스와 그룹을 사용하지 않은 MDEI 기법은 데이타 아이템 크기가 커짐에 따라 튜닝 시간이 길어지지만, 그룹을 사용한 MDEI 기법과 MHash 기법은 데이타 아이템 크기가 커지더라도 튜닝 시간이 크게 길어지지 않는다는 것이다. 데이타 아이템의 크기가 커지면 하나의 버켓에 들어갈 수 있는 데이타 아이템의 수가 작아져서 버켓 수가 많아지고 결과적으로 튜닝 시간이 길어지게 된다. 그렇지만 MHash 기법은 해시 함수를 사용하기 때문에 데이타 아이템의 크기 자체는 큰 문제가 되지 않는다. 그룹을 사용하는 MDEI 기법 역시 그룹이 버켓의 개념을 추상화 하여 그룹마다 하나의 인덱스가 들어가기 때문에 데이타 아이템의 크기가 커지더라도 튜닝 시간이 크게 증가하지 않는다.

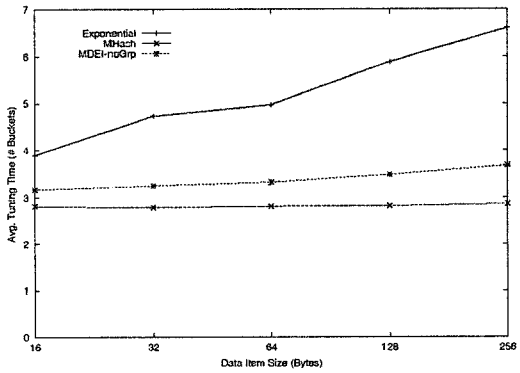

그림 14 데이타 아이템 크기에 대한 평균 튜닝 시간

6. 결 론

본 논문에서는 멀티디스크에서 동작하는 싱글 채널 인덱스 기법인 MDEI (Multi-disk Exponential Index)를 제안하였다. 기존의 싱글 채널 인덱스 기법은 대부분 데이타 아이템에 대한 접근 확률 분포가 편향되는 경우에 접근 지연시간이 매우 긴 flat 브로드캐스트를 기반으로 동작하였다. 또한 non-flat 브로드캐스트를 기반으로 동작하는 MHash 기법도 최대 방송 횟수가 제한되어 있어서 접근 지연시간을 충분히 줄이지 못했다.

본 논문에서 제안하는 MDEI 인덱스 기법은 멀티디스크 방송 프로그램에서 각각의 디스크마다 별도의 인덱스를 구성하여 멀티디스크 방송에 대한 인덱싱이 가능하도록 했다. 멀티디스크는 데이타에 대한 접근 확률이 편향된 환경에서 접근 지연시간을 효과적으로 줄일 수 있다. 실험을 통해 실제로 MDEI가 MHash 기법보다는 조금 큰 튜닝 시간을 갖지만 접근 지연시간을 매우 크게 줄일 수 있음을 보였다. 데이타 접근 확률 분포의 편향성이 클수록 MDEI는 뛰어난 성능을 보인다. 접근 확률 분포가 일정한(uniform) 경우에도 그룹의 사용으로 접근 지연시간을 크게 줄일 수 있음이 또한 확인

되었다. 그 외 몇 가지의 실험을 통해 데이타의 수가 증가하더라도 MDEI의 확장성이 뛰어남을 알 수 있었고, 다양한 데이타의 크기에도 우수한 성능을 나타내는 것을 확인할 수 있었다.

MDEI 기법은 데이타 접근 확률 분포에 따라 MHash보다 큰 튜닝 시간을 보인다. 이는 데이타 아이템을 디스크별로 찾아야 하기 때문이다. 데이타 아이템이 어떤 디스크에서 방송되는지 알 수 있다면 튜닝 시간을 더 줄이는 것이 가능할 것이다. 따라서 향후에 데이타 아이템의 방송 디스크를 알 수 있도록 하여 튜닝 시간을 줄이는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] G. Forman, and J. Zahorjan, The Challenges of Mobile Computing, in IEEE Computer, 27(6), pp. 38-47, April 1994.

[2] T. Imielinski, and B. Badrinath, Wireless Mobile Computing : Challenges in Data Management, Comm. of ACM, Vol. 37, No. 10, pp. 18-28, 1994.

[3] W.C. Lee, and D.L. Lee, "Using Signature Techniques for Information Filtering in Wireless and Mobile Environments," Distributed and Parallel Databases, Vol. 4, No. 3, pp. 205-227, July 1996.

[4] T. Imielinski, S. Viswanathan, and B.R. Badrinath, "Power Efficient Filtering of Data on Air," Proc. Fourth Int'l Conf. Extending Database Technology, pp. 245-258, Mar. 1994.

[5] T. Imielinski, S. Viswanathan, and B. Badrinath, "Data on Air: Organization and Access," IEEE Trans. Knowledge and Data Eng., Vol. 9, No. 3, pp. 353-372, May 1997.

[6] N. Shivakumar, and S. Venkatasubramanian, "Efficient Indexing for Broadcast Based Wireless Systems," ACM/Baltzer Mobile Networks and Applications, Vol. 1, No. 4, pp. 433-446, Dec. 1996.

[7] J. Xu, W. Lee, X. Tang, Q. Gao, and S. Li, "An Error-Resilient and Tunable Distributed Indexing Scheme for Wireless Data Broadcast," IEEE Trans. Knowledge and Data Eng., Vol. 18, No. 3, pp. 392-404, March 2006.

[8] J. Xu, W. Lee, and X. Tang, "Exponential Index: A Parameterized Distributed Indexing Scheme for Data on Air," Proc. ACM/USENIX MobiSys, pp. 153-164, June 2004.

[9] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik, "Broadcast Disks: Data Management For Asymmetric Communications Environments," Proc. ACM SIGMOD Conf. Management of Data, pp. 199-210, May 1995.

[10] Y. Yao, X. Tang, E. Lim, and A. Sun, "An Energy-Efficient and Access Latency Optimized Indexing Scheme for Wireless Data Broadcast," IEEE Trans. Knowledge and Data Eng., Vol. 18, No. 8, pp. 1111-1124, August 2006.

박 기 영

2001년 3월~2006년 2월 서강대학교 컴퓨터공학과 학사. 2006년 3월~2008년 2월 서강대학교 컴퓨터공학과 석사. 2008년 2월~현재 한국무역정보통신 R&D센터 연구원. 관심분야는 이동통신, 모바일 컴퓨팅 시스템, 모바일 데이타베이스

정 성 원

1988년 서강대학교 전자계산학 학사. 1990년 M.S. in Computer Science at Michigan State University. 1995년 Ph.D. in Computer Science at Michigan State University. 1997년~2000년 한국전산원 선임연구원. 2000년~현재 서강대학교 컴퓨터학과 부교수. 관심분야는 Mobile Databases, Mobile Computing Systems, Spatial Databases, Telematics, LBS, Mobile Agents