그리드 파일을 이용한 메인 메모리 이동체 색인의 설계

이승일°, 안경환, 홍봉희
·부산대학교 컴퓨터공학과
{cups2000°, khan, bhhong}@pusan.ac.kr

The Design of Moving Objects Index Using the Grid File

Seoungil Lee°, Kyounghwan Ahn, Bonghee Hong
Dept. of Computer Engineering, Pusan National University

요 약

이동체란 시간 변화에 따라 연속적으로 위치가 변화하는 객체를 말한다. 이러한 이동체는 기존의 공간 데이터와는 달리 이동체의 위치변경 보고에 따라 지속적인 갱신연산을 발생 시키는 특징을 가지고 있다.

이동체의 현재 색인에 있어 트리 기반의 색인 구조는 빈번한 갱신에 대한 색인의 변경 비용이 크므로 부적합하다. 확장 해쉬 기반의 그리드 파일 색인은 갱신연산의 비용이 적고, 그리드를 동적으로 구성하므로 공간 활용도가 높으며 영역 질의에 우수한 장점을 가지고 있다. 그러나 빈번한 갱신연산으로 인하여 해당 색인의 반복적인 분할/합병 비용을 발생시키는 문제점을 가지고 있다.

이 논문에서는 메인 메모리 기반의 그리드 파일을 구성하고, 빈번한 갱신연산에 따른 색인의 반복적인 분할/합병 비용을 제거하기 위한 합병정책을 제안한다. 특히 시간에 따라 해당 이동체의 수가 지역에 따라 변화하므로 이동체의 이동을 고려한 합병정책을 제안한다.

1. 서 론

무선 이동통신 기술의 발달과 더불어 GPS를 이용한 사용자의 위치정보를 제공해 줄 수 있는 각종 통신 기기가 보편화되면서 LBS(Location Based Service)와 관련된 서비스의 요구가 증대 되고 있다. 수많은 이동체의 위치정보는 주기적으로 서버에 보고 되고 서버에서는 이러한 이동체 정보를 저장하여 각각의 현재위치와 관련된 정보검색을 할 수 있도록 해준다. 그러므로 이동체의 현재위치와 관련된 서비스 제공을 위해서는 효율적인 이동체의 공간색인 구조가 필요하다.

트리 기반의 색인 중에 R-Tree색인 구조는 갱신연산에 대한 색인의 변경비용이 해쉬 기반의 색인에 비해 크다. 또한 Quad-Tree 및 KDB-Tree[5]의 경우 이동체가 집중된 지역에 대하여 급격한 성능 저하를 보이므로 부적합하다. 해쉬 기반의 고정 그리드 색인의 경우 갱신연산에 대한 색인의 변경 비용이 적고 데이터 분포가 정규분포인 경우 우수한 성능을 보이는 반면, 이동체가 집중된 데이터에 대하여 오버플로우를 발생시킴으로써 전체적인 색인의 검색성능 저하를 가져온다[2]. 확장 해쉬 기반인 그리드 파일의 경우 이러한 고정 그리드의 단점을 보완하여 오버플로우 발생시 분할을 통하여 그리드를 동적으로 구성함으로써 공간사용율을 높이고 영역질의에 우수한 성능을 나타낸다[2,3]. 그러나 잦은 갱신연산으로 디렉토리 및 데이터 버켓에 대한 반복적인 분할/합병을 발생시키므로 갱신연산의 비용이 증가시키는 단점을 가지고 있다.

이 논문에서는 메인 메모리 기반의 그리드 파일 이동체 색인구조와, 빈번한 갱신연산에 따른 반복적인 데이터 버켓의 분할/합병 비용을 줄이기 위한 합병정책을 제시 한다. 특히 시간에 따라 해당 버켓의 포함 이동체의 수가 변화하므로 가상합병을 통한 합병정책을 제시 한다.

이 논문의 구성은 다음과 같다. 2장에서는 이동체 색인과 관련된 관련연구와 문제점을 살펴보고 3장에서는 메인 메모리 기반의 이동체 색인구조를 제시하고 반복적인 분할/합병 비용을 줄이기 위한 방안으로 가상합병 방법과 Merge threshold의 동적 할당 방법에 대해 설명한 다음 구현 알고리즘을 제시한다. 마지막으로 4장에서는 결론과 향후 연구 과제를 제시 한다

2. 관련연구

기존의 저장 데이터는 정적인 반면 이동체 데이터는 지속적으로 변하는 동적인 데이터라고 할 수 있다. 그러므로 이동체의 위치 변경 보고 수에 따라 빈번한 갱신연산을 발생시킨다. 이동체의 양이 많을 경우 갱신연산은 점차 증가하게 되고 색인의 성능은 급속히 떨어진

다. 이러한 갱신연산의 비용을 줄이기 위한 방안으로 [1]은 전체 영역을 해쉬 기반의 작은 영역으로 나누고 각각의 영역에 대한 버켓을 할당한 다음, 같은 버켓 내의 이동에 대한 갱신연산은 수행하지 않고, 서로 다른 버켓으로 이동한 위치보고의 경우에 갱신연산을 수행함으로써 갱신연산의 횟수를 어느 정도 줄일 수 있었다. 하지만 이동체수가 점차 많아질 경우 갱신연산의 횟수는 지속적으로 증가 할 수 있다.

해쉬 기반의 고정 그리드는 영역을 일정한 간격으로 나누어 각각의 그리드에 해당 버켓을 할당하는 구조로 버켓 오버플로우가 발생할 경우 해당 버켓을 추가 할당하는 방식으로 색인 방법이 비교적 간단하다. 또한 이동체의 갱신연산 비용이 작고 데이터가 균등하게 분포된 경우 우수한 성능을 보인다. 그러나 비 균등분포 데이터의 경우 이동체가 밀집되어 있을수록 검색 성능이 급격히 떨어지게 된다. 이러한 단점을 보완한 확장 해쉬[4] 기반의 그리드 파일색인의 경우 밀집된 지역에 대하여 해당 그리드의 분할을 수행함으로써 공간 활용도를 높이고 기존의 고정 그리드의 장점을 유지할 수 있도록 하고 있다[2].

이동체 정보 저장에 있어 그리드 파일은 잦은 이동체의 위치변경 연산에 따라 그림 1에서와 같이 각각의 색인상태가 (a)처럼 분할 상태에서 (b)로 합병상태로 전환된 다음, 이동체가 다시 삽입될 경우 재분할을 하여 (a)로 이어지는 반복적인 색인의 분할/합병을 유발하게 되므로 해당 데이터 버켓의 할당 및 디렉토리 관리 비용이 증가 하게 되므로 결국 전체적인 갱신연산 처리 비용이 증가하는 문제점이 발생한다. 이것은 기존의 합병정책이 시간에 따른 이동체의 잦은 위치변경을 고려하지 못하고 포함 데이터의 수량을 기준으로 합병 버켓의 포함 데이터 크기(Merge threshold)를 기준으로 합병을 수행함으로써 합병 후 이동체가 삽입될 경우 재분할을 발생시키게 되므로 분할/합병 처리 비용을 증가 시키는 문제점이 있다. 또한 일정 시간 동안 이동체의 위치 변경이 없는 지역은 합병을 해 주어야 하지만 (c)처럼 합병시 합병조건으로 버켓의 Merge threshold가 70~75%를 만족하는 범위 이하에서 합병을 수행하므로 80%범위로는 합병이 이루어지지 않는 경우가 발생 한다. 이러한 경우 전체적인 데이터 버켓의 활용도를 떨어뜨리는 문제점을 가지고 있다. 따라서 이동체의 이동에 따른 데이터 버켓의 재분할을 고려한 합병정책과 이동체의 이동이 적은 지역은 합병을 앞당길 필요가 있다.

여기서는 이동체색인의 구조와 가상합병을 통한 합병 연기정책을 적용할 경우 버켓의 재분할에 따른 분할 비용을 줄이고, 해당 버켓의 Merge threshold값을 재분할의 크기에 따라 동적으로 적용함으로써 분할/합병의 시점을 조절하는 방안을 제안 하고자 한다.

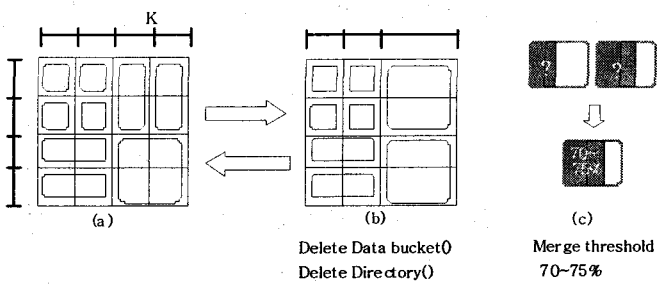
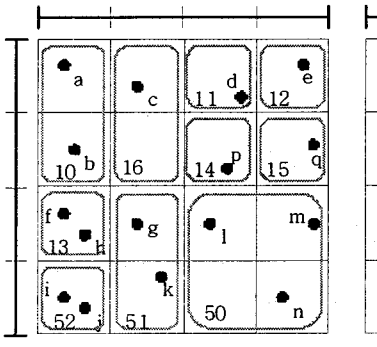


그림 1. 그리드 파일의 합병정책

3. 이동체 색인의 구조

이동체를 2차원 좌표 상의 점으로 표현하고 주기적인 위치 보고를 통해 서버의 현재위치 색인 정보를 변경하게 된다. 이때 위치보고 시점의 시간정보와 해당 위치 좌표가 저장된다. 실시간의 빠른 현재위치 색인 처리를 위해 메인 메모리 기반의 그리드 파일을 이용한 색인을 구성 한다. 그림 2의 a에서 보는 바와 같이 선형눈금자와 디렉토리 및 이동체를 저장하는 데이터 버켓으로 이루어져 있으며, b와 같이 디렉토리는 해당 데이터 버켓 주소를 가지고 있다. 또한 d처럼 이동체의 속도 및 이동체 타입 등 상세 정보를 제공하는 테이블로 구성이 되어 있다. c의 데이터 버켓 구조는 재분할과 관련된 분할횟수 및 가상합병의 상태 정보를 포함 하고 있다. 이것은 이동체의 이동 따른 색인의 분할/합병 비용을 줄이기 위한 정보로 구성 되어져 있다.



a. 데이터 버켓 경계

10	16	11	12
10	16	14	15
13	51	50	50
52	51	50	50

b. 디렉토리 구성 및 연결 버켓 ID

Bucket_ID	Object	Virtual mode	Time	Split Count
10	a. b	F	T1	1
11	d	T	T4	3
12	e	T	T5	2
13	f. h	T	T3	1
....	....	...		...
50	l. m. n	F	T8	1

c. 데이터 버켓 구조

Object_ID	Position	Speed	type
a	x1, y1	70	C
b	x2, y2	50	C
c	x3, y3	55	S
d	x4, y4	40	C
...	...	...	...
n	X8, y8	60	S

d. 이동객체의 상세 정보

그림 2. 그리드 파일을 이용한 이동체 색인 구성도

3.1 합병정책

이동체를 차량이나 사람으로 본다면 주로 도로 중심으로 이동을 하고 교차로나 사람이 많이 모이는 장소에 밀집되는 특징을 가지고 있다. 따라서 이동체의 이동 영역을 일정 크기로 나누어 볼 때 이동체가 자주 이동하는 영역과 자주 이동하지 않는 영역이 존재한다. 따라서 시간이 지남에 따라 이동체의 분포가 변하게 되고 이동체가 많은 지역은 이동체의 빈번한 삽입/삭제 연산을 해당 데이터 버켓에 발생시키고 포함 이동체의 수량 또한 가변적으로 변한다. 이러한 이동체의 특성은 색인의 분할과 합병에 영향을 주게 된다.

색인의 반복적인 분할/합병이 발생할 경우 메인 메모리 기반의 이동체 데이터베이스에서 데이터 버켓과 관련한 잦은 메모리 할당과 제거 비용이 증가하게 되므로 전체적인 성능 저하를 가져온다. 따라서 데이터 버켓의 재분할 비용을 감소시키는 방안으로 가상합병을 통한 합병연기 정책을 제안한다.

3.1.1. 가상합병과 가상분할빈도에 따른 합병

합병은 디렉토리 합병과 데이터 버켓의 합병으로 나누어진다. 디렉토리 합병은 해당 2차원 배열의 행이나 열을 제거 하는 것으로 디렉토리와 관련한 선형눈금자의 처리와 함께 진행이 된다. 데이터 버켓의 합병은 우선 합병가능 후보를 추출한 다음, 합병기준 축 선정 및 Merge threshold에 따라 합병을 하게 된다. 합병가능 후보 추출 방법은 Buddy System과 Neighbor System으로 나누어진다. Neighbor System은 합병 후보가 여러 가지로 존재할 수 있지만 잦은 분할/합병이 발생할 경우 합병 후보 선택에 대한 알고리즘이 복잡해지고, 합병 처리 비용이 점차 증가 하는 단점이 있다. 따라서 Buddy System 방식의 합병후보 선택을 통한 합병 처리를 수행하는 방법이 효과적이다[2].

여기서 제안하는 가상합병(Virtual Merge)은 물리적인 디렉토리 분할 및 데이터 버켓에 대한 분할 상태를 그대로 유지 하면서 논리적으로는 합병을 수행하고 물리적인 합병을 연기하는 방법을 말한다. 그림 3에서 보는 바와 같이 데이터 버켓 B0가 Step1에서 B1과 B2로 분할을 한 다음 시간이 지남에 따라 이동체의 삽입/삭제가 이루어져 Step2와 같이 합병을 하게 될 경우 가상합병상태로 전환을 하여 물리적인 합병을 연기 하게 된다. 가상합병 상태에서 일정시간(T) 이내에 오버플로우(overflow)로 인하여 재분할이 발생하게 되면 Step3과 같이 이전의 물리 적인 분할 상태로 전환함으로써 재분할에 따른 처리비용을 줄이도록 한다. 그러나

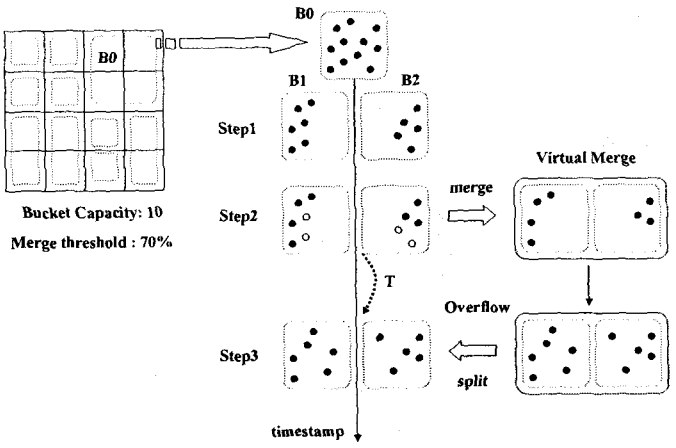


그림 3 가상합병과 재분할

가상합병상태의 시간 (T)가 길어질수록 버켓의 활용도가 점차 떨어지게 됨으로 이러한 경우 합병을 수행하여 주어야 한다. 우선 가상합병 상태에서 합병을 수행하기 위해서는 시간에 따른 가상분할빈도(Virtual Split Frequency)를 측정할 필요가 있다.

가상분할빈도는 이전의 분할을 카운트하여 시간에 대한 함수로 나타낸 값으로 가상합병 상태에서 버켓의 분할이 발생할 경우 분할횟수를 카운트하여 나타 낼 수 있다.

$$\text{가상 분할 빈도}(SF) = \frac{\text{Split\_Count}}{T1+T2}$$

그림 3의 Step1과 Step3에서 분할에 따른 카운터를 수행함으로써 가상합병이후 이동체의 삽입/삭제시 가상분할빈도를 연산 할 수 있다. 그림 4는 재분할이 반복될 경우의 가상분할빈도 연산방법으로 이전의 분할횟수를 시간 T1과 T2의 합으로 나누어 나타낼 수 있다.

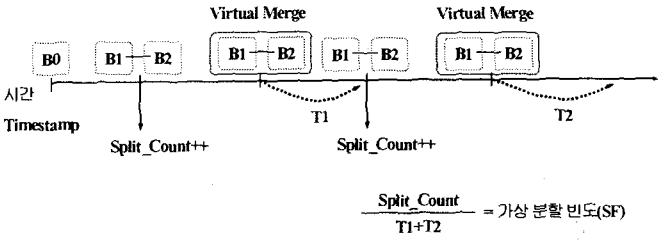


그림 4 재분할에 따른 가상분할빈도연산

합병은 그림 5에서 보는 바와 같이 시간이 지남에 따라 가상분할빈도의 크기는 점차 줄어들므로 기준분할빈도(Basic Frequency)값 보다 작아지는 시점의 삽입/삭제에서 합병을 수행한다. 기준분할빈도는 초기에 분할상태를 허용하는 값으로 추가적인 실험 평가가 필요하다. 예를 들어 초기 기준분할빈도를 1분으로 잡는다면 1분 이내에 분할이 발생하지 않을 경우 합병을 수

행하게 된다. 즉, 기준분할빈도의 수치는 약 0.016 으로 볼 수 있다.

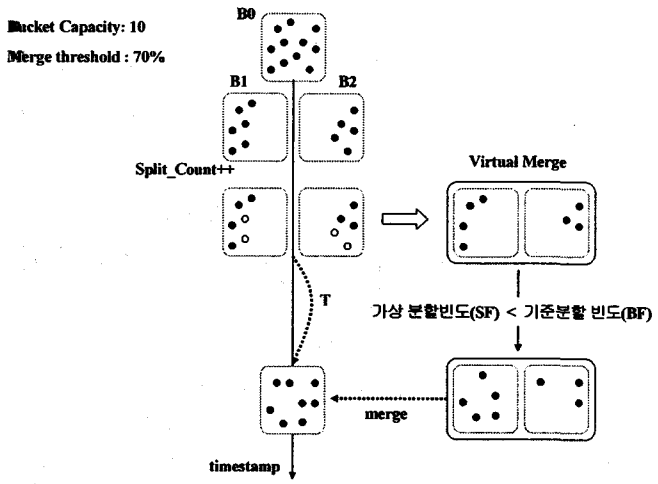


그림 5 가상분할빈도와 합병

3.1.2. Merge threshold의 조정

합병조건에 있어 합병결과 포함 데이터의 크기(Merge threshold)에 따라 합병을 수행하게 된다. 기존논문 [2]에서는 합병결과 해당 버켓의 포함 데이터(Merge threshold)의 비율을 70~75%로 유지 하는 것을 얘기하고 있다. 이것은 Merge threshold가 큰 경우 합병시점이 빨라지고, 합병이후 데이터 삽입시 재분할이 빨리 이루어지는 특징을 가지고 있기 때문에 분할/합병의 횟수가 증가하게 된다. 반대로 Merge threshold가 작은 경우 합병시점이 늦어지고, 합병이후 재분할의 시점 또한 늦어지게 된다. 그러나 버켓의 포함 이동체 수가 적어지므로 버켓의 활용도가 떨어지게 된다. 따라서 여기서는 가상합병상태에서 재분할이 발생할 경우 해당 버켓의 Merge threshold를 감소시킴으로서 다음 가상합병 시점을 연기할 수 있도록 한다. 즉 가상합병 시점이 연기됨으로서 전체적인 분할/합병의 주기가 길어지고, 가상분할빈도는 떨어지게 된다. 반대로 가상합병 상태에서 합병이 발생할 경우 Merge threshold를 증가시킴으로서 이동체가 많이 이동하지 않는 지역은 합병 시점을 앞당기도록 한다. 대신 가상분할빈도는 높아지고 해당 버켓의 포함 이동체 수는 증가 하게 된다. 그림 6에서는 가상합병 상태에서 오버플로우로 인한 재분할과 합병처리에 대한 Merge threshold의 크기를 증감 시키고 있다. Merge threshold의 증감 크기는 데이터 버켓에 대한 5% ~ 10% 내의 크기로 증감을 하고 최소 데이터 버켓의 50%, 최고 90%까지의 범위를 갖도록 한다.

두개의 합병 버켓중 하나가 가상합병 모드인 경우 그림 7의 (a)에서 보는 바와 같이 가상합병모드의 버켓이 이웃한 버켓과 합병이 발생한 경우 합병 시점을 연기하게 된다. 즉, A2와 A3 버켓의 가상합병이 오버플로우로 인한 재분할이 발생할 수 있으므로 A2와 A3의 합병이 발생하는 시점에서 A0와 합병을 수행하도록 한다. 또한 Merge threshold의 조정으로 인한 합병 버켓 각각의 Merge threshold가 서로 다를 경우 낮은 Merge threshold의 값에 따라 합병을 수행 하도록 한다. 그림 7의 (b)와 같은 경우 70%를 적용하여 가상합병을 수행하게 된다.

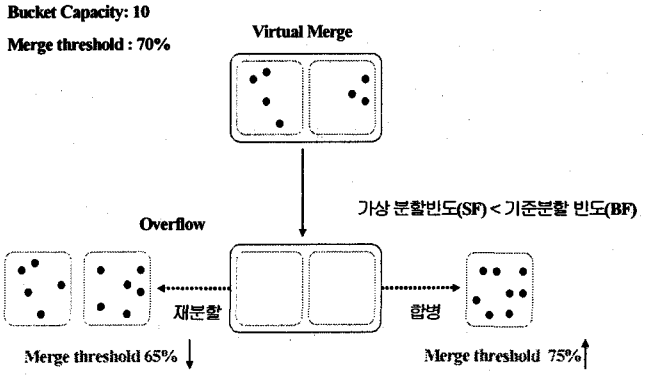


그림 6 Merge threshold의 조정

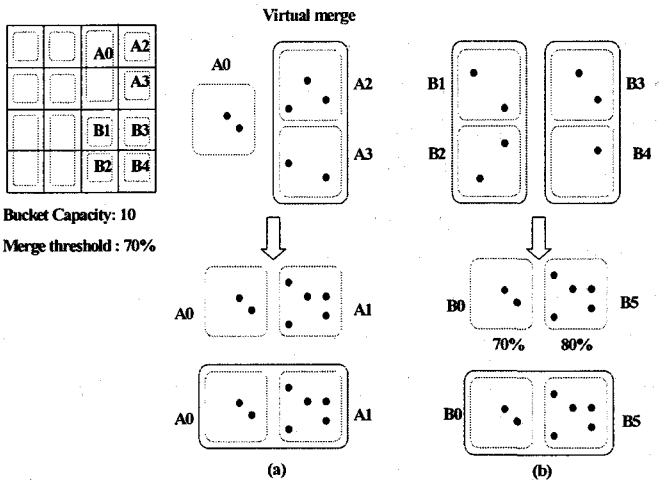


그림 7 가상합병의 합병 및 Merge threshold의 선택

3.2 알고리즘

이동체 삽입과 알고리즘에 대해 살펴보면 아래와 같다. 우선 이동체를 2차원 상의 점으로 보고, 해당 위치를 일정 시간 주기로 보고를 한다면 이동객체 ID와 해장 좌표(x,y)값을 전달 받아 삽입을 하게 된다.

```
Insert(m:moving object[ID, x, y])
begin
  // Find Grid Cell Position
  Cell = Search_grid_directory(m)
```

```
Add m
// Check data bucket overflow and virtual mode
if(Cell bucket is overflow ) then
   if( mode is virtual) then
    Re_Split_directory_and_bucket(Cell)
   else
    Split_directory_and_bucket(Cell)
    mode_is_virtual
   end_if
end_if

// Split count calculation : SF
// BF : Iinitialization Value BF(BF)
if( SF < BF and virtual mode) then
   // Data bucket and Directory merge
   merge(Cell)
end_if
end
```

삽입될 이동체의 그리드 셀의 위치를 검색한 다음 해당 버켓에 삽입을 하고 버켓의 오버플로우를 확인한다. 오버플로우 발생시 가상합병 모드상태인지 점검한 다음 가상합병상태일 경우 재분할을 수행하고 아닌 경우 일반분할을 수행 한다. 만일 오버플로우가 발생하지 않고 가상합병 모드인 경우 삽입시점의 가상분할빈도(SF) 크기를 연산하여 기준분할빈도(BF)보다 작은 경우 합병을 수행하도록 한다.

일반 분할인 경우 해당 디렉토리 및 버켓 분할을 수행하고 분할카운트 모드로 전환 하여 이후부터의 분할 카운트를 수행한다.

```
Split(cell:[bucket ID, cell_x, cell_y])
begin
 axis = split_Direction(cell)
 Add_Linear_Array(axis)
 Split_Directory(cell, axis)
 Split_ Data_Bucket(cell)
  virtual merge mode is true
 Split count++
end
```

재분할 알고리즘의 경우 현재의 가상합병상태를 해제하고 이전의 물리적인 분할 정보를 이용하여 분할상태로 전환 한다. 또한 재분할에 따른 Merge threshold를 감소 시켜 다음번 합병 시점이 연기 되도록 한다.

가상합병모드인 경우 이동체 데이터의 삽입/삭제시 가상분할빈도를 연산하여 기준분할빈도 보다 작은 경우 합병을 수행하게 된다. 또는 삭제에 따른 합병조건 검사를 수행하게 된다.

```
Re_Split(cell:[bucket ID, cell_x, cell_y])
begin
  virtual merge mode is false
 Decrease_Merge_threshold(k%)
 Split count++
end
```

합병의 경우 가상합병과 데이터 버켓의 합병으로 나누어진다. 가상합병의 경우 실제 물리적인 합병은 수행하지 않고 논리적인 가상합병모드 체크를 수행한 다음 가상합병 수행 시점의 시간을 기록한다. 이와 반대로 일정 시간(T)내에 오버플로우로 인한 재분할이 발생하지 않을 경우 가상분할빈도를 연산하여 기준분할빈도보다 낮아지는 시점의 경우 디렉토리 및 버켓 합병을 수행하게 된다.

```
Delete(m:moving object[ID, x, y])
begin
  // Find Grid Cell Position
 Cell = Search_grid_directory(m)
 delete m
 if(virtual merge mode) then
  // Split count calculation : SF
  if( SF < BF) then
    // Data bucket and Directory merge
    merge(Cell)
  end_if
 else if(merge condition) then
   Virtual_merge(Cell)
 end_if
 end_if
end
```

합병의 경우 Merge threshold 값을 증가 시켜 합병 시점을 앞당기도록 하고 전체적인 가상합병에 따른 재분할 카운트를 초기화한다.

```
Merge(cell:[bucket ID, cell_x, cell_y])
begin
  axis = Merge_Direction(cell)
  Delete_ Linear_Array(axis)
  Merge _Directory(cell)
```

```
Merge _Data_ Bucket(cell)
Initialization_split_count(0 )
Increase_Merge_threshold(k%)
Count mode is not Virtual
end

Virtual_ merge(cell:[bucket ID, cell_x, cell_y])
begin
    cell [bucketed, cell_x, cell_y]
   virtual merge mode is true
   Record_Time(T)
end
```

4. 결 론

이 논문에서는 메인 메모리 기반의 그리드 파일을 이용한 이동체 현재위치 색인 구조를 제시 하였으며, 빈번한 이동체의 위치 변경 보고에 대한 처리 비용을 줄이는 방안으로 가상합병을 통한 재분할을 카운트함으로써 합병연기 와 함께 재분할 비용을 줄이는 방안에 대하여 제시 하였다. 또한 재분할의 빈도에 따라 Merge threshold를 조정하는 방법을 적용함으로써 합병시점을 조절 하는 방법에 대하여 제시하였다.

향후 연구로 앞에서 제시한 현재위치 색인에 대한 구현 및 실험 평가를 통한 적정 기준분할빈도를 측정하고, 해쉬 기반의 색인 방법을 이용한 과거, 현재, 미래에 대한 이동체 색인구성 방법에 대하여 연구를 진행할 계획이다.

참고 문헌

[1]Zhexuan Song, Nick Roussopoulos "Hashing Moving Object," University of Maryland, 2000

[2]R.J. ENBODY, H.C.DU, "Dynamic Hash Schemes," ACM Computing Surveys, Vol.20, No.2, 1988

[3]J. Nievergelt, H hinterberger, and K. Sevcik, "The Grid File: An Adaptable, Symmetric Multikey File Structure," ACM Trans. Database Syst,vol.9, No.1, 1984, pp. 38-71.

[4]FAGIN, R., NIEVERGELT, J., PIPPENGER, N., AND STRONG, H. R. "Extendible Hashing-A FastAccess Method for Dynamic Files," ACM Trans. Database Syst, vol.4, No.3, 1979, pp. 315-344

[5]J.T. Robinson: "The K-D-B-tree: a search structure for large multidimensional dynamic indexes", Proc. ACM SIGMOD Int. Conf. on Management of Data 10-18, 1981

[6]Tobin J. Lehman, Michael J. Carey, "A study of Index Structures for Main Memory Database Management Systems", Kyoto, 1986