MapReduce와 시공간 데이터를 이용한 빅 데이터 크기의 이동객체 갱신 횟수 감소 기법

Update Frequency Reducing Method of Spatio-Temporal Big Data based on MapReduce

최 용 권[*] 백 성 하[**] 김 경 배[***] 배 해 영[****]
Youn Gwon Choi Sung Ha Baek Gyung Bae Kim Hae Young Bae

요 약 지금까지 대규모의 이동객체 관리를 위해 갱신 비용을 감소시킬 수 있는 인덱스 기법들이 제안되었다. 이동객체 인덱스는 빈번하게 위치정보가 변화하는 이동객체를 관리하기 위해 주기적으로 갱신되어야 하기 때문이다. 그러나 이러한 기법들은 이동객체의 수가 현저히 증가하는 경우 인덱스의 갱신 가능범위를 초과하는 부하가 발생한다. 본 논문에서는 이처럼 기존의 처리 가용량을 초과하는 빅 데이터 크기의 이동객체에서 발생하는 갱신요청들을 MapReduce와 기존 인덱스기법을 조합하여 감소시키는 기법을 제안한다. 이 기법에서는 수많은 이동객체에서 발생하는 갱신요청들을 MapReduce를 이용하여 각각의 이동객체 별로 그룹화하는 방법을 사용한다. 각 이동객체 별로 그룹화 데이터들의 최신의 데이터와 가장 오래된 데이터를 비교하여 갱신여부를 판단하고 최신의 요청만 갱신하도록 하여 전체 갱신 횟수를 크게 감소시킨다. 갱신이 지연된 경우 기존의 갱신요청들을 가지고 있는 해시 테이블에 일정기간 보관하여 지연된 갱신요청이 분실되지 않고 지속적으로 갱신되도록 한다. 실험을 통해 제안한 기법을 적용한 경우와 적용하지 않은 경우를 비교해 전체 갱신 횟수 및 갱신 비용이 감소되는 것을 알 수 있다.

키워드 : 빅 데이터, 시공간 데이터, 위치기반서비스, 인덱스, 클라우드시스템, 하둡

Abstract Until now, many indexing methods that can reduce update cost have been proposed for managing massive moving objects. Because indexing methods for moving objects have to be updated periodically for managing moving objects that change their location data frequently. However these kinds indexing methods occur big load that exceed system capacity when the number of moving objects increase dramatically. In this paper, we propose the update frequency reducing method to combine MapReduce and existing indices. We use the update request grouping method for each moving object by using MapReduce. We decide to update by comparing the latest data and the oldest data in grouping data. We reduce update frequency by updating the latest data only. When update is delayed, for the data should not be lost and updated periodically, we store the data in a certain period of time in the hash table that keep previous update data. By the performance evaluation, we can prove that the proposed method reduces the update frequency by comparison with methods that are not applied the proposed method.

Keywords : Big Data, Spatial-Demporal Data, LBS, Index, Cloud System, Hadoop

1. 서 론

이동 통신 기술의 발달, GPS 기술의 확산 그리고 스마트폰의 빠른 보급의 영향으로 위치 기반 서비스가 새로운 시장을 창출 할 수 있는 기반 기술로 각광 받고 있다. 이에 따라 다양하고 생활에 밀접한 서비스들이 속속 등장하고 있다. 위치 기반 서비스는 GPS와 같은 기술을 사용하여 이용자의 위치를

[*] 인하대학교 정보공학과 석사과정 sddjin@naver.com
[**] 인하대학교 컴퓨터 정보공학과 박사과정 shbaek@dblab.inha.ac.kr
[***] 서원대학교 컴퓨터교육과 조교수 gbkim@seowon.ac.kr(교신저자)
[****] 중국 중경우전대학교 대학원 명예교수 hybae@inha.ac.kr

파악하고 이와 관련된 다양한 응용 서비스를 지원한다. 위치 기반 서비스를 적용한 응용에는 지도서비스, 길 찾기, 차량 추적, 주변정보 조회, 네비게이션과 같은 생활 편의를 위한 것들이 많이 있다. 이와 같은 응용을 지원하기 위해서는 시간의 흐름에 따라 위치가 변화하는 시공간 데이터인 이동객체의 위치 정보를 효과적으로 관리하기 위한 기술이 필수적이다.

3스마트폰의 빠른 보급은 소셜 네트워크 서비스라는 새로운 환경을 만들어 냈다. 소셜 네트워크 서비스는 취미와 활동을 공유하는 사람들 간의 인적 네트워크 형성을 온라인상으로 지원하는 서비스이다. 페이스북, 트위터가 대표적인 소셜 네트워크 서비스인데 수억 명 이상의 가입자들이 발생하고 있는 글들은 상상할 수 없을 정도로 늘어나고 있고 이는 스마트폰과 소셜 네트워크 서비스가 확산 될수록 더욱 늘어나고 있다. 이처럼 위치기반 서비스의 발달과 소셜 네트워크 서비스의 확산으로 인해 두 기술을 융합한 위치기반 소셜 네트워크 서비스가 등장하고 있다. Google Latitude는 그 대표적인 예로 모바일 기기를 이용하여 자신의 위치정보를 서버로 전달하면 이 위치정보를 저장하여 자신의 위치정보를 실시간으로 전 세계 사람들과 공유 하여 서로의 위치를 확인 할 수 있고 개인의 이동 및 위치 정보를 기록하여 간단한 통계 정보를 제공하고 있다. 이와 같이 그 영역이 매우 큰 응용서비스에서는 기존의 서비스에서 관리하던 이동객체의 수보다 더 많은 이동객체를 관리해야 하고 또한 이들이 생성하는 위치정보 갱신요청 데이터들의 크기도 매우 커지게 된다. 따라서 이러한 서비스를 제공하기 위해서는 기존의 데이터베이스에서 처리하기에는 너무 큰 이동객체 갱신요청 데이터들을 처리할 수 있는 시공간 빅 데이터에 대한 처리 기술이 필요하다. 이동객체는 주기적으로 자신의 위치 정보를 서버로 보내어 이동객체의 정보를 갱신하게 된다. 이렇게 갱신된 정보는 범위 질의, k-최근접 질의, 궤적 질의와 같이 다양한 질의를 처리하는데 사용되고 이와 같은 질의들을 효율적으로 처리할 수 있도록 이동객체의 위치 정보를 관리하는 색인에 대한 많은 연구가 이루어지고 있다.

일반적으로 이동객체를 관리하기 위한 인덱스 방법에서 R-tree[2]가 가장 일반적이다. 그러나 R-tree는 다수의 이동객체에서 발생하는 갱신 연산이 발생 할 때 매우 큰 갱신 오버헤드가 발생한다. 이를 해결 하기 위하여 LUR-tree[3], Bottom-Up Update [11]와 같은 방법이 고안되었다. LUR-tree는 이동객체의 위치를 갱신하는데 불필요한 인덱스의 수정을 줄이고, 이동객체가 해당 최소경계 사각형(MBR; Minimum Bounding Rectangle)을 벗어났을 때의 경우에만 인덱스를 갱신하도록 한다. 그리고 Direct Line라는 보조 인덱스를 이용하여 이동객체를 찾아 빠르게 갱신이 이루어지도록 한다. Bottom-Up Update는 이동객체의 위치를 갱신하기 위하여 단말노드(leaf node)에 대한 보조 인덱스를 생성하고 R-tree의 비 단말 노드에 대한 직접 접근 테이블(direct access table)과 leaf node에 대한 비트 벡터(bit vector)를 생성하여 보조 인덱스에서 검색한 이동객체의 부모노드로 바로 접근 할 수 있게 하여 접근하고 단말 노드가가 가득 찼는지 아닌지를 바로 알 수 있게 하여 빠른 갱신과 노드 분할(node split)이 되었을 때에만 직접 접근 테이블이 갱신되도록 하여 갱신 비용을 감소시킨다. 이와 같은 보조 인덱스를 이용하여 이동객체로의 빠른 접근을 가능하게 하여 이동객체의 갱신비용을 감소시키는 방법들은 이동객체의 크기가 빅 데이터 크기로 증가하는 경우 다음의 문제가 발생한다. 먼저 기존의 인덱스 방법 또한 대용량의 이동객체를 다루기 위한 방법이나 빅 데이터 크기의 이동객체 데이터는 기존의 인덱스 방법들이 기준으로 삼고 있는 10K 크기의 데이터가 아닌 수 MB이상의 크기의 이동객체 데이터이다. 즉 이동객체의 크기가 기존의 크기에 비해 매우 커지게 된다. 이 크기의 이동객체를 기존의 인덱스 방법을 사용하여 관리하면 이동객체의 수가 늘어난 만큼 보조 인덱스의 크기가 매우 커지는 문제가 발생한다. 이는 기존의 인덱스 방법들이 이동객체로의 빠른 접근을 가능하게 하기 위하여 전체 이동객체에 대한 정보를 보조 인덱스에서 관리하기 때문에 위와 같이 이동객체의 수가 증가하게 되면 보조 인덱스의 크기 또한 커지는 문제가 발생한다. 그리고 커진 보조 인덱스의 크기로 인하여 이동객체 데이터에 대한 갱신 비용 또한 증가하게 된다. 현재까지 연구되고 있는 많은 색인 기법들은 어느 정도 대량의 갱신을 고려하여 연구되었다. 그러나 이동객체에서 발생하는 갱신의 크기가 빅 데이터 크기로 커진다면 기존의 색인 기법들은 늘어난 이동객체의 수로 인해 기존의 기법을 사용

하는 경우 인덱스의 허용가능 부하를 초과하는 문제가 발생한다. 기존의 다양한 색인 기법들이 갱신 비용 문제를 해결하기 위해 제안되었지만 이처럼 기존보다 매우 큰 이동객체를 관리하고 이 이동객체들로부터 갱신이 발생하거나 갱신요청이 매우 빈번하게 발생하게 된다면 갱신 비용이 크게 증가하므로 갱신이 점점 지연되어 전체 성능이 크게 저하되는 문제점이 발생한다. 이렇게 빅 데이터 크기의 데이터를 처리할 때 발생하는 문제를 해결하기 위하여 MapReduce[8]를 사용한 방법들이 연구되고 있다. 그 중 [12]는 본 논문과는 그 방향이 다르나 MapReduce를 활용하여 대규모의 이동객체 궤적 데이터를 빠르게 쿼리할 수 있도록 Map과 Reduce를 활용하는 것을 보여준다.

본 논문에서는 위와 같이 대규모의 이동객체에서 발생하는 위치정보 갱신요청 데이터들과 같이 기존의 데이터베이스에서 처리하기가 어려운 시공간 빅 데이터를 갱신할 때 발생하는 갱신 비용 문제를 해결하기 위하여 기존의 인덱스에는 영향을 주지 않고 갱신 정확성 감소를 최소화한 위치정보 갱신 빈도 감소 기법을 제안한다. 제안하는 기법은 하둡의 MapReduce를 이용하여 시공간 데이터가 포함된 갱신요청 데이터를 이동객체를 키로 하여 빠르게 그룹화하고 하나의 그룹화 된 이동객체의 갱신요청 데이터들에 포함되어 있는 시공간 데이터에서 갱신요청 시간데이터와 갱신 위치 데이터를 비교하여 이동객체에서 발생한 여러 갱신요청 데이터 중 최소한의 갱신요청만이 이루어지도록 하여 갱신 횟수를 감소시키도록 한다. 이 갱신 횟수 감소로 인하여 빅 데이터 크기의 이동객체 갱신 시 감소된 갱신횟수로 인하여 전체 갱신 시간을 감소시킬 수 있다. 그리고 갱신 횟수의 감소로 빅 데이터 크기의 이동객체로 인하여 증가된 보조 인덱스에서 갱신 시 발생하는 관리 비용을 감소시킬 수 있다. 또한 본 기법은 이동객체들의 인덱스 기법의 앞부분에서 실행할 수 있음으로써 기존에 사용하는 기법의 보조 인덱스의 변화 없이 그대로 사용이 가능하므로 다양한 인덱스 기법에 적용이 가능하다. 마지막으로 본 기법에서 사용한 MapReduce를 이용하여 전체 시스템을 클라우드 환경으로 확장하면 MapReduce 시 발생하는 부하의 분산 처리가 가능하다.

2. 관련 연구

2.1 MapReduce 모델

구글에 의해 고안된 MapReduce는 함수형 프로그래밍 언어인 LISP을 모델로 하여 Map과 Reduce라는 2개의 함수를 이용하여 대용량 데이터를 쉽고 빠르게 처리할 수 있는 방법론이다. Hadoop[5] MapReduce는 구글 MapReduce의 소프트웨어 구현체이며 현재 빅 데이터 처리를 위한 대표적인 분산 컴퓨팅 플랫폼이다.

MapReduce의 작업은 Map과 Reduce의 두 가지 단계로 나누어진다. Map 단계에서 입력 데이터는 HDFS(Hadoop File System)상의 파일이 될 수도 있고 DB의 레코드들이 될 수도 있다. 어떤 입력 데이터이든지 Map 함수의 입력으로는 키(k1)와 값(v1)의 형태로 들어오게 되는 데 사용자는 Map 함수 내의 입력 데이터인 k1과 v1을 어떻게 처리해서 내보낼지 생각을 하면 되고 그 로직을 Map 함수 내에 작성을 하면 된다. 사용자가 처리한 Map 함수의 출력 키(k2)와 값(v2)은 로컬 디스크에 키를 기준으로 정렬이 되어서 저장이 된다. 모든 Map 처리 단계가 끝나면 하둡 프레임워크는 여러 서버들의 Map의 출력물을 네트워크로 읽어와 합병 정렬 하여 Reduce 함수로 보낸다. Reduce 단계에서는 Map 함수의 출력키를 중심으로 정렬(groupby)된 형태인 키(k2)와 값(v2)의 리스트 형태로 입력이 들어온다. Reduce 함수내의 입력 데이터를 어떻게 처리할 지 생각을 하고 로직을 작성하여 최종적으로 원하는 결과 값을 만들 수 있다. 최종적으로 Reduce의 결과 값은 HDFS에 저장하여, 원할 경우 데이터베이스나 다른 파일시스템에도 저장할 수 있다.

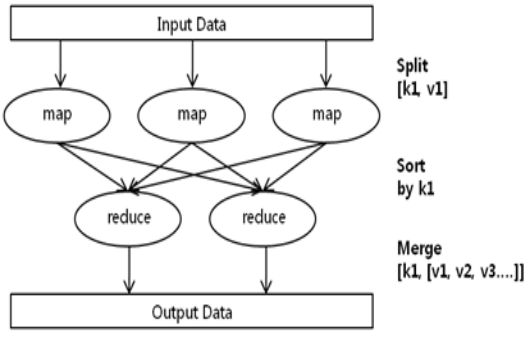

그림 1. MapReduce 처리 과정

2.2 보조 인덱스를 이용한 이동객체 관리 방법

이동객체들을 관리하는 여러 가지 인덱스 기법은 이동객체들의 갱신 연산 비용이 크다는 문제점과 빠른 검색을 위해 모든 이동객체에 대한 보조 색인의 유지비용이 크다는 문제점을 가지고 있다. 이를 해결하기 위해 많은 연구가 진행 되어 왔으며 그 중 대표적인 방법으로 LUR-tree, Bottom Up Update등이 있다.

기존의 R-tree를 이용하여 이동객체를 색인하게 되면, 이동객체의 위치가 변경될 때 단지 데이터만 변경하는 것이 아니라, 데이터의 변경에 의해서 인덱스 노드의 합병 및 분할이 일어나는 등 트리 전체적인 구조에 대한 변경이 발생하게 된다. 이러한 현상은 이동객체의 수가 증가하고 그 위치가 지속적으로 변경되면 많은 갱신 비용이 발생하게 된다. 이러한 문제를 해결하기 위하여 LUR-tree와 Bottom Up Update가 제안되었다. LUR-tree는 이동객체의 새로운 위치가 기존 위치의 MBR의 범위를 벗어나지 않는 경우, 트리의 구조는 변경하지 않고 단지 노드 내부의 위치만을 변경하여 R-tree에서 발생하는 갱신 연산 비용을 감소시킨다. 이를 위해 LUR-tree는 객체 식별자(Object ID)와 객체가 속한 노드의 포인터(ptr)를 쌍으로 갖는 Direct Link라고 하는 보조 인덱스를 사용한다. LUR-tree는 갱신이 일어나면 우선 Direct Link를 통해 객체가 속하는 단말 노드를 검색하고 객체의 새로운 위치가 여전히 단말 노드의 MBR 내에 위치하면, 동일 노드 내에서 위치만 갱신하고, 그렇지 않다면 삭제 및 삽입을 통해 갱신 연산을 수행하며 tree 와 보조 인덱스 Direct Link를 갱신한다.

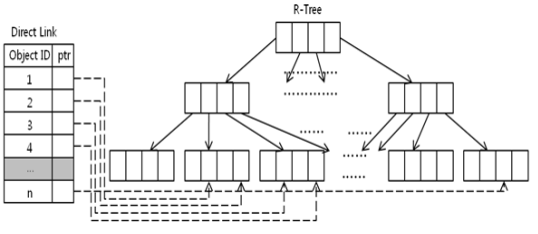

그림 2. LUR-tree 구조[3]

기존 R-tree의 갱신 기법과 LUR-tree의 지연 갱신 기법의 예가 아래 그림에서 보여 진다. 아래 그림의 (a)와 같이 이동객체 Obj1이 현재 위치에서 이동을 하려고 할 때, 기존의 R-tree에서는 이전 위치 값을 (b)에서와 같이 인덱스(R2)에서 삭제하고, 새로운 위치 값을 인덱스(R1)에 삽입한다. 만약 이 R-tree의 리프 노드의 최대 객체의 수를 5개로 가정하자. (c)는 인덱스에 새로운 위치 값을 가지는 Obj1을 인덱스(R1)에 삽입하는 과정에서 오버플로우가 발생하여 노드(R1)가 분리되는 것을 보여준다. 그러나 이 같은 방법은 오버플로우에 의한 노드의 분리가 루트까지 연쇄적으로 발생할 수 있고 최악의 경우에는 루트가 새로 생성될 수도 있기 때문에 갱신 오버헤드가 크다. 따라서 LUR-tree 에서는 (d)와 같이 R2에 속해있던 객체의 새로운 위치가 여전히 R2에 속한다면, 객체의 삭제 및 삽입 과정을 수행하지 않고, 단지 노드(R2) 내의 객체(Obj1)의 위치만을 변경한다.

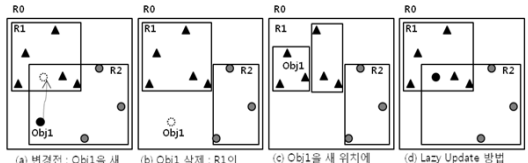

그림 3. 기존 R-tree 갱신 기법과 LUR-tree Lazy Update 기법 예[3]

Bottom-up Update는 R-tree 기반 색인 구조에서 발생하는 갱신 성능을 향상시키기 위하여 노드들에 대한 직접적인 접근을 가능하게 하기 위해 메모리 기반 구조를 사용한다. 이동객체에 대한 갱신을 bottom up 방식으로 수행하기 위해 노드에 대한 summary structure와 보조 인덱스를 사용하였다. 보조 인덱스는 LUR-tree와 같이 이동객체가 저장되어있는 단말 노드들 직접 접근하기 위한 정보를 저장한다. Summary structure는 R-tree 기반의 색인 구조의 중간 노드를 직접 접근하기 위하여 직접

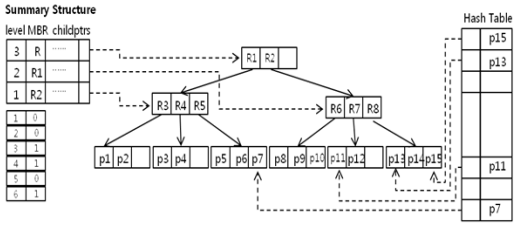

그림 4. R-tree에 대한 Summary structure[11]

접근 테이블과 단말 노드의 상태를 저장하는 비트 벡터로 이루어져 있다.

위와 같이 보조 인덱스를 이용한 인덱스 기법은 약 10~100K 정도의 이동객체를 관리한다. 이는 일반적인 이동객체 스트림 데이터를 관리할 때의 수이다. 그러나 빅 데이터 크기의 이동객체는 수 MB 이상의 이동객체들을 관리해야 한다. 즉 빅 데이터 크기의 이동객체를 관리해야 하는 경우 보조 인덱스를 사용한 기법들은 이동객체들을 빠르게 접근하기 위하여 이동객체의 수가 늘어나는 만큼 보조 인덱스에 이동객체들을 추가하여 관리해야 하므로 전체 관리해야 하는 보조 인덱스의 크기가 빅 데이터 크기의 이동객체 수만큼 증가하게 된다. 그리고 관리하는 이동객체 수의 증가는 이동객체의 갱신요청을 처리할 때 보조 인덱스의 탐색시간이 증가하게 된다. 그리고 이동객체 수의 증가로 인덱스 기법이 처리해야 하는 갱신 횟수 또한 크게 증가하게 된다. 따라서 기존에 비해 전체 갱신 비용이 크게 증가하게 된다.

2.3 MapReduce의 활용한 대규모 데이터 처리

대규모의 이동객체를 관리하기 위하여 MapReduce를 이용한 연구로 [12]가 있다. 이 논문은 대규모의 이동객체의 궤적데이터를 빠르게 쿼리하기 위하여 MapReduce를 이용하여 각 이동객체별로 궤적데이터를 분리해낸다.

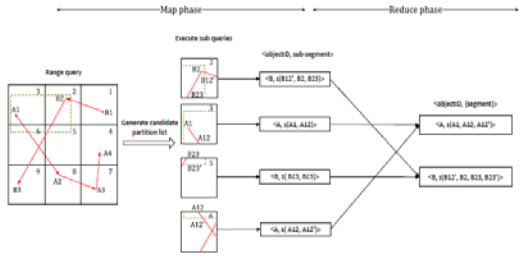

그림 5. MapReduce를 이용한 이동객체 영역 질의 처리[12]

그림 5는 이 과정을 보여주고 있다. 먼저 이동객체의 궤적데이터를 MapReduce를 이용하여 Map단계에서 쿼리 영역을 작은 조각으로 나누어 각 영역에 있는 이동객체의 궤적데이터를 이동객체별로 분리한다. 이 분리된 궤적데이터는 <A, s(A1, A12)> 형태의 각각의 고유한 이동객체 식별자와 부분궤적 데이터들의 쌍으로 이루어진다. 그 후 Reduce 단계에서는 각 쌍들의 데이터들을 각각의 이동객체별로 모아 <A, s(A1, A12, A12)>와 같은 완전한 궤적데이터들과 이동객체 식별자의 쌍으로 그룹화 한다. 이처럼 MapReduce를 활용하면 대규모의 데이터의 처리가 가능해지고 분산된 데이터들을 고유의 키 값별로 데이터를 모아주기 때문에 이후의 데이터 처리가 매우 유용해진다.

3. MapReduce를 이용한 갱신요청 빈도 감소(Update Request Frequency Reduce) 기법

위치 측위 장치들의 대중화로 시간과 공간정보를 포함한 시공간 데이터를 발생하는 객체들이 급격히 증가하고 있다. 그리고 이 증가된 데이터는 기존 데이터에 비해 너무 커서 기존 방법이나 도구로는 다루기 어려운 빅 데이터에 가깝게 변하고 있다. 이에 따라 거대화된 시공간 데이터들에 대한 갱신 비용 또한 급격하게 증가하고 있다. 따라서 거대한 시공간 데이터들을 효율적으로 갱신하기 위한 처리 방법이 요구된다. 따라서 본 논문은 빅 데이터 크기의 이동객체로부터 발생하는 갱신요청 데이터(Update Request Data)들을 효과적으로 처리하기 위하여 하둡의 MapReduce에서 Reduce부분을 확장하여 갱신요청 데이터들을 그룹화 하는 부분에 대하여 설명할 것이다. 그 다음으로 그룹화된 데이터들을 가지고 갱신 시간과 갱신 지점 그리고 갱신 데이터 해시 테이블(Update Data Hash Table)을 비교하여 갱신요청 빈도(Update Request Frequency)를 감소시키는 부분에 대하여 설명할 것이다. 전체적인 프레임워크는 그림 6에서 확인할 수 있다.

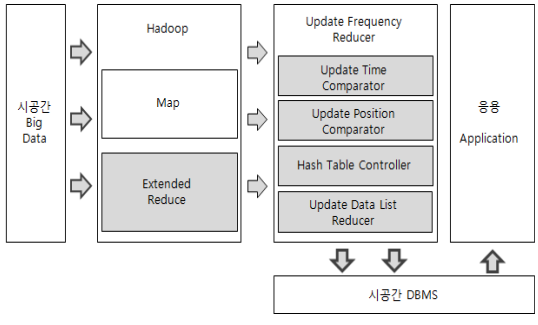

그림 6. 전체 시스템 프레임워크

3.1 MapReduce를 이용한 갱신요청 데이터들의 그룹화

본 장에서는 대용량 이동객체 갱신을 위하여 MapReduce를 이용한 이동객체의 그룹화 방법에 대해 설명한다. 이동객체의 갱신요청 데이터는 갱신을 요청하는 이동객체의 고유ID, 이동객체의 갱신 시간(Update Time) 그리고 이동객체의 갱신 지점(Update Point) 등과 같은 시공간 데이터와 식별 가능한 고유의 ID를 포함하고 있다. 따라서 이러한 ID를 키로 사용하여 데이터에 대한 그룹화가 가능하다.

먼저 MapReduce의 분할과정이 진행된다. 분할과정은 입력된 여러 개의 갱신요청 데이터들에서 갱신을 요청한 각각의 객체 단위로 데이터를 읽어 나눈다. 분할과정에서 나누어진 데이터들은 각각의 Map으로 나뉘어 전달된다. 전달된 데이터들은 Map 과정에서 <key , value>로 이루어진 하나의 쌍의 형태로 변하게 된다. 여기서 키 값은 각 이동객체의 고유한 객체 식별자로 이루어지고 value는 이동객체에 대한 관련 정보로 이루어진다. 아래 그림은 MapReduce의 분할과정과 Map과정을 거쳐 생성된 <key, value> 쌍들을 보여준다.

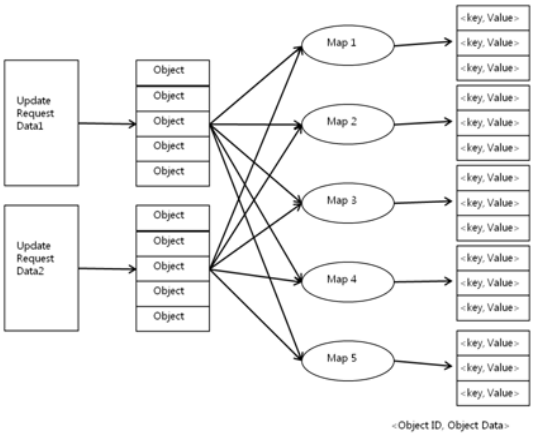

그림 7. MapReduce 에서 분할과정과 Map 과정

위 Map과정으로 생성된 <key, value> 쌍들은 키에 따라 각 Reduce에 전달된다. 각 Reduce는 같은 키를 가지는 <key, value> 쌍들이 전달되게 된다. 이렇게 전달된 <key, value> 쌍들은 Reduce에서 같은 키를 가지는 것끼리 그룹화 된다.

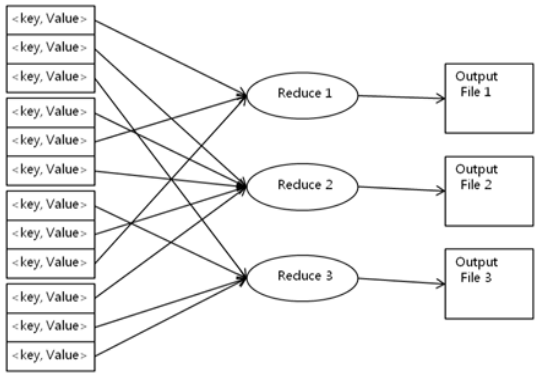

그림 8. MapReduce에서 Reduce 과정

아래의 테이블과 같이 객체 식별자와 객체 데이터로 이루어진 갱신요청 데이터가 입력 데이터로 들어왔다고 가정해본다. 갱신요청 데이터는 Object ID는 각 이동객체를 식별하는 이동객체 식별자, Position은 이동객체의 위치, Update Time은 이동객체의 갱신 요청 시간 시:분:초로 이루어져 있다.

표 1. 갱신요청 데이터

Object ID	Position	Update Time	Object ID	Position	Update Time
006	<9, 80>	10:23:11	004	<20, 61>	10:23:19
002	<60. 12>	10:23:18	006	<14, 65>	10:23:20
007	<45, 27>	10:23:19	007	<55, 29>	10:23:17
001	<40, 40>	10:23:13	001	<48, 41>	10:23:18
004	<12, 60>	10:23:14	008	<63, 15>	10:23:17
001	<45, 42>	10:23:15	008	<58, 13>	10:23:19
007	<50, 30>	10:23:14	007	<50, 28>	10:23:20

이렇게 입력된 데이터는 분할과정에서 각 이동객체별로 나뉘어져서 Map으로 전달된다. Map은 입력된 데이터를 객체 식별자를 키로, 이동객체 데이터를 값으로 하는 <객체 식별자, 객체 데이터(위치, 갱신 시간)> 쌍으로 만든다. 만들어진 <key, value> 쌍들은 동일한 객체 식별자를 가지는 데이터끼리 묶여 Reduce로 전달된다. 이 과정은 map과정에서 생성된 데이터 쌍들을 같은 객체 식별자(키)를 가지는 데이터들이 하나의 Reduce로 전달된다. Reduce1이 키 값이 001, 007인 데이터들을 가지고 오면 결과는 <001, {<40, 40>, <45, 42>, <48, 41> }>, <007, {<45, 27>, <50, 30>, <55, 29>, <50, 28> }> 와 같이 하나의 객체 식별자로 데이터가 그룹화 된다.

표 2. 그룹화 된 데이터

Object ID	Position	Update Time
001	<40, 40>	10:23:13
	< 45, 42>	10:23:15
	<48, 41>	10:23:18
002	<55. 55>	10:23:15
	<60, 57>	10:23:17
	<63, 56>	10:23:19
004	<12, 60>	10:23:13
	<16, 61>	10:23:14
005	<23, 86>	10:23:11
	<25, 83>	10:23:13
	<26, 81>	10:23:14
006	<9, 80>	10:23:11
	<14, 65>	10:23:15
007	<45, 27>	10:23:19
	<50, 30>	10:23:14
	<55, 29>	10:23:17
	<50, 28>	10:23:20
008	<69, 31>	10:23:17
	<72, 33>	10:23:18
	<68, 36>	10:23:20
	<71, 40>	10:23:22
	<69, 43>	10:23:24

MapReduce과정을 수행함으로써 표 1과 같이 정렬되지 않은 갱신요청 데이터들을 각각의 이동객체들에서 발생한 갱신요청 데이터들로 그룹화 함으로써 표 2와 같이 각 이동객체 별로 정렬하는 효과를 얻을 수 있고 가장 마지막에 요청된 갱신요청 데이터를 쉽게 찾을 수 있게 하여 이후 사용될 기법들을 쉽고 빠르게 적용할 수 있도록 한다.

3.2 갱신 데이터 해시 테이블을 이용한 갱신 빈도 감소

본 절에서는 MapReduce를 통하여 그룹화 된 데이터를 이용하여 갱신 데이터 해시 테이블을 생성하는 방법과 갱신 데이터 해시 테이블을 이용한 갱신 빈도를 줄이는 방법에 대해서 설명한다. 본 논문은 갱신 빈도를 감소시키기 위하여 MapReduce를 거쳐 동일한 객체 식별자를 가지는 업데이트 데이터들에서 객체 식별자와 객체 갱신 데이터를 사용한다. 그리고 다음의 데이터로 이루어진 갱신 데이터 해시 테이블을 사용한다. 갱신 데이터 해시 테이블에 포함되는 데이터는 마지막 갱신 시간(Last Update Time), 최근 갱신 시간(Recent Update Time), 마지막 갱신 지점(Last Update Position), 최근 갱신 지점(Recent Update Position)이다. 마지막 갱신 시간은 마지막으로 갱신된 갱신 시간을 저장한다. 마지막 갱신 지점은 마지막 갱신 시간과 마찬가지로 가장 마지막으로 갱신된 요청의 위치를 저장한다. 최근 갱신 시간은 갱신요청이 있었지만 갱신되지 않은 경우 갱신되지 않은 데이터 중에서 가장 마지막의 갱신 시간을 저장한다. 최근 갱신 지점도 최근 갱신 시간과 동일하게 위치 데이터를 업데이트 한다. 그리고 MaxUpdateTime은 갱신 유무를 결정하는 최대 시간의 차이 값으로써 MaxUpdate Time이상의 시간차가 발생하면 갱신을 결정하게 되고 단위는 초이다. 그러나 이 값은 본 논문에서는 10초로 한정하였다. 그리고 MaxUpdateRange의 값은 갱신 유무를 결정하는 최대 거리의 차이 값으로써 이동객체 사이의 유클리디안 거리이고 그 값은 10m로 한정하였다. 그리고 MaxUpdateTimeOut은 최대로 갱신을 지연시킬 수 있는 시간제한 값으로써 10초로 한정하였다. 그 이유는 이 값들은 이동객체들의 갱신 주기와 이동객체들의 이동속도에 따라 이 값들은 가변적으로 설정해야한다. 한 예로 만약 MaxUpdateTime값이 너무 크면 이동객체의 갱신 시간 간격이 너무 커지게 되어 이동객체 정보에 대한 정확성이 감소되고 MaxUpdateTime값이 너무 작으면 갱신 시간 간격이 너무 작아지게 되어 갱신 횟수가 증가하기 때문이다. MaxUpdateRange와 MaxUpdateTimeOut을 한정한 이유도 Max UpdateTime과 동일하다.

3.2.1 갱신 시간 비교를 통한 갱신 결정

갱신 시간 비교 알고리즘은 맵 단계에서 그룹화 된 이동객체 데이터가 가지고 있는 시간정보와 갱신 데이터 해시 테이블이 가지고 있는 시간정보를 서로 비교하여 갱신 여부를 결정한다. 제안 기법은 갱신되지 않은 이동객체에 대한 데이터를 유지하기 위해 갱신 데이터 해시 테이블을 사용한다. 갱신 데이터 해시 테이블에는 객체 식별자, 마지막 갱신 시간, 최근 갱신 시간, 마지막 갱신 지점, 최근 갱신지점을 유지한다. 데이터들이 각각의 이동객체 별로

그룹화 되면 그룹화된 데이터에서 가장 먼저 요청된 데이터와 가장 나중에 요청된 데이터를 찾고, 해당 이동객체의 갱신 데이터를 MaxUpdateTime과 비교하여 갱신여부를 결정한다. 만약 갱신이 되지 않는다면 갱신 지점 비교 알고리즘을 호출한다.

알고리즘 1. 갱신 시간 비교 알고리즘

```
Procedure updateTimeCompare(GroupData gData,
UpdateDataHashTable hashTable,
MaxUpdateTime maxTime, UpdateDataList
updateList)
1:  For each object set oDataset(ID) <- all
object data that has same object ID in gData
2:     gF <- find a late object data in
          oDataset(ID)
3:     gL <- find a recent object data in
          oDataset(ID)
4:     If (gL.updateTime - gF.updateTime)
         < maxTime
5.       gH <- find a gL in hashTable
6.       if gL is in hashTable
7:         If ( gL.updateTime - gH.updateTime)
             < maxTime
8:           call updatePositionCompare
9:         else
10:          add a gL object to updateList
11:        end if
12.      else
13:        call updateHashTable
14:        return
15:      end if
16:    else
17:      add a gL object to updateList
18:    end if
19: return
```

그림 9는 갱신 시간 비교 알고리즘의 예이다. MapReduce Output Data안의 데이터에서 동일한 객체에서 발생한 요청들 중 가장 먼저 요청된 것과 가장 나중에 요청된 것을 각각 Object(gF), Object(gL), 갱신 데이터 해시 테이블에서 가져온 객체를 Object(hT)라고 하자. Object(gF)의 갱신 시간과 Object(gL)의 갱신 시간을 비교한다. Object(gF)와 Object(gL)의 갱신 시간사이의 시간차가 MaxUpdateTime보다 크다면 해당 이동객체는 갱신 데이터 해시 테이블과 비교하지 않고 바로 최종 갱신 리스트에 Object(gL)을 추가 한다. 그리고 Object(gL)의 갱신 데이터를 갱신 데이터 해시 테이블에 업데이트한다. 그러나 Object(gF)와 Object(gL)의 갱신 시간사이의 시간차가 MaxUpdateTime보다 크지 않다면 갱신 데이터 해시 테이블에서 Object(gL)과 같은 아이디를 가지고 있는 Object(hT)를 찾아 Object(hT)가 가지고 있는 데이터를 얻는다. 그리고 Object(gL)의 마지막 갱신 시간과 Object(hT)의 마지막 갱신 시간과 서로 비교하여 최대 업데이트 지연 시간인 MaxUpdateTime이상 시간이 차이가 난다면 Object(gL)에 갱신 데이터를 갱신 데이터 해시 테이블에 업데이트 한 후 갱신 리스트에 Object(gL)를 추가한다. Object(gF)의 갱신 시간과 Object(gL)의 갱신 시간의 비교와 Object(gL)의 갱신 시간과 Object(hT)의 갱신 시간 비교 모두 MaxUpdateTime보다 작다면 이 이동객체는 갱신 리스트에 추가 하지 않고 갱신 데이터 해시 테이블의 최근 갱신 시간과 최근 갱신 지점에

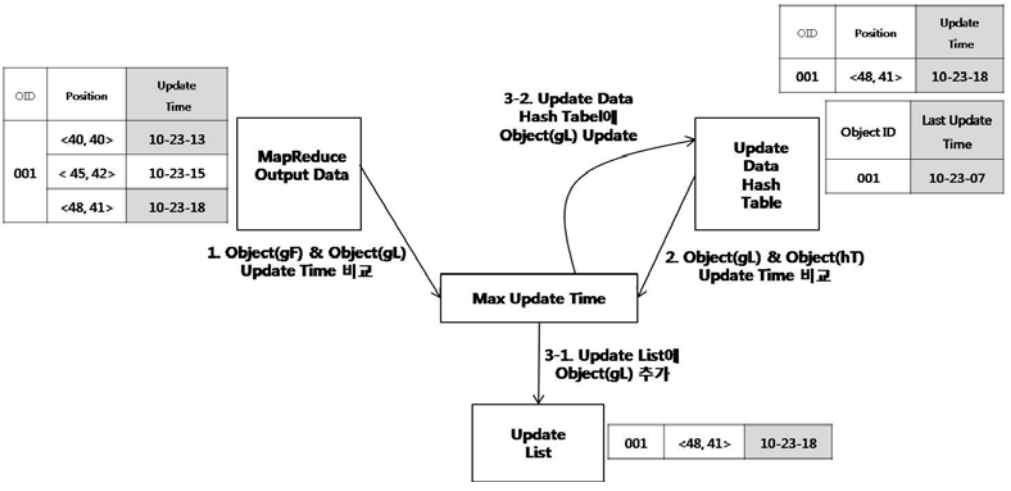

그림 9. 갱신 시간 비교를 통한 갱신 결정 과정

Object(gL)의 갱신 시간과 위치를 업데이트 한다. MaxUpdateTime을 10이라 가정하고 객체 식별자 001, 004와 007가 갱신요청을 처리한다고 가정해보자. 객체 식별자 001의 데이터 중 Object(gL)의 갱신 시간은 10:23:18이고 Object(gF)의 갱신 시간은 10:23:13이다. Object(gL)과 Object(gF)의 갱신 시간 사이의 시간 차이는 MaxUpdateTime인 10을 넘지 못하므로 갱신 데이터 해시 테이블에서 객체 식별자 001을 찾는다. 갱신 데이터 해시 테이블에 있는 Object(hT)의 마지막 갱신 시간은 10:23:07이다. 두 갱신 시간 10:23:18과 10:23:07은 MaxUpdateTime인 10을 넘으므로 객체 식별자 001를 갱신 리스트에 추가 한 후 객체 식별자 001의 갱신 시간과 위치 데이터를 갱신 데이터 해시 테이블의 마지막 갱신 시간과 마지막 갱신 지점에 업데이트 한다. 객체 식별자 007의 데이터 중 Object(gL)의 갱신 시간은 10:23:20이고 Object(gF)의 갱신 시간은 10:23:09이다 두 갱신 시간의 시간 차이는 MaxUpdateTime인 10을 넘으므로 바로 갱신 리스트에 객체 식별자 007을 추가 한 후 객체 식별자 007의 갱신 시간과 위치 데이터를 갱신 데이터 해시 테이블의 마지막 갱신 시간과 마지막 갱신 지점에 업데이트 한다. 객체 식별자 004의 데이터 중 Object(gL)의 갱신 시간은 10:23:15이고 Object(gF)의 갱신 시간은 10:23:13이다 두 갱신 시간의 시간 차이는 MaxUpdateTime을 넘지 못하므로 갱신 데이터 해시 테이블에서 Object(hT)를 찾는다. 갱신 데이터 해시 테이블에 있는 Object(hT)의 마지막 갱신 시간은 10:23:09이다. Object(gL)의 갱신 시간 10:23:15 와 Object(hT)의 갱신 시간 10:23:09는 MaxUpdateTime을 넘지 못하므로 갱신 리스트에 추가 하지 않고 객체 식별자 004의 갱신 시간과 위치 데이터를 갱신 데이터 해시 테이블의 최근 갱신 시간과, 최근 갱신 지점에 업데이트 한다.

3.2.2 갱신 지점 비교를 통한 갱신 결정

갱신 지점 비교 알고리즘은 갱신 시간 비교 알고리즘에서 갱신 여부가 결정되지 않았을 때 그룹화된 이동객체 데이터가 가지고 있는 위치정보와 갱신 데이터 해시 테이블이 가지고 있는 위치정보를

알고리즘 2. 갱신 지점 비교 알고리즘

```
Procedure updatePositionCompare(ObjectDataSet
oDataset(ID), UpdateDataHashTable hashTable,
MaxUpdatePosition maxPosition, UpdateDataList
updateList)
1:  gF <- find a late object data in oDataset(ID)
2:  gL <- find a recentobject data in
          oDataset(ID)
3:  If (gL. updatePosition - gF. updatePosition)
       < maxPosition
4.    gH <- find a gL in hashTable
5.    if gL is in hashTable
6:      If ( gL. updatePosition - gH.
            updatePosition) < maxPosition
7:        call updateHashTable
8:      else
9:        add gL to updateList
10:     end if
11.   else
12:     call updateHashTable
13:     return
14:   end if
15: else
16:   add a gL to updateList
17: end if
18: return
```

표 3. 갱신 전의 갱신 데이터 해시 테이블

Object ID	Last Update Time	Recent Update Time	Last Update Position	Recent Update Position
001	10:23:07	-	<38, 36>	-
004	10:23:06	-	<10, 58>	-
007	10:23:08	-	<43, 22>	-

표 4. 갱신 후의갱신 데이터 해시 테이블

Object ID	Last Update Time	Recent Update Time	Last Update Position	Recent Update Position
001	10:23:18	-	<48, 41>	-
004	10:23:06	10:23:14	<10, 58>	<16, 61>
007	10:23:20	-	<50, 28>	-

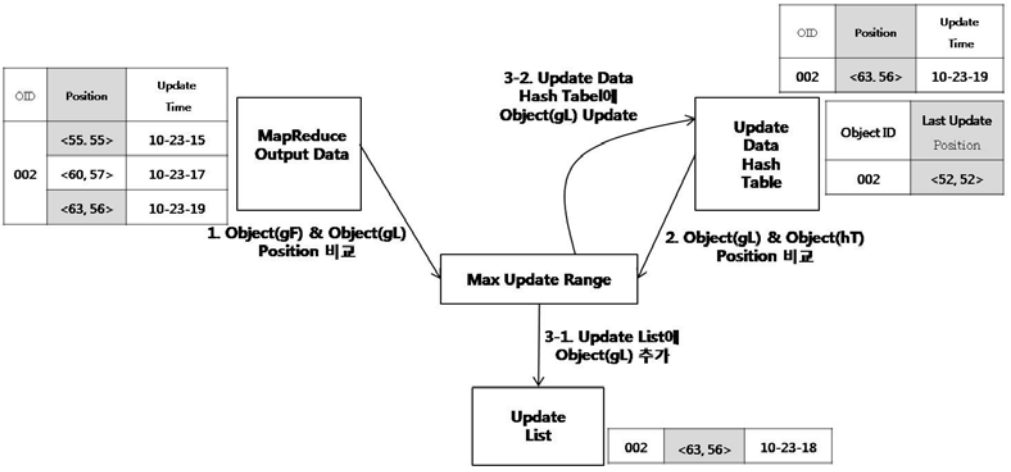

그림 10. Update Position 비교를 통한 갱신 결정 과정

서로 비교하여 갱신 여부를 결정한다. 갱신 시간 비교 알고리즘과 유사하게 그룹화된 데이터에서 가장 먼저 요청된 데이터와 가장 나중에 요청된 데이터를 찾고, 해당 이동객체의 갱신 데이터를 Max UpdatePosition과 비교하여 갱신여부를 결정한다. 만약 갱신이 되지 않는다면 갱신 데이터 해시 테이블에 해당 이동객체에 대한 데이터를 저장한다.

그림 10은 갱신 지점 비교 알고리즘의 예이다. Object(gF)의 위치와 Object(gL)의 위치를 비교한다. Object(gF)와 Object(gL)의 위치 차이가 MaxUpdateRange보다 크다면 이 객체는 갱신 데이터 해시 테이블과 비교하지 않고 바로 갱신 리스트에 Object(gL)을 추가 하고 Object(gL)의 갱신 시간 값과 위치 값을 갱신 데이터 해시 테이블의 O(hT)의 마지막 갱신 시간과 마지막 갱신 지점에 업데이트 한다. Object(gF)와 Object(gL)의 위치사이의 거리 차이가 MaxUpdateRange보다 크지 않다면 갱신 데이터 해시 테이블에서 Object(gL)과 같은 오브젝트인 Object(hT)를 찾는다. Object(gL)의 위치와 O(hT)의 마지막 갱신 지점과 서로 비교하여 MaxUpdateRange이상 차이가 나면 갱신 리스트에 Object(gL)을 추가한다. 그리고 Object(gL)의 갱신 시간 값과 위치 값을 갱신 데이터 해시 테이블의 Object(hT)의 마지막 갱신 시간과 마지막 갱신 지점에 업데이트 한다. Object(gF)의 위치 값과 Object(gL)의 위치 값의 차이와 Object(gL)의 위치 값과 Object(hT)의 마지막 갱신 지점 값의 차이가 모두 MaxUpdateRange보다 작다면 이 이동객체는 갱신 리스트에 Object(gL)을 추가하지 않고 갱신 데이터 해시 테이블의 최근 갱신 시간과 최근 갱신 지점에 Object(gL)의 갱신 시간과 위치를 업데이트 한다. MaxUpdateRange를 10이라 가정하고 객체 식별자 002, 005 그리고 006이 갱신요청을 했다고 가정해보자. Output Data에서 객체 식별자 002의 Object(gF)의 위치 값은 <55, 55> 이고 Object(gL)의 갱신 지점은 <63, 56> 이다. Object(gL)과 Object(gF)의 갱신 지점 값의 차이는 MaxUpdate Range를 넘지 않으므로 갱신 리스트에 추가를 하지 않는다. 그리고 갱신 데이터 해시 테이블에서 Object(hT)의 최근 갱신 지점 값 <48, 46>과 Object(gL)의 위치 값을 비교한다. 두 이동객체 사이의 거리는 MaxUpdateRange를 넘으므로 Object (gL)을 갱신 리스트에 추가 한 다. 그리고 갱신 데이터 해시 테이블에 Object(hT)의 마지막 갱신 시간과 마지막 갱신 지점 값에 Object(gL)의 갱신 시간 값과 위치 값 <63, 41>을 업데이트 한다. Output Data에서 객체 식별자 006의 Object(gF)의 위치 값은 <9, 80>이고 Object(gL)의 위치 값은 <14, 65> 이다. Object(gF)과 Object(gL)의 위치 값의 차이는 MaxUpdateRange를 넘으므로 바로 갱신 리스트에 Object(gL)을 추가 한다. 그리고 갱신 데이터 해시 테이블에 Object(hT)의 마지막 갱신 시간과 마지막 갱신 지점 값에 Object(gL)의 갱신 시간 값과 위치 값 <14, 65>를 업데이트 한다. 객체 식별자 005의 데이터 중 Object(gL)의 위치은 <26, 81>이고 Object(gF)의 위치는 <23, 86>이다.

두 위치 값의 차이는 MaxUpdateRange를 넘지 못하므로 갱신 데이터 해시 테이블에서 Object(hT)를 찾는다. 갱신 데이터 해시 테이블에 있는 객체 식별자 005인 Object(hT)의 최근 갱신 지점 값은 <21, 88>이다. Object(gL)의 위치 값 <26, 81>과 Object(hT)의 최근 갱신 지점 값 <21, 88>의 차이는 MaxUpdateRange값을 넘지 않으므로 갱신 리스트에 추가 하지 않고 객체 식별자 005의 갱신 시간과 위치 데이터를 갱신 데이터 해시 테이블의 최근 갱신 시간과, 최근 갱신 지점에 업데이트 한다.

3.3 갱신 데이터 해시 테이블을 이용한 메모리 관리

3.3.1. ObjectUpdateTimeMinHeap을 이용한 갱신 데이터 해시 테이블 메모리 관리

갱신 데이터 해시 테이블을 이용한 객체들의 관리는 메모리의 한계가 존재한다. 이동객체들의 수가 매우 많은 경우 모든 이동객체 들을 갱신 데이터 해시 테이블에서 다 관리할 수가 없다. 따라서 이 테이블의 크기를 일정하게 하고 현재 가장 빈번하게 갱신되고 있는 이동객체들로 테이블을 유지하고 있어야 빈번하게 갱신이 발생하는 이동객체들의 갱신 횟수를 감소시킬 수 있을 뿐만 아니라 메모리의 사용도 효율적으로 관리 할 수 있다. 본 절에서는 이러한 문제를 해결하기 위해 ObjectUpdate TimeMinHeap을 사용하였다. ObjectUpdateTime MinHeap은 갱신 시간 값을 키 값으로 하고 Object ID 값을 value로 가지는 Minimum Heap이다. 갱신 시간을 Min Heap으로 유지하면 루트노드에는 항상 갱신을 한 지 가장 오래된 객체의 갱신 시간 값이 위치하게 된다. 루트노드에 있는 이동객체는 갱신을 수행한 지 가장 오래된 이동객체이므로 메모리를 일정하게 유지하고자 할 때 루트노드에 있는 이동객체를 삭제하여 쉽게 관리가 가능하다. 갱신 데이터 해시 테이블에 있는 이동객체 정보를 갱신하게 되면 현재 Min Heap의 크기를 최대 힙 크기와 비교하여 최대 힙 크기보다 작으면 Min Heap과 갱신 데이터 해시 테이블에 추가를 하고 그렇지 않으면 Min Heap의 루트노드에 있는 이동개체를 Min Heap과 갱신 데이터 해시 테이블에서 삭제하고 입력된 이동객체를 Min Heap과 갱신 데이터 해시 테이블에 추가한다.

알고리즘 3. 갱신 데이터 해시 테이블 관리 알고리즘

```
Procedure updateHashTable(UpdateDataHashTable
hashTable, MinHeap heap, MaxHeapSize
maxSize, Object gL)
1:  If heap.Size < maxSize
2:    if a gL object is in hashTable
3:      update a gL data to hashTable
4:      update a gL data to heap
5:    else
6:      add a gL to heap
7:      add a gL to hashTable
8:    end if
9:  else
10      rootObj <- a root element of heap
11:     delete rootObj from heap
12:     delete rootObj from hashTable
13:     add obj to heap
14:     add obj to hashTable
15:     update rootObj data
16: end if
```

표 5. 갱신 전의 갱신 데이터 해시 테이블

Object ID	Last Update Time	Recent Update Time	Last Update Position	Recent Update Position
002	10:23:11	-	<52, 52>	-
005	10:23:10	-	<21, 88>	-
006	10:23:09	-	<5, 75>	-

표 6. 갱신 후의 갱신 데이터 해시 테이블

Object ID	Last Update Time	Recent Update Time	Last Update Position	Recent Update Position
002	10:23:19	-	<63, 56>	-
005	10:23:09	10:23:14	<21, 88>	<26, 81>
006	10:23:15	-	<14, 65>	-

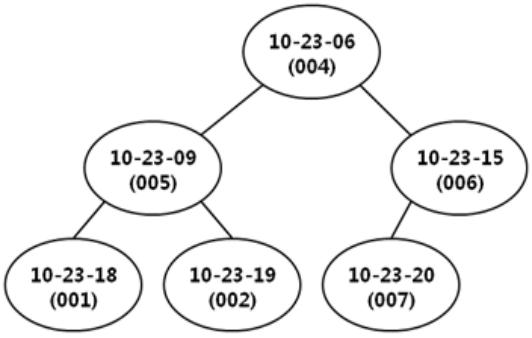

그림 11. 노드 삭제 전의
ObjectUpdateTimeMinHeap

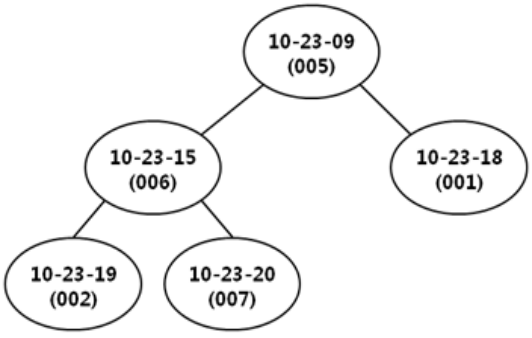

그림 12. 노드 삭제 후의
ObjectUpdateTimeMinHeap

그림 11과 그림 12는 갱신 데이터 해시 테이블 관리 알고리즘에 의해 변화하는 Min Heap의 예이다. 새로운 이동객체가 갱신 데이터 해시 테이블에 추가 하는 경우 ObjectUpdateTimeMinHeap의 크기를 확인한다. ObjectUpdateTimeMinHeap의 크기가 MaxObjectUpdateTimeMinHeapSize보다 작으면 ObjectUpdateTimeMinHeap에 이동객체의 갱신 시간 값을 키 값으로 그리고 Object ID를 Value로 하여 ObjectUpdateTimeMinHeap에 추가 한다. 이와 반대로 ObjectUpdateTimeMinHeap의 크기가 Max Size이면 ObjectUpdateTimeMinHeap에서 루트노드를 확인한 후 루트노드를 삭제한다. 루트노드에 있는 이동객체가 갱신을 수행한 지 가장 오래된 이동객체이므로 갱신 데이터 해시 테이블에서 해당 이동객체를 삭제한다. 그리고 새로운 이동객체를 갱신 데이터 해시 테이블에 추가 한 후 ObjectUpdate TimeMinHeap도 추가를 해준다. 그러나 루트노드의 이동객체를 삭제 하는 경우 갱신 데이터 해시 테이블에서 해당 이동객체의 데이터 값이 최근 갱신 시간 값과 최근 갱신 지점 값이 있는 경우에는 갱신을 실행한 시간은 가장 오래 됐지만 실제 갱신을 해야 했으나 보류했던 이동객체이다. 따라서 이러한 이동객체들은 해당 이동객체를 최근 데이터를 가지고 갱신 리스트에 추가해주어 갱신을 보류했던 이동객체들의 최근 데이터 값이 손실 되지 않도록 한다.

3.3.2 ObjectUpdateTimeMinHeap 모니터링을 통한 메모리 관리

ObjectUpdateTimeMinHeap은 주기적으로 Heap의 Minimum 값의 갱신 시간이 MaxUpdate TimeOut 시간 이상 지났는지 체크한다. Min 값의 갱신 시간이 MaxUpdateTimeOut 시간 보다 지났다면 Min Value인 객체 식별자를 가지고 갱신 데이터 해시 테이블에서 해당 객체 식별자의 데이터를 가져온다. 그 다음 해당 객체 식별자의 데이터에서 최근 갱신 시간 데이터가 있는지 체크한다. 최근 갱신 시간이 있다면 갱신요청은 있었으나 갱신은 실행되지 않고 지연되었다는 것이므로 해당 객체 식별자의 최근 갱신 데이터를 가지고 갱신 리스트에 추가한다. 최근 갱신 시간 데이터가 없는 이동객체라면 업데이트를 수행한 후 MaxUpdate Time Out 시간 이상 지난 이동객체이므로 Object UpdateTimeMinHeap과 갱신 데이터 해시 테이블에서 해당 객체 식별자에 대한 데이터를 삭제한다. 이렇게 Object Time Min Heap을 이용하여 갱신 기록이 가장 과거인 이동객체를 상수시간 내에 빠르게 찾도록 하고 갱신 데이터 해시 테이블에서 해당하는 객체 식별자를 가진 데이터를 삭제함으로 써 갱신 데이터 해시 테이블의 크기를 일정하게 관리한다.

4. 성능 평가

4.1 평가 환경

성능평가에 사용된 시스템 환경은 Window7 32bit OS환경에서 CPU는 AMD Phenom 2 X4 955 3.2GHz이고 메모리 용량은 4GB이다. 그리고 하둡을 사용하기 위한 Linux 환경을 구성하기 위하여 Cygwin을 사용하였다. Cygwin은 Window 환경에서 Linux 환경을 만들어 주는 프로그램으로, Linux,

UNIX 명령어의 사용을 가능하게 해준다. 하둡은 Hadoop-0.20.2 버전을 사용하였다. 하둡의 Map Reduce에서 사용한 Task의 수는 각각 2개이다. 본 실험에 사용한 이동객체는 [16]의 기법을 사용하였다. 생성한 이동객체의 수는 100K이고 각각의 이동객체에서 매 시간 t(1초)마다 갱신요청 데이터가 발생할 확률을 5%(하, 약 120k개 갱신요청), 15%(중, 약 185k개 갱신요청), 25%(상, 약 250k개 갱신요청) 별로 측정 하여 갱신요청 빈도를 조절 하였다. 그리고 임계값 MaxUpdateTime과 MaxUpdateTimeOut은 10초로 MaxUpdateRange는 10m로 설정하였다.

4.2 성능 평가

본 절에서는 제안 기법의 성능을 분석하기 위하여 4.1절에서 제시한 실험 환경에서 실험을 진행한다. 실험은 MapReduce의 그룹화 되는 비율과 각각의 기법을 적용하였을 때 감소되는 갱신의 비율 그리고 최종적으로 각각의 기법을 모두 적용하여 최종적으로 감소되는 갱신요청 데이터의 비율 측정을 수행한다.

4.2.1 MapReduce로 그룹화 된 데이터들을 MaxUpdateTime을 적용하여 감소된 갱신요청 데이터 비율

이번 측정은 빅 데이터 크기의 갱신요청 데이터들을 MapReduce로 각각의 이동객체 별로 그룹화한 데이터들을 가지고 각 이동객체의 갱신 시간을 비교하여 MaxUpdateTime을 초과한 것만 갱신할 경우 감소되는 갱신요청 데이터의 비율을 확인하기 위한 것이다.

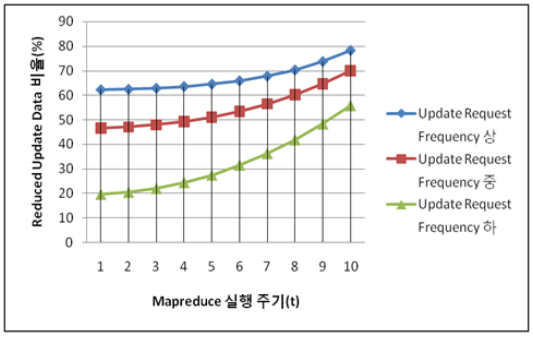

그림 13. UpdateTime을 비교 했을 시 MapReduce 실행 주기에 따른 갱신요청 데이터들의 감소 비율

그림 13는 MapReduce를 실행하는 주기에 따라 이동객체의 갱신 시간을 비교한 경우 실제 실행한 갱신요청 데이터들의 비율을 갱신요청 빈도에 따라 측정한 결과이다. MapReduce를 실행하는 주기를 점차 늘려가며 측정한 결과 전체적으로 실제 갱신을 실행한 데이터의 양이 감소되는 것을 볼 수 있다. 이는 MapReduce를 실행하는 주기가 길어질수록 하나의 이동객체로 그룹화 되는 갱신요청 데이터들이 많아지고 MaxUpdateTime을 비교 후 가장 마지막의 갱신요청만 처리되기 때문에 전체 들어온 갱신요청 데이터들의 수에 비해 감소되는 갱신요청 데이터의 양이 증가하기 때문이다. 특히 갱신요청 빈도가 높을수록 감소된 갱신요청 데이터의 양이 많은 것을 볼 수 있는데 이 또한 이 전과 같이 전체 갱신요청 데이터들의 수가 늘어날수록 감소되는 갱신요청 데이터의 양이 증가 하므로 감소되는 비율 또한 증가하는 것으로 나타나게 된다.

4.2.2 MapReduce로 그룹화 된 데이터들을 MaxUpdateRange를 적용하여 감소된 갱신요청 데이터 비율

이번 측정은 이전 측정과 동일하나 이동객체의 갱신 시간의 비교가 아닌 이동객체의 갱신 지점을 비교하여 MaxUpdateRange를 초과한 것만 갱신할 경우 감소되는 갱신요청 데이터의 비율을 확인하기 위한 것이다

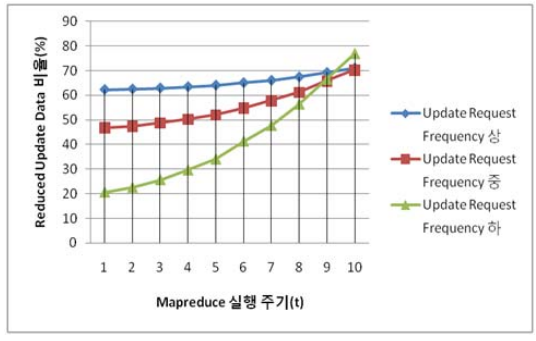

그림 14. 갱신 지점을 비교 했을 시 MapReduce 실행 주기에 따른 갱신요청 데이터들의 감소 비율

그림 14는 그림 13와 동일하게 MapReduce를 실행하는 주기에 따라 이동객체의 갱신 지점을 비교한 경우 실제 실행한 갱신요청 데이터들의 비율을

갱신요청 빈도에 따라 측정한 결과이다. 그림 13와 마찬가지로 MapReduce를 실행하는 주기를 증가시키며 측정한 결과 실제 갱신을 실행한 데이터의 양이 감소되는 것을 볼 수 있다. 이는 그림 14와 동일한 이유로 감소되는 양이 증가 하는 것이고 특히 갱신요청 빈도가 낮은 경우 그 증가 하는 비율이 커지는 것을 볼 수 있다. 이는 갱신요청 빈도가 낮은 경우 MapReduce 실행 주기가 늘어날수록 전체 갱신요청 데이터 대비 감소되는 양이 상대적으로 많은 비율로 나타나기 때문이다. 그리고 그림 13에서와 동일하게 갱신요청 빈도가 높을수록 감소되는 갱신요청 데이터의 양이 커짐을 확인 할 수 있다.

4.2.3 갱신 시간과 갱신 지점을 모두 비교하여 감소된 갱신요청 데이터 비율

이번 측정은 4.2.1와 4.2.2 두 개를 같이 적용하여 실제 본 논문의 기법이 적용되었을 경우 감소되는 갱신요청 데이터의 비율을 확인하기 위한 것이다.

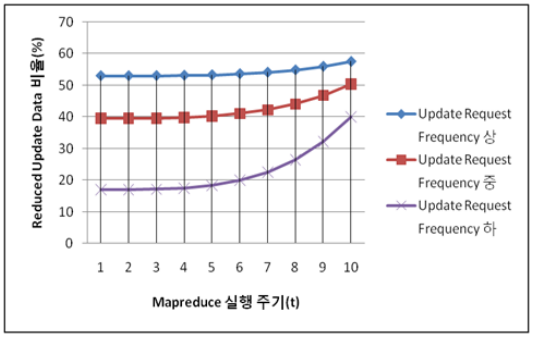

그림 15. 갱신 시간과 갱신 지점을 모두 비교했을 때 MapReduce 실행 주기에 따른 갱신요청 데이터들의 감소 비율

그림 15는 실제 본 논문이 제시한 기법이 적용되었을 경우 MapReduce 실행 주기 크기에 따라 감소되는 갱신요청 데이터의 비율을 보여준다. 그림 15도 마찬가지로 MapReduce 실행 주기가 길어질수록 감소되는 갱신요청 데이터의 비율이 증가함을 볼 수 있다. 그러나 그림 14과 그림 15와는 조금 다르게 갱신요청 빈도가 많은 경우 감소되는 비율이 크게 변화하지 않는 것을 볼 수 있다. 이는 앞서 측정한 갱신 시간을 비교한 경우와 갱신 지점을 비교한 경우 모두 높은 비율로 감소하였기 때문에 최종적으로 실제 감소한 비율은 크게 차이가 나지 않게 되는 것이다. 또한 그림 13과 그림 14를 비교하면 전체적으로 감소 비율이 낮아진 것을 확인 할 수 있다. 이는 갱신 시간과 갱신 지점을 모두 적용함으로써 전체 갱신이 실행되는 횟수가 증가하기 때문이다. 그리고 갱신요청 빈도가 낮을 경우 MapReduce 실행 주기가 커질 경우 감소되는 갱신요청 데이터의 비율이 커짐을 볼 수 있다. 이는 빈도가 낮을 경우 MapReduce 실행 주기를 크게 가져가면 높은 감소율을 볼 수 있다는 것을 알려준다. 그리고 갱신요청 빈도가 높을수록 높은 감소율을 보여 주고 있다. 즉 빈번하고 많은 갱신요청이 발생할수록 본 기법을 적용할 시 높은 효율을 얻을 수 있음을 보여준다.

4.2.4 MapReduce 실행 주기에 따른 전체 갱신 시간 비교

이번 측정은 본 논문의 기법을 적용하였을 때 실제로 발생하는 갱신의 시간을 서로 비교하여 감소된 갱신요청 데이터에 따라 갱신의 시간 차이를 확인하기 위한 것이다.

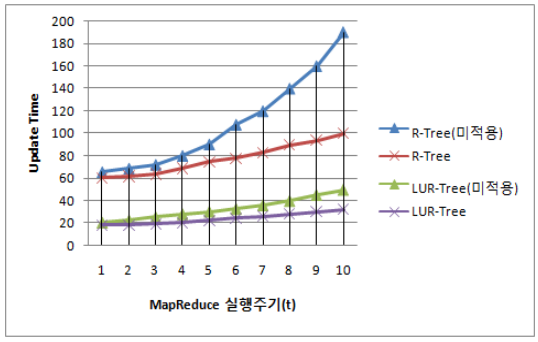

그림 16. MapReduce 실행 주기에 따른 R-tree와 LUR-tree에서의 갱신 시간 비교

그림 16은 갱신요청 빈도가 상인 상태에서 본 논문의 기법을 적용하였을 때 실제로 R-tree와 LUR-tree에서 갱신을 실행하였을 경우 발생하는 갱신 시간을 본 논문의 기법을 미 적용하였을 경우와 비교한 것이다. 갱신요청 빈도를 상으로 제한한 이유는 갱신요청 빈도가 낮은 경우 갱신 시간의 차이가 크지 않아 그 영향이 작기 때문이다. 그림 16을 보면 R-tree와 LUR-tree 모두 MapReduce 실행 주기가 커질수록 갱신 시간의 차가 커지는 것을 확인 할 수 있다. 특히 R-tree에서는 매우 큰 차이를 보이는

데 본 논문의 기법을 적용하지 않은 경우 전체 갱신요청 데이터가 늘어날수록 R-tree에서 발생하는 전체 갱신 시간은 매우 크게 증가하였으나 본 논문의 기법을 적용한 경우 갱신 시간이 증가하기는 하였으나 급격한 증가는 발생하지 않았다. 이와 동일하게 LUR-tree에서도 본 논문의 기법을 적용한 경우 갱신 시간이 본 논문의 기법을 적용하지 않은 것에 비해 크게 감소한 것을 볼 수 있다. 즉 MapReduce의 실행주기의 증가나 갱신요청 빈도의 증가 등 전체 갱신요청 데이터가 늘어나는 경우 갱신 시간은 크게 증가하나 본 논문의 기법을 적용하면 갱신 시간은 약간의 갱신 시간의 증가는 있으나 본 논문의 기법을 적용하지 않은 경우와 비교하여 많은 갱신 시간을 감소시키는 것을 확인 할 수 있다.

5. 결론 및 향후 연구

본 논문에서는 빅 데이터 크기의 이동객체에서 발생하는 갱신정보를 MapReduce와 시공간 데이터를 이용하여 갱신 횟수를 감소시킬 수 있는 기법을 제안하였다. 빅 데이터 크기의 이동객체에서 발생하는 갱신요청을 MapReduce를 이용하여 각각의 이동객체 별로 데이터들을 그룹화 하였고 갱신정보에 포함되어있는 시공간 정보를 이용하여 갱신 여부를 결정하도록 하여 전체 갱신 횟수를 줄였다.

기존의 기법으로는 처리하기가 어려운 빅 데이터 크기의 갱신 데이터를 하둡의 MapReduce를 이용하여 각각의 이동객체에서 발생된 갱신 데이터들끼리 그룹화하여 이후 처리 시 빠르게 갱신 여부를 판단할 수 있도록 하였다. 그리고 MapReduce를 통하여 각각의 이동객체 별로 그룹화된 갱신 데이터들을 각각의 이동객체가 가지고 있는 시공간 데이터들과 기존의 갱신데이터를 가지고 있는 갱신 데이터 해시 테이블의 시공간 데이터들을 서로 비교하여 갱신여부를 판단하여 실제 실행해야 하는 갱신 횟수를 줄이도록 하였다. 제안 기법은 기존의 인덱스기법의 변경이 필요하지 않고 기존의 기법의 앞부분에서 발생한 갱신 데이터를 가지고 최종 갱신 횟수를 감소시키므로 어느 인덱스에도 적용이 가능한 유연함을 가질 수 있다.

성능 평가에서 갱신 처리 시 전체 데이터가 많을수록 감소되는 갱신 감소 비율이 더욱 커짐을 확인하였고 데이터가 적은 경우에도 MapReduce주기가 긴 경우 갱신 감소 비율이 증가됨을 확인 하였다. 즉 빅 데이터 크기의 갱신 발생 시 더 높은 효율을 볼 수 있고 데이터의 크기가 커질수록 갱신 감소 효율이 높아지는 것을 알 수 있다. 또한 처리해야 할 갱신 데이터의 크기가 커지는 경우 기존 기법은 갱신 처리 시간이 크게 증가하나 본 기법에서는 전체 갱신 처리 시 비용이 낮은 비율로 증가하는 것을 확인 할 수 있다.

향후 연구로 본 논문에서는 MaxUpdateTime과 MaxUpdateRange 그리고 MaxUpdateTimeOut값을 일정한 값으로 한정하였다. 그러나 이 값들은 실험적인 값들이고 이 값의 변화에 따라 전체 성능이 가변적일 수 있기 때문에 각 상황에 따른 최적 값에 대한 연구가 필요하다. 그리고 MapReduce의 빠른 처리를 위하여 시공간 데이터에 적합한 키 분산 기법에 대한 연구도 필요하다.

참 고 문 헌

[1] Apache Hadoop, http://hadoop.apache.org/.

[2] A. Guttman, 1984, “R-tree: a dynamic index structure for spatial searching”, Proc Of Intl Cong On Management of Data, ACM SIGMOD.

[3] Dongseop Kwon, Sangjun Lee, Sukho Lee, 2002, “Indexing the Current Positions of Moving Object using the Lazy Update R-tree”, IEEE MDM '02, pp. 113~120.

[4] Ferrari, L.; Mamei, M., 2011, “Discovering daily routines from Google Latitude with topic models”, In Proc of the Pervasive Computing and Communications Workshops (PERCOM Workshops), IEEE International Conference on.

[5] F. Bentley and C. Metcalf, 2008, “Location and activity sharing in everyday mobile communication”, CHI '08 extended abstracts on human factors in computing systems, Florence, Italy: ACM, pp. 2453-2462.

[6] Hadoop: Open source implementation of MapReduce, http://lucene. apache.org/hadoop/.

[7] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, D. Stott Parker, 2007, “Map-reduce-merge: simplified relational data processing on large clusters”, In Proc of the ACM SIGMOD international

conference on Management of data.
[8] J. Dean, S. Ghemawat, 2004, “MapReduce: Simplified Data Processing on Large Clusters”, In Proc of the 6th Symposium on Operating Systems Design and Implementation, San Francisco CA, Dec.
[9] Jeffrey Dean, Sanjay Ghemawat, 2010, “MapReduce: a flexible data processing tool”, In Proc of Magazine Communications of the ACM, Volume 53 Issue 1, January.
[10] L. Barkhuus et al., 2008, “From awareness to repartee: sharing location within social groups”, Proc. of the twenty-sixth annual SIGCHI conf. on human factors in computing systems, Florence, Italy: ACM, pp. 497-506.
[11] M. Lee, W. Hsu, C. Jensen, B. Cui, K. Teo, 2003, “Supporting Frequent Updates in R-tree: A bottom-Up Approach”, In Proc of the Intl Conf on Very Large Data Bases, 2003.
[12] Qiang Ma, Bin Yang, Weining Qian, Aoying Zhou, 2009, “Query processing of massive trajectory data based on mapreduce”, CloudDB, USA.
[13] S. Ghemawat, H. Gobioff, S. Leung. 2003, “The Google file system”, In Proc of ACM Symposium on Operating Systems Principles, Lake George, NY, Oct, pp 29 - 43.
[14] S. Weil, S. Brandt, E. Miller, D. Long, C. Maltzahn, 2006, “Ceph: A Scalable, High-Performance Distributed File System”. In Proc. of the 7th Symposium on Operating Systems Design and Implementation, Seattle, WA, November.
[15] The Apache Software Foundation, The Hadoop Distributed File System: Architecture and Design.
[16] Thomas Brinkhoff, 2000, “Generating Network-Based Moving Objects”, 12th International Conference on Scientific and Statistical Database Management Berlin, Germany, IEEE Computer Society Press, July pp. 26-28.
[17] Thomas Brinkhoff, 2002, “A Framework for Generating Network-Based Moving Objects”, Proc of GeoInformatica, Vol. 6, No. 2, pp. 153-180.
[18] 천종현, 정명호, 장용일, 오영환, 배해영, 2006, “UCN-트리: 제한된 망 구조 내의 이동체를 위한 통합 색인,” 한국공간정보시스템학회 논문지, 제8권, 제1호, pp. 37-57
[19] 김정현, 박순영, 장용일, 김호석, 배해영, 2005, “색인 구조 예측을 통한 이동체의 지연 다량 삽입 기법,” 한국공간정보시스템학회 논문지, 제7권, 제3호, pp. 3-134

논문접수 : 2012.02.29
수 정 일 : 1차 2012.04.16 / 2차 2012.04.23
심사완료 : 2012.04.27

최 용 권
2007년 인하대학교 컴퓨터공학부 졸업 (공학사)
2010년~현재 인하대학교 정보공학과 (석사과정)
관심분야는 분산 데이터베이스, 위치기반 서비스

백 성 하
2005년 인하대학교 컴퓨터공학부 졸업 (이학사)
2007년 인하대학교 컴퓨터 정보공학과 (공학석사)
2007년~현재 인하대학교 컴퓨터 정보공학과(박사과정)
관심분야는 데이터 스트림, 클러스터, 위치기반 서비스

김 경 배

1992년 인하대학교 전자계산공학과 (공학사)
1994년 인하대학교 전자계산공학과 (공학석사)
2000년 인하대학교 전자계산공학과 (공학박사)
2000년~2004년 한국전자통신연구원 (선임연구원)
2004년~현재 서원대학교 컴퓨터교육과 조교수
관심분야는 이동실시간 데이터베이스, 스토리지 시스템

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
1982년~현재 인하대학교 컴퓨터공학부 교수
1999년~현재 지능형 GIS연구센터 센터장
2000년~현재 중국 중경우전대학교 대학원 명예교수
2004년~2006년 인하대학교 정보통신대학원 원장
2006년~2009년 인하대학교 대학원장
관심분야는 분산 데이터베이스, 공간 데이터베이스, 지리정보 시스템, 멀티미디어 데이터베이스