시간지원 데이타베이스에서 다차원 시간 집계 연산의 효율적인 처리 기법

(On Efficient Processing of Multidimensional Temporal Aggregates in Temporal Databases)

강 성 탁 † 정 연 돈 †† 김 명 호 †††

(Sung Tak Kang) (Yon Dohn Chung) (Myoung Ho Kim)

요 약 시간지원 데이타베이스 시스템은 자료의 과거 및 현재, 그리고 미래의 상태까지 관리함으로써, 사용자에게 시간에 따라 변화하는 자료에 대한 저장 및 질의 수단을 제공한다. 시간지원 데이타베이스에서의 집계 연산은 집계 연산과 질의에 시간 애트리뷰트를 고려하므로 기존의 집계 연산과는 큰 차이가 있다. 본 논문에서는 다차원 시간 집계 연산에 초점을 둔다. 다차원 시간 집계 연산은 시간 애트리뷰트 뿐만 아니라 하나 이상의 일반 애트리뷰트까지 고려한 시간 집계 연산으로 이력 데이타 웨어 하우스, 전화 기록 관리(CDR) 등에 유용하다. 본 논문에서는 다차원 시간 집계 연산을 효율적으로 처리하기 위한 자료 구조인 PTA-tree를 제안하고, 이를 이용한 시간 집계 처리 기법을 제안한다. 또한 본 논문에서는 제안된 PTA-tree를 이용한 기법과 기존의 SB-tree를 확장한 기법의 성능을 최악 경우 분석과 실험을 통해 비교한다.

키워드 : 시간지원 데이타베이스, 시간 집계 연산

Abstract Temporal databases manage time-evolving data. They provide built-in supports for efficient recording and querying of temporal data. The temporal aggregate in temporal databases is an extension of the conventional aggregate to include time concept on the domain and range of aggregation. This paper focuses on multidimensional temporal aggregation. In a multidimensional temporal aggregate, we use one or more general attributes as well as a time attribute on the range of aggregation, thus it is a useful operation for historical data warehouse, Call Data Records(CDR), etc. In this paper, we propose a structure for multidimensional temporal aggregation, called PTA-tree, and an aggregate processing method based on the PTA-tree. Through analyses and performance experiments, we also compare the PTA-tree with the simple extension of SB-tree that was proposed for temporal aggregation.

Key words : temporal database, temporal aggregation

1. 서 론

일반적인 데이타베이스는 현실 세계에서 발생한 사건에 대하여 가장 최근의 상태만을 반영한다. 따라서, 과거의 자료가 필요한 경우 이에 대한 별도의 처리 방법을 필요로 하며, 질의 작성이 복잡해지게 된다. 이에 반하여 시간지원 데이타베이스(temporal database)는 자료의 과거 상태와 현재 상태, 그리고 미래의 상태까지도 관리한다. 즉, 시간지원 데이타베이스는 사용자에게 시간에 따라 변화하는 자료에 대한 저장 수단 및 질의 방법을 제공한다.

시간지원 데이타베이스는 의사 결정 시스템 등에서 필수적인 경향 분석과 예측, 컴퓨터 응용 디자인에서의 버전 관리, 의료 기록 관리, 비디오 데이타 관리 등과 같이 관리되는 자료의 시간적 특성이 중요시 되는 모든 분야에 폭 넓게 응용될 수 있다. 또한 시간지원 데이타베이스는 최근 중요성이 부각되고 있는 데이타 웨어하

· 본 연구는 한국과학재단 특정기초연구(과제번R01-1999-0024) 지원으로 수행되었음.

† 비 회 원 : 한국과학기술원 전산학전공
stkang@dbserver.kaist.ac.kr
†† 비 회 원 : 한국과학기술원 전산학전공 교수
ydchung@dbserver.kaist.ac.kr
††† 종신회원 : 한국과학기술원 전산학전공 교수
mhkim@dbserver.kaist.ac.kr

논문접수 : 2001년 9월 7일
심사완료 : 2002년 9월 16일

우스(data warehouse)에서의 자료 이력 관리에도 효과적으로 이용될 수 있다.

시간지원 데이타베이스 분야에 대한 초기 연구는 주로 자료 모델링 기법과 질의 언어 등과 같은 개념적인 문제들에 초점을 맞추었으며[1], 색인 기법과 질의 처리 방법, 저장 구조 등과 같은 구현과 관련된 문제들에도 많은 연구가 수행되어 왔다[2, 3, 4, 5]. 최근에는 기존의 데이타베이스 표준 질의 언어인 SQL에 시간 개념을 추가하여 확장한 TSQL에 대한 연구와 더불어, 시간 개념의 추가로 인해 의미가 확장되는 연산자 및 새로이 정의되는 연산자와 이에 대한 처리 기법에 관련된 연구 등이 수행되었다[5, 6, 7, 8, 9, 10, 11].

질의 연산자 중에서 집계 연산(*aggregation*)은 릴레이션 전체 혹은 그 일부를 구성하는 튜플들에 적용되어 전체값을 계산하거나 대표값을 선택하는 연산으로 질의어의 중요한 요소이다. 집계 연산은 다양한 응용 분야에서 빈번히 사용되며 그 처리 비용 또한 매우 크기 때문에 질의 성능 평가의 많은 부분을 차지하기도 한다[12]. 시간 애트리뷰트를 포함하는 집계 연산(temporal aggregation)은 자료의 시간적 특성을 고려하지 않은 기존의 집계 연산과는 큰 차이가 있다. 따라서, 일반적인 데이타베이스에서 연구된 집계 연산 처리 기법을 이용해서 시간지원 데이타베이스의 집계 연산을 효율적으로 처리할 수 없다. 시간 집계 연산의 처리를 위한 기존 연구로서 aggregation-tree를 이용한 기법[7]이 제안되었으며 기존의 집계 처리 연산 기법에서 확장된 시간 집계 연산 처리 기법들보다 우수한 성능을 나타낸다고 알려져 있다[7]. 근래에는 디스크에 저장하기 위한 B-tree에 기반한 시간 집계 연산 처리 기법[10, 11]이 제안되었다.

의사 결정을 위한 데이타웨어 하우스 등에 필요한 다양한 집계 연산을 처리하기 위해서는 시간 구간 외에 일반 애트리뷰트까지 고려할 필요가 있다. [11]에서는 집계 연산의 조건에 일반 애트리뷰트까지 포함하여 시간 집계를 처리하는 방법을 제안하고 있다. 그러나, 이 방법은 시간이 단순 증가하는 처리 시간(transaction time) 차원만 고려함으로써 임의의 시간에 대해서는 사용할 수 없다. 또한, 조건에 일반 애트리뷰트를 하나 밖에 포함할 수 없다는 문제점이 있다.

본 논문에서는 시간 뿐만 아니라 일반 애트리뷰트까지 포함하는 다차원 시간 집계 연산을 효율적으로 처리하기 위하여 PTA-tree(Point Transformation-based Aggregation-tree)를 제안하고, 이를 기반으로 한 시간 집계 처리 기법을 제안한다. PTA-tree는 하나 이상의 일반 애트리뷰트와 시간 애트리뷰트를 포함하는 다차원 시간 집계 연산을 처리할 수 있다.

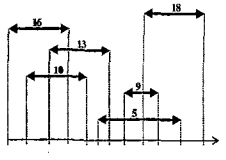
본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 시간 집계 연산의 의미를 규정하며, 기존의 시간 집계 처리 기법들을 살펴본다. 제 3장에서는 시간 집계 처리를 위한 새로운 자료 구조인 PTA-tree를 제안하며, 이를 이용한 시간 집계 처리 기법을 제안한다. 제 4장에서는 본 논문에서 제안하는 PTA-tree기법과 디스크 기반의 효율적인 시간 집계 처리기법으로 제안된 SB-tree[10]를 확장한 기법의 성능을 분석과 실험을 통해 비교하며, 마지막으로 5장에서는 연구 결과를 정리하고 향후 연구 방향에 대하여 기술한다.

2. 연구 배경

2.1 다차원 시간 집계 연산(Multidimensional Temporal Aggregation)

시간지원 데이타베이스에서는 각 튜플이 유효 시간 혹은 처리 시간을 나타내는 시간 애트리뷰트를 포함하고 있으므로, 집계 연산의 결과도 시간을 고려하도록 확장되어야 한다[13]. 즉, 각 튜플들의 시간 애트리뷰트에 의하여 전체 시간 구간이 분할되며 이렇게 분할된 각각의 시간 구간마다 집계 연산의 결과값이 달라질 수 있다. 따라서, 전체 시간 구간에 대하여 집계 연산 값이 변하지 않는 구간들과 그 구간에서의 집계 연산 값을 계산하여야 시간을 고려한 올바른 집계 연산 값을 구할 수 있다. 이와 같이 집계 연산 값이 변하지 않는 시간 구간들과 그 구간에서의 집계 연산 값을 계산하는 시간 지원 데이타베이스에서의 집계 연산을 '시간 집계'라 한다. 이 때, 집계 연산 값이 변하지 않는 시간 구간을 '불변 구간(constant interval)'이라 한다.

다음의 그림 1은 시간지원 집계 MAX와 COUNT의 예이다. 예제에서 MAX에 대한 불변 구간들은 <0,9>, <10, 15>, <16, 16>, <17, 19>, <20, 28> 등이며, COUNT에 대한 불변 구간들은 <0, 2>, <3, 5>, <6, 9>, <10, 11>, <12, 12> 등이다. 시간은 유한 시간(*discrete time*)으로 가정한다.



MAX

start	end	value
0	9	16
10	15	13
16	16	5
17	19	9
20	28	18
...	...	...

COUNT

start	end	value
0	2	1
3	5	2
6	9	3
10	11	2
12	12	1
...	...	...

(a) (b)

그림 1 시간지원 집계 MAX와 COUNT

[11]에서는 시간 구간 뿐만 아니라 하나의 일반 애트리뷰트를 포함하는 RTA (Range-Temporal Aggregation) 연산과 이의 처리 기법을 제안하였다. 이와 같이 시간 구간 외에 일반 애트리뷰트를 조건으로 하여 시간 집계를 처리하는 것은 의사 결정 등에 있어 유용하다. 또한, 매우 크기가 큰 이력 데이타웨어하우스(historical datawarehouse)에서는 특정 시간과 애트리뷰트 범위에 대한 시간 집계를 구하는 다차원 시간지원 집계 연산은 매우 유용하고 실용적인 연산이다[11].

본 연구에서는 시간 집계의 조건으로 시간 뿐만 아니라 하나 이상의 일반 애트리뷰트까지 포함하는 일반적인 형태의 다차원 시간 집계(Multidimensional Temporal Aggregation) 처리에 대해 알아본다. 그림 2는 다차원 시간 집계의 간단한 예를 표현한 것으로, 고용 이력 릴레이션으로부터 고용 기간이 100에서 200사이이며 A부서에서 일한 사원들의 정보로부터 봉급(salary)에 대한 시간 집계 SUM을 구하라는 것이다.

```
Compute temporal-aggregate SUM(salary)
From EmployeeHistory
Where 100≤employ_time≤200 and dept='A';
```

그림 2 다차원 시간 집계의 예

2.2 기존의 시간 집계 연산 처리 기법

보편적인 집계 연산을 처리하기 위한 단순한 방법으로는 Epstein의 알고리즘[14]이 있으며 시간 집계를 처리하기 위해 확장될 수 있다[15]. [15]에서는 연결 리스트를 이용하여 각 노드에 불변 구간과 집계 값을 저장하였다. 질의 조건을 만족하는 모든 튜플에 대해 연결 리스트에 저장된 불변 구간과 집계 값을 변경함으로써 최종적으로 불변 구간과 각 구간에서의 집계 값을 얻을 수 있다. 그러나, 연결 리스트는 많은 접근 시간을 필요로 하며, 또한 시간 구간이 긴 튜플의 경우 여러 불변 구간을 변경시켜야 하므로 비효율적이다. [5]에서 제안된 방법은 [15]와 비슷하게 연결 리스트를 사용하지만, 먼저 데이타를 모두 읽어들여 불변 구간을 먼저 확정하여 연결 리스트로 만든 후에 집계 값만을 변경하도록 하여 접근 시간을 줄였다. 그러나, 튜플들을 두 번 읽어들여야 하므로 기억장치 접근 회수가 많아지고, 시간 구간이 긴 하나의 튜플에 대해 여러 불변 구간의 집계 값을 변경해야 하는 단점이 존재한다.

Aggregation-tree[7]는 전체 시간 구간의 효율적인 분할에 의한 불변 구간의 계산과 집계 값의 계산을 위한 방법으로 제안되었다. Aggregation-tree를 이용한 시간 집계의 처리는 '트리 생성'과 '깊이 우선 탐색(depth first search)을 통한 결과 생성'의 두 단계로 구성된다. Aggregation-tree의 노드에는 시간 구간과 집계 대상 값이 저장되며, 자식 노드의 시간 구간은 부모 노드의 시간 구간을 분할한 결과이다. 즉, 같은 부모 노드를 가진 형제 노드들의 시간 구간은 상호 배타적이며, 형제 노드들의 시간 구간의 합(union)은 부모 노드의 시간 구간이 된다. 이러한 집계 트리 구조에서 리프 노드(leaf node)의 시간 구간은 하나의 불변 구간이 되며, 리프 노드의 집계 값은 이 불변 구간의 시간지원 집계 값을 구하는 과정에서 사용된다. Aggregation-tree에서 불변 구간의 실제 집계 값은 집계 처리의 두 번째 단계인 깊이 우선 탐색 과정을 통해 계산된다. 즉, 트리의 루트 노드에서 리프 노드까지의 경로에 있는 노드들의 값으로 구하고자 하는 집계에 적합한 계산을 수행한다. 예를 들어, SUM의 경우에는 루트에서 리프 노드까지의 노드 값을 더하며, MAX를 위해서는 루트에서 리프 노드까지의 집계 값 중에서 최대값을 구한다. Aggregation-tree는 균형화(balance)되어 있지 않으며 최악 경우 트리 생성에 $O(n^2)$의 시간 복잡도를 필요로 하는 문제점이 있다.

최근에는 시간 집계 결과를 효율적으로 디스크에 저장하고 관리하는 것에 대한 연구가 진행되고 있으며 결과로써 SB-tree와 MVSB-tree가 제안되었다[10, 11]. SB-tree는 Segment-tree와 B-tree에 기반한 방법으로써 저장 방법은 Aggregation-tree와 유사하다[9]. 내부 노드의 각 원소는 시간과 이 시간이 의미하는 시간 구간의 집계 값, 자식 노드로의 포인터를 가진다. B-tree에 기반하기 때문에 시간 값 만으로도 시간 구간을 표현할 수 있다. 부모 노드의 시간 구간은 자식 노드에 있는 원소들의 시간 구간의 합(union)이며, 자식 노드에 있는 원소들의 시간 구간은 상호 배타적이다. 집계 결과의 계산은 Aggregation-tree에서와 마찬가지로 루트 노드에서부터 리프 노드까지의 집계 값을 이용하게 된다.

MVSB-tree (Multi Version SB-tree) 는 SB-tree와 Multiversion B-tree에 기반한 것으로 시간 애트리뷰트 뿐만 아니라 일반 애트리뷰트까지 조건에 포함하는 더욱 일반적인 시간지원 집계 문제를 해결하기 위해 제안되었다[11]. MVSB-tree는 트리 군(forester)로써 여러 개의 SB-tree의 루트 노드를 가지며 각 SB-tree는 겹치지 않는 시간 구간을 저장한다. SB-tree의 루트 노드는 B-tree와 같은 구조체에 저장 된다. MVSB-tree는 계속적으로 증가하는 처리 시간을 가지는 데이타만 저장하며 시간 구간에 따라 SB-tree의 노

드 분할을 수행한다. MVSB-tree는 데이타의 삽입은 효율적이나 처리 시간만 고려하였으므로 유효 시간을 가진 데이타를 처리할 수 없다. 또한, 처리하고자 하는 일반적인 시간지원 집계도 하나의 일반 애트리뷰트만 고려하고 있으며 둘 이상의 애트리뷰트를 포함하는 다차원 시간 집계 문제로는 확장할 수 없다는 단점이 있다.

3. 제안하는 다차원 시간 집계 처리 기법

이 장에서는 효율적인 다차원 시간 집계 처리를 위한 새로운 접근 방법인 PTA-tree와 이를 이용한 집계 처리 방법에 대하여 설명한다.

3.1 기본 개념

다차원 시간 집계는 시간 구간 뿐만 아니라 여러 일반 애트리뷰트까지 고려하여 시간 집계를 처리하는 것이다. 기존의 시간 집계 처리를 확장하여 처리할 수도 있으나 기존 방법은 시간 구간 중심으로 데이타를 저장하므로 데이타의 시간 구간이 길 경우 중복해서 데이타가 저장되어야 한다. 최악의 경우 하나의 데이타에 대해 O(log N) 만큼의 데이타가 중복해서 저장되어야 하며 이로 인해 저장 공간을 많이 필요로 하는 문제점이 있다.

제안하는 방법은 일반 애트리뷰트를 중심으로 데이타를 저장하며, 이 때 조건의 대상이 될 일반 애트리뷰트를 공간상의 점으로 취급하여 처리하는 방법이다. 이 방법은 기존의 공간 데이타베이스에서 연구된 방법을 확장하여 이용할 수 있다. 그러나, 공간 상의 한 점에 대해 여러 데이타가 존재할 수 있으므로 핫 스팟(hot spot) 문제에 대한 해결 방안을 포함해야 한다.

공간 데이타를 저장하는 색인 방법은 크게 K-D-B-tree와 같이 축을 중심으로 공간을 분할하는 PAM(Point Access Method) 기법과 R-tree와 같이 데이타가 차지하는 공간을 MBR(Minimum Bounding Rectangle)로 표현하는 기법, 그리고 다양한 변환(transformation)을 이용하여 공간을 분할하는 기법이 있다[16]. PAM 방법은 삽입이 효율적이나 각 공간이 겹치지 않도록 분할하는 것으로 인해 데이타 페이지에 저장되는 최소 데이타 수를 보장할 수가 없으므로 성능 예측에 문제가 있다. R-tree와 같이 데이타가 차지하는 공간을 MBR로 표현하는 기법에서는, 여러 데이타가 동일한 하나의 점으로 중복될 경우 최소 데이타 수는 저장 가능한 수의 1/3을 보장할 수 있으나, 한 점에 해당하는 데이타만으로 하나의 페이지가 가득찰 경우에 대한 처리 기법이 없다. 이를 처리한다 하더라도 노드에 저장되는 영역 자체가 겹치기 때문에 같은 점에 해당하는 데이타가 여러 경로에 저장되어 디스크 낭비가 발생할 수 있다. 공간 변환을 이용한 기법은 Z-ordering, Hilbert-Curve 등의 공간 포화 곡선(Space Filling Curve)을 이용하여 다차원 공간을 1차원 공간이나 원래 데이타보다 적은 차원의 공간으로 변환하여 저장하며 기존의 색인 구조를 활용할 수 있다는 장점으로 인해 실제 데이타베이스에서 활용된다[17].

본 연구에서는 저장 공간을 가장 효율적으로 사용할 수 있도록 일반 애트리뷰트를 공간 포화 곡선을 사용하여 일차원 값으로 변환하고 이 값을 키로 하여 집계 데이타를 PTA-tree에 저장하도록 한다. 그리고, PTA-tree를 이용하여 다차원 시간 집계를 처리하는 방법을 제시하도록 한다. 변환 기법으로는 여러 기법이 있으나 단순한 열 순환(column-wise scan) 방법을 사용하도록 한다. 이 변환 기법에서는 다차원 애트리뷰트를 하나의 거대한 수로 취급한다. 예를 들어, 2개의 일반 애트리뷰트 {2,3}이 있고 각 애트리뷰트는 한자리라고 가정하면 이 일반 애트리뷰트로 부터 변환된 키 값은 23이 된다.

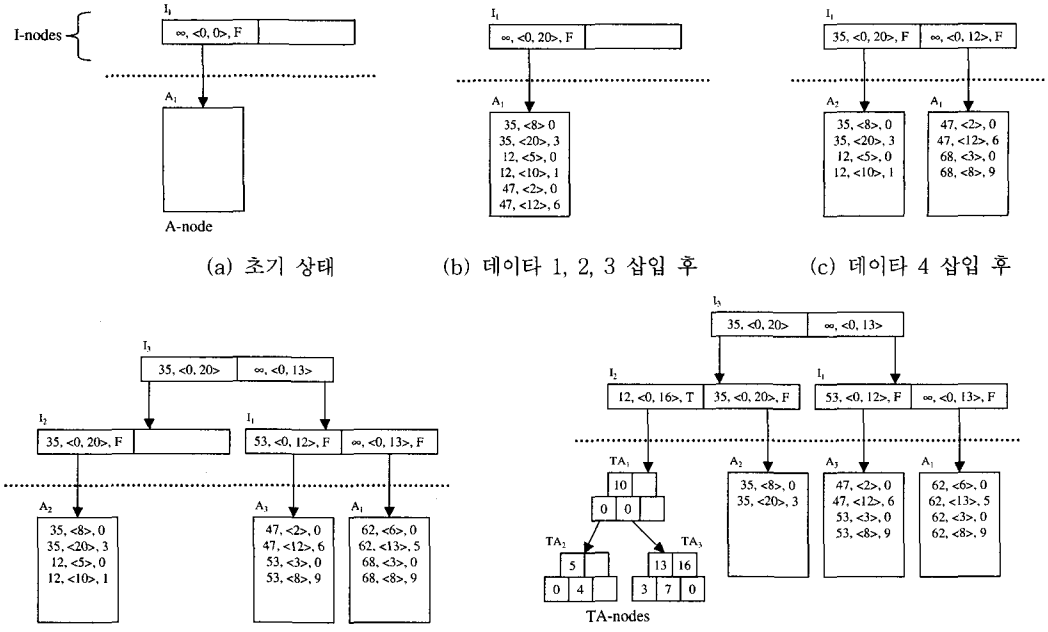
다차원 시간 집계를 위한 데이타에서는 하나의 점에 많은 데이타가 몰리는 핫 스팟 문제가 발생할 수 있으므로 이 문제에 대한 해결 방안이 필요하다. 핫 스팟 문제가 발생했을 경우에는 해당 데이타만 추출하여 시간 기반으로 저장하도록 하며 이렇게 저장된 데이타를 따로 가리킬 수 있도록 해야 한다. PTA-tree의 리프 노드에 저장되는 원소는 기본적으로는 키의 구간을 가리키므로 시간 기반으로 저장된 노드를 가리키는 원소는 따로 구별되도록 해야 한다. 이렇게 구간이 아니라 핫 스팟에 의해 따로 시간 구간에 의해 저장되는 노드를 가리키는 원소를 상승 원소(promoted element)라고 하며, 핫 스팟이 일어나서 상승 원소를 추가하는 것을 상승(promotion)이라고 한다.

3.2 PTA-tree의 구조

PTA-tree(Point Transformation-based Aggregation-tree)는 시간 집계의 처리에 필요한 시간 데이타의 정보를 일반 애트리뷰트로부터 변환된 키 기반으로 저장한다. 이렇게 일차원의 키를 활용함으로써 기존의 B^+-tree를 기반으로 한 구조에 집계 데이타를 저장할 수 있다. 따라서, 균형화 되어 있고 효율적인 생성과 유지가 가능하다. PTA-tree의 전체적인 형태는 그림 3과 같다.

PTA-tree는 집계 데이타를 저장하는 부분과 색인을 하는 두 부분으로 구성된다. 집계 데이타는 A-node나 TA-node에 저장된다. 다음은 각 노드에 대한 간략한 설명이다.

- A-node (Aggregate node) : 일반 애트리뷰트를 변환한 키 값과 시간을 포함한 집계 데이타를 저장한다. 즉, 일반 애트리뷰트로 변환된 키 K와 시간



(a) 초기 상태 (b) 데이타 1, 2, 3 삽입 후 (c) 데이타 4 삽입 후

(d) 데이타 5, 6 삽입 후 (e) 데이타 7, 8 삽입 후

그림 3 PTA-tree 생성 과정

구간을 위한 시간값 T, 집계 값을 위한 V를 저장한다. A-node에서 같은 키 값을 가지는 집계 데이타는 불변 구간을 구성하도록 저장되며, T는 각 불변 구간의 시작 혹은 끝 시간이 될 수 있으나 여기서는 끝 시간으로 한다.

- TA-node (Time-based Aggregate node) : 동일한 일반 애트리뷰트에 대한 집계 데이타를 저장하는 노드로서 시간 구간에 대해서만 저장한다. 즉, 시간 구간과 각 구간에서의 집계 값을 저장하며 B-tree와 유사한 트리 형태를 가진다.
- I-node (Indexing node) : 변환된 키 값을 기반으로 색인을 하는 부분이며 B^+-tree에 기반한 구조를 가진다. 일반 애트리뷰트를 변환한 키 값과 자식 노드에 있는 원소들의 시간 구간의 합(union)을 저장한다. 최하단 노드의 각 원소에서는 가리키는 자식 노드가 A-node인지 TA-node인지 구분하기 위한 Flag를 가진다. Flag 값이 F인 원소는 키 구간에 해당하는 데이타를 저장하는 A-node를 가리키며 Flag 값이 T인 원소는 TA-node를 가리키며 키 값은 키 구간이 아니라 TA-node에 저장된 시간 구간의 키 값을 저장한다.

그림 3은 PTA-tree의 예로써 표 1에 나타난 집계 데이타로부터 PTA-tree를 생성하는 과정을 나타낸 것이다. 그림 3에서 I-node는 최대 2개의 원소를, A-node는 최대 6개의 원소를 가질 수 있다고 가정하였으며, TA-node는 시간 값을 최대 2개까지 저장할 수 있다고 가정하였다. 또한, 시간의 최소 값은 0으로 가정하였다.

그림 3(a)는 PTA-tree 의 초기 상태를 나타낸다. 초기 상태는 하나의 I-node와 하나의 A-node로 구성되며 I-node에는 그 A-node를 가리키는 최대 키 값을 가지는 원소가 존재한다. 그림 3(b)는 표 2의 데이타 1, 2, 3을 추가한 후의 모습이다. 각 데이타는 동일한 키에 대해서 시간 구간을 분할하도록 삽입된다. 그림 3(c)는

표 1 집계 연산 SUM을 위한 다차원 집계 데이타

No	일반 애트리뷰트	시간 구간	값	키
1	{3, 5}	<9, 20>	3	35
2	{1, 2}	<6, 10>	1	12
3	{4, 7}	<3, 12>	6	47
4	{6, 8}	<4, 8>	9	68
5	{5, 3}	<10, 15>	20	53
6	{6, 2}	<7, 13>	5	62
7	{1, 2}	<6, 13>	3	12
8	{1, 2}	<14, 16>	7	12

3(b)에 데이타 4를 추가한 후의 모습으로 노드 A_1에 저장될 공간이 없으므로 분할되었으며 새로운 노드 A_2가 추가되었으며 A_2를 가리키는 원소가 I_1에 추가되었다. (d)는 데이타 5, 6이 삽입된 후의 모습으로 A-node 분할 뿐만 아니라 I-node 분할까지 발생하였다. 새로운 루트 I_3에는 분할된 I_1, I_2를 가리키는 원소들이 저장되었다. (e)는 데이타 7, 8이 삽입된 후의 모습으로 원소 상승이 발생된 경우이다. 데이타 7, 8은 A_2에 삽입되며 충분한 공간이 없으므로 분할이 발생한다. 이때, A_2에 저장된 원소들 중 키 값이 12인 원소들이 A-node에 저장 가능한 원소 수의 50% 이상을 차지하므로 이들을 추출하여 TA-node로 재구성하게 되며 최하단 I-node 에는 이렇게 생성된 TA-node를 위한 상승 원소가 추가된다. (e)에서 I_3에 추가된 원소에서 Flag의 값은 T로 상승 원소임을 나타내며 TA-node 들의 루트인 TA_1 을 가리킨다.

I-node는 B$^+$-tree에 기반한 색인으로 조건의 대상인 일반 애트리뷰트를 변환한 값을 키로 한다. I-node는 실제 집계 데이타를 TA-node와 A-node에 효율적으로 접근하기 위해 사용된다. 기본적인 특성은 B$^+$-tree와 유사하나 시간 구간을 포함하도록 확장되었으며 내부 노드와 최하단 노드의 형태가 다르다. 또한, 가장 오른쪽 경로에 해당하는 노드에는 가장 큰 키 값인 ∞를 저장하는 원소들이 존재한다. 그림 4는 내부 I-node의 구조를 나타낸 것이다.

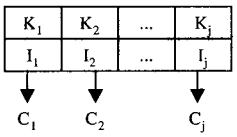


그림 4 내부 I-node의 구조

내부 I-node에서 각 원소는 {K, I, C}로 구성된다. K는 일반 애트리뷰트를 변환한 키의 값을 나타내며, I는 시간 구간으로 하위 노드들에 저장된 시간 구간들의 합이다. 시간 구간 I는 시간 구간의 시작 시간과 끝 시간, 즉, $[t_s, t_e]$ 형태로 저장된다. C는 자식 노드로의 포인터로써 키 값이 K 이하인 원소를 포함하는 I-node를 가리킨다.

최하단 I-node의 원소는 내부 I-node의 원소와 유사하나 상승된 것인지 아닌지 분류하기 위한 플래그 F를 가진다. F는 상승 요소인지 구분하는 플래그로써 C가 가리키는 노드가 A-node 인지 TA-node 인지를 구분하기 위한 것이다. F는 A-node를 가리키면 F(False), TA-node를 가리키면 T(True)의 값을 가진다.

A-node는 집계 데이타를 저장하는 노드로써 일반 애트리뷰트를 변환한 키 값과 시간, 집계 값을 저장한다. A-node의 구조는 그림 5과 같다.

K_1	K_2	...	K_j
T_1	T_2	...	T_j
V_1	V_2	...	V_j

그림 5 A-node의 구조

A-node의 원소는 {K, T, V}로 구성된다. K는 변환된 키 값이며 T는 시간 구간을 표현하기 위한 것으로 시간 구간의 끝 시간이다. V는 해당하는 키 값, 시간 구간의 집계 값이다. A-node는 같은 키 값을 가지는 데이타의 시간은 겹치지 않는 시간 구간으로 저장한다. 따라서, 하나의 시간 값으로만 저장하여도 시간 구간을 다시 생성할 수 있다. 즉, 연속된 원소 E_1, E_2, E_3의 시간 값이 T_1, T_2, T_3 라면, 원소 E_2의 시간 구간 I_2는 $<T_1+1, T_2>$가 된다. 이를 위해서는 새로운 데이타를 삽입할 때 같은 키 값을 가지는 데이타가 이미 존재할 경우 시간 구간이 겹치지 않도록 한다. 예를 들어, 키 값이 1000인 데이타의 시간 구간이 <10, 20>, 집계 값이 1인 데이타가 삽입되면 "1000, <9>, 0", "1000, <20>, 1" 인 두 개의 데이타가 A-node에 저장된다. 이 방법을 통해 시간대가 유사한 데이타가 많이 존재할 경우 기존 데이타의 갱신만으로도 데이타 삽입을 효율적으로 처리할 수 있다.

TA-node는 B-tree와 유사한 구조로써 Aggregation-tree에서와 같은 방법으로 시간 구간에 대한 집계 값을 저장한다. 내부 노드의 원소도 집계 값을 저장하며 이는 각 원소가 가리키는 시간 구간을 포함하는 시간 구간을 가지는 데이타가 삽입되었을 경우 갱신된다. TA-node는 SB-tree와 유사한 구조로써 동일한 키 값을 가지는 데이타들에 대해 시간으로만 집계 값을 저장한다. TA-node는 A-node의 분할 때 동일한 키 값을 가지는 원소가 A-node 저장량의 일정 부분 이상 차지할 때 그 키 값을 가지는 집계 데이타들만 추출하여 생성하게 된다. 이후 동일한 키 값을 가지는 데이타는 모두 이 TA-node에 시간 구간만 고려되어 저장하게 된다.

3.3 삽입 알고리즘

새로운 집계 데이타는 PTA-tree에서 I-node로 구성된 색인을 이용해 삽입될 A-node나 TA-node를 결정

하게 된다. 내부 I-node에서는 B$^+$-tree에서와 동일하게 삽입될 데이타의 키 값을 이용하여 방문할 다음 자식 노드를 결정하게 된다. 최하단 I-node에서는 삽입될 데이타와 동일한 키 값을 가지는 상승 원소가 존재하면 그 원소가 가리키는 TA-node에 데이타를 삽입한다. 그렇지 않으면 키 값이 포함되는 키 구간의 데이타를 저장하는 A-node에 저장된다. 이러한 I-node에서의 삽입 알고리즘은 재귀적으로 작성할 수 있으며 다음의 알고리즘 I-Insert는 I-node에서의 삽입 알고리즘을 나타낸다. 알고리즘에서 Nc는 알고리즘을 수행할 현재 I-node를 나타내며 D는 삽입할 데이타를 나타낸다. 삽입할 데이타의 변환된 키 값은 D.K, 시간 구간은 D.I 이다. 또한, X.R은 노드의 원소 X의 키 값 구간을 의미한다. 알고리즘에서, 초기에 Nc는 I-node의 루트 노드로 설정된다.

```
Algorithm: I-Insert(I-node Nc, Data D)
if Nc is an internal node
begin
  find a set of elements E that each element's
  key-range R  D.K
  for each element X in E, call I-Insert(X.C, D)
end
else
begin
  if ∃X in Nc which satisfies X.F=T and X.K=D.K,
    insert D into the TA-node which is pointed by X.C
  else
    find an element X in E whose key-range X.R ∋ D.K
    insert D into the A-node which is pointed by X.C
end
```

```
Algorithm: A-Insert(A-node Nc, Record D)
    if there is no record in Nc whose key K equals D.K,
       insert two records, {D.K, D.ts-1, 0} and {D.K, D.te,
       D.V}, into Nc ;
else
begin
       for each record X in Nc whose key K = D.K,
        begin
              if X.I covers D.I,
              begin
                 insert two records, {D.K, D.ts-1, X.V} and
                 {D.K, D.te, D.V};
                 goto EXIT;
        end
              else if D.I covers X.I, update X.V;
              else if X.I ∋ the start time ts of D.I,
              begin
                 insert {D.K, D.ts-1, D.V} into Nc;
                 update X.V;
        end
              else if X.I ∋ the end time te of D.I,
              begin
                 compute aggregate value Vnew with D.V
                 and X.V;
                 insert {D.K, D.te, Vnew};
                 goto EXIT;
              end
    end
EXIT:
end
```

A-node에서의 삽입은 각 키에 대해서 시간 구간을 시간 중심으로 이루어지게 된다. 따라서, 동일한 키 값을 가지는 데이타의 시간 구간과 새로 삽입하는 데이타의 시간 구간과의 포함 관계에 따라서 시간 구간을 분할하거나 기존 데이타의 값을 갱신하게 된다. 새로운 데이타의 키 값과 동일하면서 데이타의 시간 구간에 완전히 포함되는 원소들의 경우 그 값만 갱신하게 되며 그렇지 않고 시작 시간이나 끝 시간을 포함하는 시간 구간이 있다면 시간 구간을 분할하도록 새로운 원소를 추가하고 집계 값을 수정하게 된다. 알고리즘 A-Insert는 A-node의 삽입 알고리즘을 나타낸다. 알고리즘에서 Nc는 알고리즘을 수행할 A-node, D는 삽입되는 데이타를 나타낸다.

3.4 분할 알고리즘

새로운 집계 데이타가 A-node나 TA-node에 삽입되었을 때 노드에 빈 공간이 없으면 노드 분할을 수행하게 된다. TA-node는 B-tree와 동일한 분할 알고리즘을 사용하며 새로운 원소의 집계 값은 0으로 초기화된다. A-node의 분할은 키 값을 기준으로 노드 분할을 수행하게 된다. 그러나, A-node에서는 특정한 키 값에 데이타가 집중되는 핫 스팟(hot spot) 문제가 있을 수 있으며 이 경우에 대한 처리가 필요하다. A-node의 분할 알고리즘에서는 특정 키 값의 데이타가 A-node 저장량의 일정량 이상일 경우 이 데이타들을 이용하여 새로운 TA-node를 생성하며 이 한계량은 50%로 설정한다. 알고리즘 A-Split은 A-node 분할 알고리즘을 나타낸다.

```
Algorithm: A-Split
calculate the number of records, num(K), for each
distinct key K;
if there is a key K' whose num(K') is greater than half
   of the capacity of the A-node
        extract a set of record whose keys equals K'
        and construct new TA-nodes using them;
else
        split the A-node into two A-nodes to have
        approximately the same number of records;
```

I-node의 원소는 A-node 분할에 의해 생성되며 I-node에 공간이 없을 경우 분할이 일어나게 된다. I-node의 분할 방법은 내부 노드와 최하단 노드에서 차이가 있다. I-node는 변환 키를 기반으로 B$^+$-tree에 기반하여 저장하므로 내부 노드에서의 분할 방법은 기존의 B$^+$-tree에서의 분할 방법과 유사하다. 단, 루트 노드가 새로 생성될 경우에는 새로운 노드가 가장 큰 키 값을 포함하는 원소를 포함하여 2개의 원소를 가지도록 한다. 최하단 I-node의 분할에서는 Flag가 T인, 즉 상승 원소를 고려하여 분할을 수행해야 한다.

최하단 I-node에서는 A-node를 가리키는 원소와 TA-node를 가리키는 상승 원소가 함께 존재하게 된다. 최하단 노드에서 상승 원소가 존재할 경우 이 원소가 가리키는 변환 키의 범위를 포함하는 원소가 반드시 존재해야 한다. 따라서, 최하단 I-노드의 분할에서는 기본적으로 B$^+$-tree에서와 같이 분할 방법을 수행하되 새로이 분할된 노드에 할당된 마지막 원소가 상승 원소일 경우 새로운 비상승 원소를 추가한다. 그리고, 해당하는 A-node에는 분할된 다른 최하단 I-node의 최초 비상승 원소가 가리키는 A-node의 데이타들을 분할하게 된다. 예를 들어, I-node X를 분할하여 Y, X' 의 두 노드로 분할되고 Y의 마지막 원소 E_k가 상승 원소이고 X'의 최초 비상승 원소가 E_f일 경우, Y에는 새로운 비상승 원소 E_j를 추가하게 된다. 이 때, E_j는 새로운 A-node를 가리키며 변환 키 값은 E_k.K를 가진다. 또한, E_j의 A-node에는 E_f.C가 가리키는 A-node의 데이타에서 변환 키 값이 E_k.K 이하인 원소들을 추출하여 저장하게 된다. 즉, 새로 추가된 비상승 원소의 키 값이 가리키는 구간에 포함되는 데이타들은 새로운 A-node에 저장하는 것이다. 알고리즘 I-Deepest-Split은 최하단 I-node의 분할 알고리즘을 나타낸다.

Algorithm: *I-Deepest-Split*

```
    split node N into N, N' to have approximately same
    number of records;
if the last record X in N is a promoted record,
begin
        insert a new unpromoted record E into N;
        allocates a new A-node and set E.C as the new
        A-node;
        set E.K to X.K;
        select first unpromoted record E' from N';
        extract records from an A-node that is indicated
        by E'.C;
        insert extracted records into the A-node that is
        indicated by E.C;
  end
```

3.5 질의 처리 알고리즘

다차원 시간 집계 연산은 시간 구간 뿐만 아니라 일반 애트리뷰트의 값이나 범위를 조건으로 사용한다. 시간 집계 연산의 결과는 시간 구간과 해당하는 집계 값의 집합으로 구성된다. PTA-tree에서는 I-node 부분에서 조건에 맞는 경로를 따라 A-node와 TA-node를 방문하여 시간 집계 연산을 처리한다. I-node에서는 키 값을 기준으로 하여 조건에 맞는 경로를 찾게 된다. 방문된 A-node, TA-node에서는 시간 구간과 집계 값의 집합을 중간 결과물로 생성하게 된다. 집계 연산의 최종 결과는 각 경로의 데이타 노드에서 생성된 집계 결과를 병합 정렬(merge sort)에서와 같은 방법으로 생성한다. 알고리즘 I-ComputeAgg는 질의 처리 알고리즘을 나타낸 것이다. R은 일반 애트리뷰트의 질의 구간, I는 시간 질의 구간을 나타낸다.

Algorithm: *I-ComputeAgg* (I-Node N_c, attribute interval R, time interval I)

```
if Nc is an internal I-node,
begin
  for each record E in Nc whose key range intersects
  with R and time interval intersects with I,
  call I-ComputeAgg(E.C, R, I) and acquire a partial
  result;merge partial results into one result and return
  it;
end
else
begin
  for each record E in Nc whose key range intersect
    with R and time interval intersect with I,
    visit the A-node pointed by E.C and acquire partial
    result;merge partial results into one result and
    return it;
end
```

4. 성능평가

이 장에서는 PTA-tree를 기반으로 한 처리 기법의 성능을 비교한다. 비교 대상으로는 가장 효율적인 기존의 디스크 기반 집계 트리인 SB-tree를 다차원 시간 집계를 처리할 수 있도록 직관적으로 확장한 기법을 이용하도록 한다. SB-tree는 디스크 기반 기법 중 가장 효율적인 방법이므로 각 노드에 시간 구간 외에 일반 애트리뷰트의 구간을 추가하고 각 시간 구간별로 일반 애트리뷰트를 키로 하여 집계 데이타를 저장하는 색인을 추가하여 확장하도록 한다. 즉, 직관적으로 SB-tree에 저장되는 각 시간 구간에 B-tree와 같이 키를 기반으로 하는 트리를 할당하여 각 시간 구간을 포함하는 집계 데이타를 B-tree에 키 값 기반으로 저장하는 것이다. 본 논문에서는 실험적 평가 결과 뿐만 아니라 최악 경우의 공간 복잡도 분석 결과를 제시한다.

4.1 복잡도 분석

Property1. *최악 경우 공간 복잡도는 확장된 SB-tree에서는 O(N logN), PTA-tree에서는 O(N)이다.*

확장된 SB-tree에서는 하나의 데이타를 시간 구간에 따라 중복하여 저장하게 된다. 최악의 경우 루트 노드에서부터 리프 노드까지 2개의 경로를 따라 루트를 제외한 각 노드에서 (B-1)번의 삽입을 수행하게 된다. 여기서 B는 노드에 저장 가능한 최대 원소의 수이다. 즉, SB-tree의 높이를 h라 할 때 최악 경우 하나의 데이타를 저장할 때 (B-2)+(h-1)*(B-1) 만큼의 시간 구간에 데이타를 저장하게 되므로 많은 중복이 발생하게 된다. 여기서 SB-tree의 높이 h는 O(log N)에 비례하므로

결국 하나의 데이타를 저장할 때 최악 경우 O(log N) 만큼 데이타 중복이 발생하게 된다. 따라서, N개의 데이타에 대하여 최대 O(N log N) 만큼의 데이타가 생성된다.

반면 PTA-tree에서는 하나의 데이타를 처리하는 과정에서 변환 키가 동일한 경우에 대해 시간 구간을 분할하도록 하므로 최대 2개의 데이타가 추가된다. 따라서, N개의 데이타에 대하여 최대 2N 개의 데이타가 생성될 수 있으며 최악 경우 O(N)의 저장 공간을 필요로 하게 된다. 따라서, 공간 복잡도 면에서 확장된 SB-tree보다 우수함을 알 수 있다.

Property2. *트리의 공간 사용량을 예측하기 위해서는 노드의 최저 사용량 혹은 최저 사용량을 알 수 없는 노드의 최악 경우 개수를 파악할 수 있어야 한다. 확장된 SB-tree에서는 노드의 최저 사용량을 알 수 없으며, PTA-tree에서는 최저 사용량을 알 수 없는 노드의 갯수는 2*(최하단 I-node 수 - 1) 이다.*

확장된 SB-tree에서는 내부 노드 마다 노드의 시간 구간에 해당하는 변환 키를 저장하는 노드를 가지게 된다. 이 경우 데이타가 밀집되지 않은 시간 구간의 경우 노드에 저장 가능한 수에 비해 적은 수의 데이타가 저장된다. 따라서, 노드의 최소 공간 사용량을 알 수 없으므로 공간 사용량을 예측할 수 없는 문제점이 있다.

PTA-tree에서는 각 노드 종류마다 최소 공간 사용량을 예측할 수 있다. I-node에서는 기본적으로 B-tree의 구조를 채택하므로 최소 50%의 공간 사용량을 보장할 수 있다. A-node에서는 동일한 키 값을 가진 데이타가 존재하며 이를 이용해 분할을 수행하므로 최소 33%의 공간 사용량을 보장할 수 있다. 그러나, A-node 분할에서 TA-node가 생성될 경우 A-node의 공간 사용량은 보장할 수 없으나 TA-node는 A-node에 저장 가능한 양의 50% 이상이 동일한 키 값을 가질 경우 생성된다. 즉, 최소 공간 사용량을 알 수 없는 A-node가 존재할 경우는 TA-node가 생성된 경우이며 이 경우 평균적으로 A-node에 저장 가능한 양의 50% 를 사용하고 있다고 볼 수 있다. 따라서, 데이타 영역의 공간 사용량은 최소 33%의 A-node 공간 사용량을 보장하는 것으로 볼 수 있다. I-node 분할 때 상승 원소인지 여부에 따라서 최소 공간 사용량을 알 수 없는 A-node가 생성되며 이러한 A-node의 최대 갯수는 2*(최하단 I-노드 - 1)이 된다. 결론적으로 A-node의 허용량을 기준으로 데이타 영역은 최소 33%의 최소 공간 사용량을 보장하며 공간 사용량을 예측할 수 없는 A-node가 존재하나 그 최대 갯수는 예측 가능하므로 전체적인 공간 사용량을 예측할 수 있다.

표 2 공간 복잡도 분석

	최악 경우	저용량 노드 수
확장 SB-tree	O(N log N)	예측 불가
PTA-tree	O(N)	2*(최하단 I-node 수-1)

4.2 성능 실험

본 연구에서는 집계 데이타 저장시 필요한 디스크의 사용량과 질의 처리시 필요한 디스크 접근 회수를 평가의 척도로 하여 기존의 SB-tree를 확장한 다차원 시간 집계 트리와 PTA-tree에서의 시간 집계 처리 성능을 평가한다. 디스크의 사용량은 질의 처리의 비용을 가늠할 수 있도록 해주며 질의 처리시 디스크 접근 회수는 질의 응답 시간과 매우 밀접한 관련이 있는 중요한 요소이다.

표 3 실험 패러미터

실험 패러미터	종류
데이타 개수	50000, 100000, 150000, ..., 500000
일반 애트리뷰트 수	2
시간 구간의 범위	[0, 10], [0, 100000]
일반 애트리뷰트의 범위	[0, 100], [0, 100000]
애트리뷰트/포인터 크기	4바이트

본 실험은 다음과 같이 수행된다. 먼저 실험의 대상이 되는 시간 데이타를 생성하고 이에 대한 확장된 SB-tree 및 PTA-tree를 구성한 후, 질의의 대상이 되는 일반 애트리뷰트 구간과 시간 구간을 생성하여 생성된 트리에서 질의 처리를 수행한다. 이 과정에서 생성된 트리의 디스크 페이지 사용량을 관찰하며, 질의 처리에 필요한 디스크 접근 회수를 관찰한다. 실험 과정에서 사용된 패러미터(parameter)로는 애트리뷰트 값의 범위, 시간 구간의 범위, 시간지원 데이타의 개수, 디스크 페이지의 크기 등을 사용한다. 본 실험에서는 시간 구간의 범위와 애트리뷰트 값의 범위가 조밀한(dense) 지 여부에 따라 디스크의 사용량을 실험하도록 한다. 시간지원 데이타의 개수는 50000, 100000, 150000, 200000, 250000, 300000, 350000, 400000, 450000, 500000개를 대상으로 한다. 시간 구간의 범위는 조밀하지 않은 경우 [0, 100000]으로, 조밀한 경우 [0, 100]으로 설정한다. 100000이라는 시간은 1시간을 최소 시간 단위로 할 때, 10년간의 시간을 표현할 수 있는 값이다. 시작 시간과 끝 시간은 인위적(random)으로 생성되며 균일 분포(uniform distribution)를 이룬다고 가정한다. 일반 애트리뷰트의 수는 2개로 하였으며 값의 범위는 조밀한 경우와 조밀하지 않은 경우에 대해 실험을 수행하였다. 조밀한 경우, 각 애트리

뷰트는 [0, 100000]의 범위를 가지며 조밀한 경우 각 애트리뷰트는 [0, 10]의 범위를 가진다. 애트리뷰트의 값도 인위적으로 생성되며 균일 분포를 이룬다고 가정한다. 시간을 나타내는 애트리뷰트와 다른 애트리뷰트, 포인터의 크기는 모두 4바이트로 한다.

4.3 실험 결과 및 분석

본 실험에서 제시된 각 실험 결과는 12번의 반복 실험을 통하여 최대값과 최소값을 제외한 나머지 10개의 측정치에 대한 평균값이다. 먼저 시간지원 집계 처리를 위하여 생성된 트리의 디스크 사용량에 관하여 알아본다. 그림 6은 시간구간의 범위 [0, 100000], 애트리뷰트 값의 범위 [0, 100000]인 조밀하지 않은 데이타 분포에 대한 디스크 사용량을 나타낸다. 전체적으로 PTA 트리가 확장된 SB-tree에 비해 디스크 사용량을 적게 사용하며 데이타 수가 50만개인 경우에도 확장된 SB-tree가 사용하는 디스크 페이지 수의 1/3 이하로 적은 디스크 페이지를 사용함을 알 수 있다.

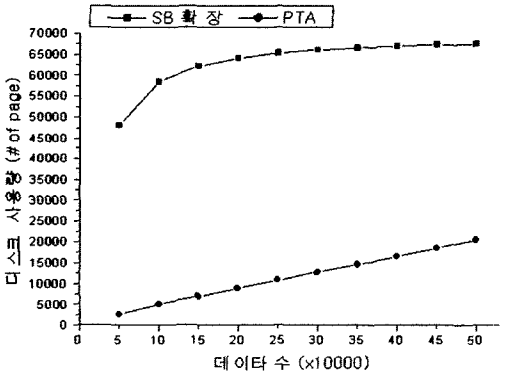


그림 6 조밀하지 않은 데이타 분포에 대한 디스크 사용량

그림 7은 시간구간의 범위는 [0, 100000]로 조밀하지 않으며, 애트리뷰트 값의 범위는 [0, 10]로 조밀한 데이타 분포에 대한 디스크 사용량을 나타낸다. 이 경우 애트리뷰트의 범위는 [0,10]*[0,10]이므로 하나의 애트리뷰트 당 최대 5000개까지 시간 구간이 저장된다. 따라서, PTA-tree에서는 TA-node가 생성되어 집계 데이타를 저장하는 공간이 절약된다. PTA-tree의 디스크 사용량은 확장된 SB-tree의 1/3 이하로 적은 양을 사용함을 알 수 있다.

그림 8은 시간 구간의 범위 [0, 100], 애트리뷰트 값의 범위 [0, 10]으로 모두 조밀한 데이타 분포에 대한 각 트리의 디스크 사용량을 나타낸다. 이 경우 각 시간 구간에 저장되는 데이타와 각 애트리뷰트 값에 저장되는 데이타의 수가 아주 조밀하므로 거의 비슷한 만큼 디스크 페이지를 사용하며 차이도 거의 없음을 알 수 있다.

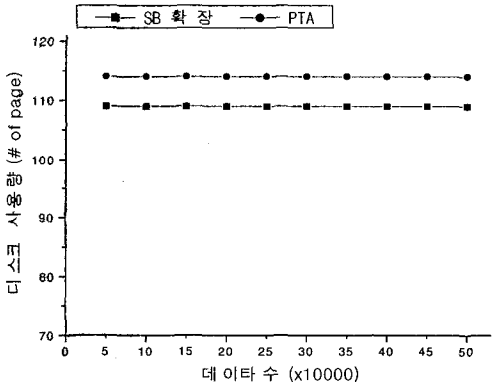


그림 8 조밀한 데이타 분포에 대한 디스크 사용량

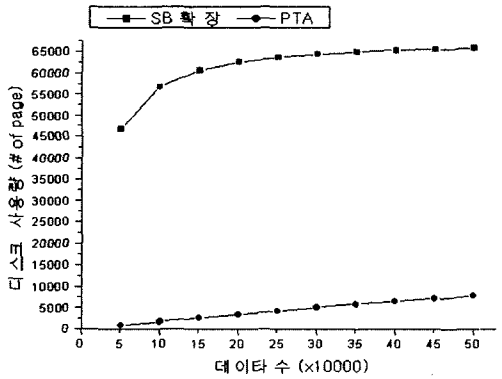


그림 7 조밀하지 않은 시간 분포에서의 디스크 사용량

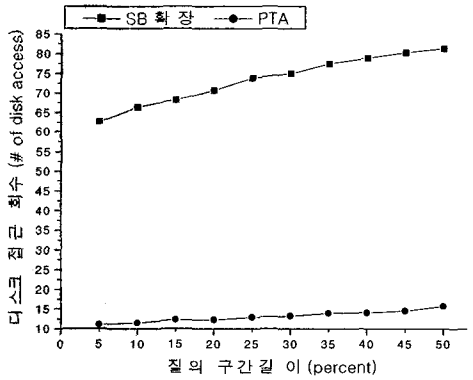


그림 9 질의 구간 길이에 따른 디스크 접근 회수

그림 9는 집계 처리에 필요한 디스크 접근 회수를 나타낸다. 디스크 접근 회수는 실제 실행 시간에 많은 비중을 차지하므로 집계 처리의 실행 시간에 대한 척도가 된다. 데이타의 갯수는 50000개로 하였으며 질의 조건인 시간 구간과 일반 애트리뷰트 구간의 최대 길이는 최대 구간 길이의 5, 10, 15, 20, 25, 30, 35, 40, 45, 50% 이내로 제한하였으며, 데이타 분포는 조밀하지 않도록 하여 디스크 접근 회수를 측정하였다. PTA-tree가 확장된 SB-tree 보다 3배 이상 적은 디스크 접근 회수를 가짐을 알 수 있다.

5. 결 론

질의 연산자 중 집계 연산은 릴레이션 전체 혹은 그 일부를 구성하는 튜플들에 적용되어 전체 값을 계산하거나 대표 값을 선택하는 연산으로 질의어의 중요한 요소이며, 다양한 응용 분야에서 많이 사용된다. 그러나, 시간 애트리뷰트를 포함하는 시간지원 데이타베이스에서의 집계 연산은 시간 애트리뷰트를 고려하지 않은 기존의 집계 연산과는 큰 차이가 있다. 따라서, 일반적인 데이타베이스에서 연구된 집계 연산 처리 기법을 시간지원 데이타베이스에 그대로 적용하는 것은 불가능하다. 이러한 시간 집계 연산을 효율적으로 처리하기 위한 방법이 연구되어 왔으나 전화기록(CDR), 데이타 웨어하우스 같은 방대한 기록에서 좀 더 유용한 정보를 얻기 위해서는 시간 구간 뿐만 아니라 다른 일반 애트리뷰트까지 조건으로 하는 일반적인 다차원 시간지원 집계 처리를 처리할 수 있는 방안이 필요하다. 이러한 방안으로 MVBT-tree가 제안되었으나 시간이 증가하는 처리 시간만 고려하였고 일반 애트리뷰트는 하나 밖에 포함할 수 없으므로 다차원으로 확장될 수 없다는 문제점을 지니고 있다.

본 논문에서는 시간 애트리뷰트 뿐만 아니라 하나 이상의 일반 애트리뷰트를 포함하는 다차원 시간지원 집계를 효율적으로 처리할 수 있는 구조로서 PTA-tree를 제안하고, 이를 이용한 시간지원 집계의 처리 기법을 제안하였다. PTA-tree는 B$^+$-tree에 기반한 트리로써 일반 애트리뷰트를 변환 기법을 이용하여 키로 변환하고, 집계 데이타를 키 중심으로 저장한다. 하나의 변환된 키에 데이타가 집중될 경우 이 데이타들만 분리하여 따로 시간 기반으로 집계 데이타를 저장하는 TA-node에 저장하며 PTA-tree에 키 구간이 아닌 하나의 키를 가리키는 요소를 추가(promotion)하도록 하여 문제를 해결하였다. 제안된 PTA-tree는 균형화된 트리로서 트리 구성이 용이하며 기존의 시간 집계 처리 기법을 확장한 방법으로는 효율적이지 못한 다양한 경우에 확장된 방법보다 유사하거나 훨씬 나은 성능을 보인다.

본 논문에서는 복잡도 분석과 실험을 통해 PTA-tree가 기존의 시간 집계 연산 처리 기법을 확장한 기법에 비해 더 우수한 성능을 나타냄을 보였다. 근래에는 데이타가 대용량화되고 있으므로 병렬 시스템이나 분산 시스템이 활용되고 있다. 향후 PTA-tree를 이러한 환경에서 활용할 수 있도록 확장하는 방안에 대한 연구와 서로 다른 일반 애트리뷰트 조합에 대해 각각 PTA-tree를 둘 경우 주어진 질의를 처리하기 위해 어느 PTA-tree를 사용할 것인지를 결정하는 문제에 대한 연구가 필요하다.

참 고 문 헌

[1] H.Gregersen, and C.S.Jensen. *Temporal Entity-Relationship Models - A Survey.* Technical Report R-96-2039, Aalborg University, 1996.

[2] J.S.Kim and M.H.Kim, *An Effective Data Clustering Measure for Temporal Selection and Projection Queries*, Decision Support Systems, 30(1), p33-50, Elsevier Science, 2000.

[3] M.D.Soo, R.T.Snodgrass, and C.S.Jensen. *Efficient Evaluation of the Valid-Time Natural Join.* Proceedings of the International Conference on Data Engineering, 1994.

[4] B.Salzberg, and V.J.Tsotras. *A Comparison of Access Methods for Time Evolving Data.* ACM Computing Surveys, 1997.

[5] P.A.Tuma. *Implementing Historical Aggregates in TempIS.* Master's Thesis, Wayne State University, Nov. 1992.

[6] C.Dyreson, W.Evans, H.Lin and R.T.Snodgrass. *Efficiently supporting Temporal Granularities.* IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 4, pages 568-587, 2000.

[7] N.Kline, and R.T.Snodgrass. *Computing Temporal Aggregates.* International Conference on Database Engineering, pages 222-231, 1995.

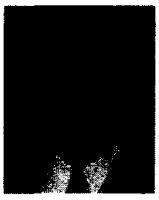
[8] B.Moon, I.F.V.Lopez and V.Immanuel. *Efficient Algorithms for Large Temporal Aggregation.* IEEE Transactions on Knowledge and Data Engineering, 2002.

[9] J.S.Kim, S.T.kang and M.H.Kim, *On Temporal Aggregate Processing based on Time Points*, Journal of Korea Information Science Society (KISS), p.1418-1427, Vol.26, No.12, 1999.

[10] J.Yang and J.Widom, *Incremental Computation and Maintenance of Temporal Aggregates.* International Conference on Data Engineering, pages 51-60, 2001.

[11] D.Zhang, A.Markowetz, V.Tsotras, D.Gunopulos and B.Seeger, *Efficient Computation of Temporal Aggregates with Range Predicates*, A Time Center Technical Report TR-52, Dec 2000
[12] J.Gray. *The Benchmark Handbook for Database and Transaction Processing Systems*. Morgan Kaufmann, 1991.
[13] R.T.Snodgrass, I.Ahn, G.Ariav, and et al. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.
[14] R.Epstein. *Techniques for Processing of Aggregates in Relational Database Systems*. UCB/ERL M7918, Feb 1979.
[15] R.T.Snodgrass, S.Gomez and L.E.McKlenzie. *Aggregates in the temporal query language TQuel*. IEEE Transaction on Knowledge and Data Engineering, Vol. 5, No. 5, pages 826-842, October 1993.
[16] V.Gaede and O.Gunther. *Multidimensional Access Methods*. ACM Computing Surveys, Vol. 30, Issue. 2, pages 170-231, 1998.
[17] F.Ramsak, V.Markl, R.Fenk and et al. *Integrating the UB-tree into a Database System Kernel*. International Conference on Very Large Databases, pages 263-272, 2000.

강 성 탁
1996년 한국과학기술원 전산학과(학사). 1998년 한국과학기술원 전산학과(석사). 1998년 ~ 현재 한국과학기술원 전산학전공(박사). 관심분야는 Temporal Database, Spatio-Temporal Database, Data Warehouse, OLAP



정 연 돈
1994년 고려대학교 전산과학과(학사). 1996년 한국과학기술원 전산학과(석사). 2000년 한국과학기술원 전산학전공(박사). 2001년 한국과학기술원 전산학전공 Post-Doc. 2001년 ~ 현재 한국과학기술원 전산학전공 연구 교수. 관심분야는 Spatio-Temporal Database, Mobile Computing, Distributed Systems, XML, Database Systems

김 명 호
정보과학회논문지 : 데이타베이스 제 29 권 2호 참조