선택 프레디키트를 포함하는 시간 집계의 효율적 처리

(Efficient Processing of Temporal Aggregation including Selection Predicates)

강 성 탁 [†] 정 연 돈 [††] 김 명 호 [†††]

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

요 약 시간지원 데이타베이스 시스템에서의 시간 집계 연산은 일반적인 집계 연산의 확장으로써, 집계의 범위 조건에 '시간'을 포함한다. 시간 집계 연산은 이력 데이타 웨어하우스, 전화 기록 관리(CDR) 등에 유용하다. 본 논문에서는 질의 조건에 여러 개의 선택 프레디키트들을 포함하는 시간 집계 연산을 효율적으로 처리하기 위한 자료 구조인 ITA-tree를 제안하고, 이를 이용한 시간 집계 처리 기법을 제안한다. ITA-tree에서는 레코드의 시간 구간을 T-value라는 하나의 값으로 변환한 후, B^+-tree와 비슷하게 이 값을 이용하여 색인을 생성한다. 또한, 많은 레코드가 동일한 T-value 값을 가지게 되는 핫-스팟 문제를 위해 개선된 ITA-tree인 eITA-tree를 제안한다. 본 논문에서는 제안된 기법들의 성능을 분석과 실험을 통해 비교한다.

키워드 : 시간지원 데이타베이스, 시간 집계 연산, 데이타베이스

Abstract The temporal aggregate in temporal databases is an extension of the conventional aggregate to include the time on the range condition of aggregation. It is a useful operation for Historical Data Warehouses, Call Data Records, and so on. In this paper, we propose a structure for the temporal aggregation with multiple selection predicates, called the *ITA-tree*, and an aggregate processing method based on the structure. In the ITA-tree, we transform the time interval of a record into a single value, called the *T-value*. Then, we index records according to their T-values like a B^+-tree style. For possible hot-spot situations, we also propose an improvement of the ITA-tree, called the *eITA-tree*. Through analyses and experiments, we evaluate the performance of the proposed method.

Key words : temporal database, temporal aggregation, database

1. 서 론

일반적인 데이타베이스는 현실 세계에서 발생한 사건에 대하여 가장 최근의 상태만을 반영하지만, 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[1]. 시간지원 데이타베이스는 경향 분석, 버전 관리, 비디오 데이타 관리 등과 같이 관리되는 자료의 시간적 특성이 중요시 되는 모든 분야에 폭 넓게 응용될 수 있다. 또한 시간지원 데이타베이스는 데이타 웨어하우스(data warehouse)에서의 자료 이력 관리에도 효과적으로 이용될 수 있다.

시간지원 데이타베이스 분야에 대한 초기 연구로는 주로 자료 모델링 기법[2]과 질의 언어[3,4], 색인 기법과 질의 처리 방법, 저장 구조 등과 같이 구현과 관련된 문제들[5-11]이 수행되어 왔다. 최근에는 시간 집계와

† 정 회 원 : 한국과학기술원 전산학과
stkang@dbserver.kaist.ac.kr
†† 종신회원 : 고려대학교 컴퓨터학과 교수
ydchung@korea.ac.kr
††† 종신회원 : 한국과학기술원 전산학과 교수
mhkim@dbserver.kaist.ac.kr
논문접수 : 2004년 2월 4일
심사완료 : 2007년 10월 22일

정보과학회논문지: 데이타베이스 제35권 제3호(2008.6)

같은 연산들과 이들의 처리 기법에 관한 연구들이 수행되고 있다[12-20].

그러나, 시간 집계 연산을 위한 대부분의 연구들에서는 데이타의 시간 애트리뷰트만 질의의 조건으로 고려하고 있다. 시간지원 데이타베이스의 다양한 분석을 위해서는 질의 조건에 시간 애트리뷰트 뿐만 아니라 다른 선택(selection) 프레디키트들도 필요하다. 이와 같이, 시간 구간 외에 여러 선택 프레디키트들을 조건으로 할 수 있는 시간 집계를 *일반 시간 집계(General Temporal Aggregation)* 연산이라고 한다. 일반 시간 집계 연산은 특히 크기가 큰 이력 데이타웨어하우스(historical datawarehouse)에서 유용하다[19]. 그림 1은 일반 시간 집계 연산의 간단한 예를 표현한 것으로, 고용 이력 릴레이션으로부터 고용 기간이 1980에서 2000사이, 나이가 20에서 30사이, 봉급이 30000에서 40000사이인 데이타에 대한 시간 집계를 구하는 것이다. 여기서, *employ_time*은 시간 애트리뷰트이며, *age*와 *salary*는 선택 조건 애트리뷰트들이다.

```
Compute temporal-aggregate
From    EmployeeHistory
Where   1980≤employ_time≤2000 and
        20≤age≤30 and 30000≤salary≤40000;
```

그림 1 일반 시간 집계 연산의 예

본 논문에서는 시간 뿐만 아니라 선택 조건 애트리뷰트까지 포함하는 일반 시간 집계 연산을 효율적으로 처리하기 위하여 ITA-tree(*Interval-Transformation-based Aggregation-tree*)를 제안하고, 이를 기반으로 한 시간 집계 처리 기법을 제안한다.

본 논문은 다음과 같이 구성되어 있다. 제 2장에서는 시간 집계 연산의 의미를 규정하며, 기존의 시간 집계 처리 기법들을 살펴본다. 제 3장에서는 ITA-tree와 시간 집계 처리 기법을 제안한다. 제 4장에서는 많은 레코드가 동일한 키 값을 가지게 되는 핫-스팟 문제를 설명하며 이 문제를 효과적으로 처리하기 위해 개선된 ITA-tree를 제안한다. 제 5장에서는 제안된 기법들을 분석하며 제 6장에서는 실험을 통해 성능을 비교한다. 마지막으로 7장에서는 연구 결과를 정리하고 향후 연구 방향에 대하여 기술한다.

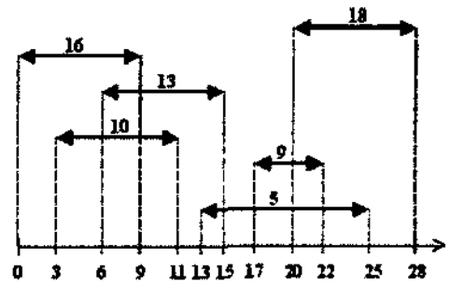
2. 연구 배경과 기존의 시간 집계 처리 기법

2.1 시간 집계 연산

전체 시간 구간은 시간 데이타의 시간 애트리뷰트에 의해 여러 개의 시간 구간으로 분할되며 이렇게 분할된 각각의 시간 구간마다 집계 연산의 결과값이 달라질 수 있다. 이 때, 집계 연산 값이 변하지 않는 시간 구간을 '불변 구간(constant interval)'이라 한다. 이와 같이 집계 연산 값이 변하지 않는 시간 구간들과 그 구간에서의 집계 연산 값을 계산하는 시간지원 데이타베이스에서의 집계 연산을 '시간 집계'라 한다. 다음의 그림 2는 시간 집계 MAX와 COUNT의 예이다[13]. 그림 2(a)는 시간 데이타를 보여주며, 2(b)는 시간 데이타를 알기 쉽게 그림으로 표현한 다이어그램이다. 그림 2(c)는 시간 집계 MAX와 COUNT의 결과이다. 예제에서 MAX에 대한 불변 구간들은 [0, 9], [10, 15], [16, 16], [17, 19], [20, 28]이다. 그리고, COUNT에 대한 불변 구간들은 [0, 9], [10, 15], [16, 16], [17, 19], [20, 28]이다(본 논문에서 시간은 0이상의 정수라 가정하며, 시간 구간은 [*start*, *end*]와 같은 형식으로 표기한다).

start time	end time	value
0	9	16
3	11	10
6	15	13
13	25	5
17	22	9
20	28	18

(a) 시간 데이터



(b) 시간 다이어그램

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

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

(c) 시간 집계 연산(MAX와 COUNT) 결과

그림 2 시간 집계 MAX와 COUNT의 예

2.2 기존의 시간 집계 연산 처리 기법

기존에 시간 집계를 처리하기 위한 많은 연구들이 진행되어왔다. 그러나, 대부분[13-17]은 시간 집계를 메인-메모리에서 처리한다. 근래에는 시간 집계 결과를 효율적으로 디스크에 저장하고 관리하는 것에 대한 연구[19, 20]가 진행되었다. 많은 애플리케이션에서 시간지원 데이타베이스에 저장되는 데이타의 양은 방대하기 때문에, 메모리-기반의 기법들은 부적합하다. 따라서, 본 논문에서는 디스크-기반 기법에 초점을 둔다.

SB-tree[20]는 디스크-기반 기법으로써 제안되었으며, Segment-tree와 B-tree에 기반하고 있다. SB-tree에서 불변 구간들과 각 불변 구간의 집계 값은 말단 노드에 저장된다. 집계 결과는 루트 노드에서 말단 노드까지의 경로 상에 있는 노드들에 저장된 집계값들로부터 계산된다. SB-tree는 시간 구간만 고려하며, 선택 조건 애트리뷰트는 고려하지 않는다.

MVSB-tree(Multi Version SB-tree)는 시간 애트리뷰트 외에 하나의 선택 조건 애트리뷰트를 포함하는 좀 더 일반적인 집계 연산을 처리하기 위한 기법으로써 제

안되었다. 그러나, MVSB-tree는 오직 단순 증가하는 시간인 트랜잭션 시간(transaction time)[1,4]만 지원하며, 둘 이상의 선택 조건 애트리뷰트는 지원할 수 없다. 따라서, MVSB-tree는 여러 선택 조건 애트리뷰트와 유효 시간(순서에 상관없는 시간)을 위한 일반 시간 집계는 처리할 수 없다.

본 논문에서는 다음을 시간 집계 처리 기법의 목적으로 고려하고 있다.

- 일반 시간 집계의 효과적인 처리(즉, 하나 이상의 선택 조건 애트리뷰트 지원)
- 두 가지 시간 개념 지원[1,4]: 유효 시간(valid time)과 트랜잭션 시간(transaction time)
- 효율적인 디스크 공간 활용성

기존의 연구들은 시간 애트리뷰트만 고려하거나, 하나의 선택 조건 애트리뷰트와 단순 증가하는 트랜잭션 시간만 지원하는 문제점이 있다. 따라서 순서에 상관 없는 유효 시간과 둘 이상의 선택 조건 애트리뷰를 포함하는 일반 시간 집계 연산을 효율적으로 처리할 수 있는 새로운 기법이 필요하다.

3. ITA-tree: 제안하는 일반 시간 집계 처리 기법

제안하는 새로운 기법에서는 색인 구성을 위한 값을 생성하기 위해 변환 기법[21-23]을 사용한다(본 논문에서, 변환된 값은 T-value라고 한다). 새로운 트리 구조인 *ITA-tree(Interval-Transformation-based Aggregation tree)*에서는 레코드의 시간 구간에 따라 T-value를 생성하고, T-value에 따라 레코드들로 B^+-tree 형태의 색인구조를 생성한다.

3.1 T-value 생성

ITA-tree에서는 레코드를 그들의 시간 구간으로부터 변환된 T-value들에 기반하여 저장한다. 변환 과정은 두 개의 단계로 구성된다. 첫 번째 단계에서는, corner 변환을 사용하여 시간 구간을 공간의 점으로 맵핑한다[21-23]. 시간 구간 $[t_s, t_e]$는 공간의 점 (t_s, t_e)로 맵핑된다. 그러면, 주어진 시간 구간에 교차하는 시간 구간을 찾으려면, 한 시간 구간의 시작 시간을 다른 시간 구간의 끝 시간과 비교하면 된다. 주어진 시간 구간 $[t_s, t_e]$에 교차하는 시간 구간들의 시작 시간은 t_e이하이고, 끝 시간은 t_s이상이다.

그림 3은 시간 구간 I([3,8])를 corner 변환한 것을 나타낸다. p는 시간 구간 I로부터 변환된 공간의 점이다. 또한, 시간 구간 I에 교차하는 시간 구간들은 $x \leq 8$이고 $y \geq 3$인 공간상의 점이 된다. 영역 A, B, C (y=x 라인의 위쪽)는 이러한 점들이 있는 영역을 나타낸다. 영역 A의 점들은 $x \leq 3$이고 $y \geq 8$인 점들이며, 영역

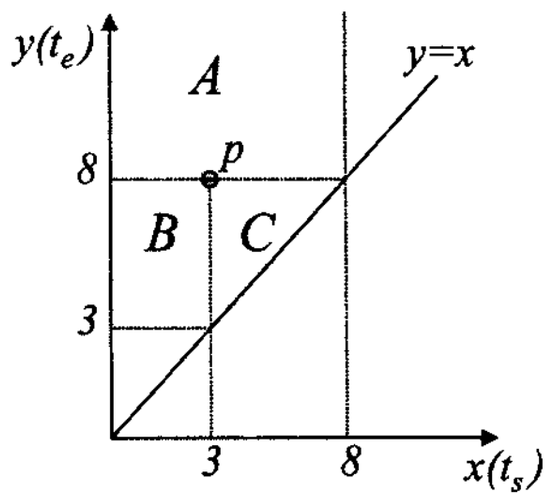


그림 3 주어진 시간 구간에 교차하는 변환 영역

B의 점들은 $x \leq 3$이고 $3 \leq y \leq 8$인 점들이며, 영역 C의 점들은 $3 \leq x \leq 8$이고 $3 \leq y \leq 8$인 점들이다. 따라서, 주어진 시간 구간 I([3,8])에 교차하는 시간 구간을 찾을 때는 영역 A, B, C(그림의 진한 영역)에 있는 점들을 찾게 된다.

두 번째 단계에서는, 변환된 공간 상의 점을 선형(linear) 변환 기법을 사용하여 하나의 값으로 변환하며, 이 값을 *T-value*라 한다. 다차원 데이타를 일차원 데이타로 변환하는 기법은 Z-ordering, Hilbert-Curves 등이 연구되어 왔다[24]. 본 논문에서는 단순화를 위해 기본적인 컬럼-스캔(Column-Scan) 변환 기법을 사용한다. 컬럼-스캔 변환 기법에서는 공간의 점을 하나의 큰 수로 변환한다. 예를 들어, 시간의 값이 1자리 수라고 가정한다. 그러면, 시간 구간 *[3, 4]*의 T-value는 *34*이다. 만약, 시간 값이 두 자리 수라면, T-value는 0304가 될 것이다 (본 논문의 나머지 예제에서는, 기본적인 컬럼-스캔 변환 기법을 사용한다).

3.2 ITA-tree의 구조

ITA-tree는 N-node와 A-node로 구성된다. 다음은 각 노드에 대한 설명이다.

- N-node(iNdex node): N-node는 A-node나 다른 N-node를 인덱싱한다. N-node의 레코드는 T-value, 자식 노드로의 포인터, 자식 노드에 있는 모든 선택 조건 애트리뷰트 값으로부터 계산된 선택 조건 애트리뷰트들의 값의 범위로 구성된다.
- A-node(Aggregate node): A-node의 레코드는 T-value, 선택 조건 애트리뷰트, 그리고 해당되는 T-value와 선택 조건 애트리뷰트의 집계값으로 구성된다.

N-node의 레코드 X는 <*K*, *I*, *C*>로 구성된다. *X.K*는 T-value 범위의 상한선이며, *X.I*는 선택 조건 애트리뷰트들의 값의 범위이며, *X.C*는 자식 노드로의 포인터이다. 자식 노드는 N-node 혹은 A-node이다. N-node의 자식 노드가 N-node이면 '내부 N-node', 그렇지 않으면 '말단 N-node'라 한다. N-node의 선택 조건 애트리뷰트들의 값의 범위는 하위 노드에 있는 레코드들의 선택 조건 애트리뷰트들의 값의 범위를 합(union)

한 것이다.

A-node의 레코드는 <*K*, *I*, *V*>로 구성된다. *K*는 T-value, *I*는 선택 조건 애트리뷰트들의 값의 범위, *V*는 집계 값이다.

그림 4는 ITA-tree 예제로써, 표 1의 예제 데이타로부터 ITA-tree를 생성하는 과정을 나타낸다. 표 1은 이동 통신을 위한 게이트웨이 상태 정보를 나타낸 것이다. 두 개의 선택 조건 애트리뷰트(게이트웨이 타입과 설치된 지역)는 게이트웨이 정보이며, 시간 구간과 집계(접속 기기 수)는 게이트웨이가 사용되는 시간과 그 시간에 접속한 이동 기기의 수이다. 표 1에서 각 시간은 한 자리 수이다. 따라서, T-value는 두 자리 수이다. 즉, 표 1에 있는 레코드들의 T-value는 35, 46, 35, 79, 89, 56, 68이다. N-node에는 최대 2개, A-node에는 최대 3개의 레코드가 저장될 수 있다고 가정하였다.

A-node의 레코드는 T-value, 선택 조건 애트리뷰트들, 그리고 집계 값으로 구성된다. 그림 4(b)의 레코드 "35, (1, 5), 1"에서, 35는 T-value, (1, 5)는 선택 조건 애트리뷰트들, 그리고 1은 집계 값이다. N-node 레코드의 구조도 유사하며, 추가적으로 선택 조건 애트리뷰트들의 값의 범위를 가진다.

그림 4(a)는 ITA-tree의 초기 상태를 나타낸다. 하나의 N-node와 하나의 A-node가 있으며, N-node에는 A-node를 가리키는 하나의 레코드가 있다. 이 레코드의 T-value는 최대값으로써 ∞이다. 그림 4(b)는 4(a)에 레코드 1, 2, 3을 삽입한 후의 모습이다. 그림 4(c)는 레코드 4, 5를 추가한 후의 모습이다. 레코드 4를 추가할 때, 노드 A_1에는 공간이 없다. 따라서, A_1은 A_1과 A_2로 분할되었다. 그리고, A_2를 가리키는 새로운 레코드가 N-node N_1에 삽입되었다. 그림 4(d)는 레코드 6, 7을 삽입한 후의 모습이다. 이 때, A_2는 A_2와 A_3로 분할되었다. 또한, N_1은 N_1과 N_2로 분할되었으며 새로운 N-node N_3가 새로운 루트 노드가 되었다.

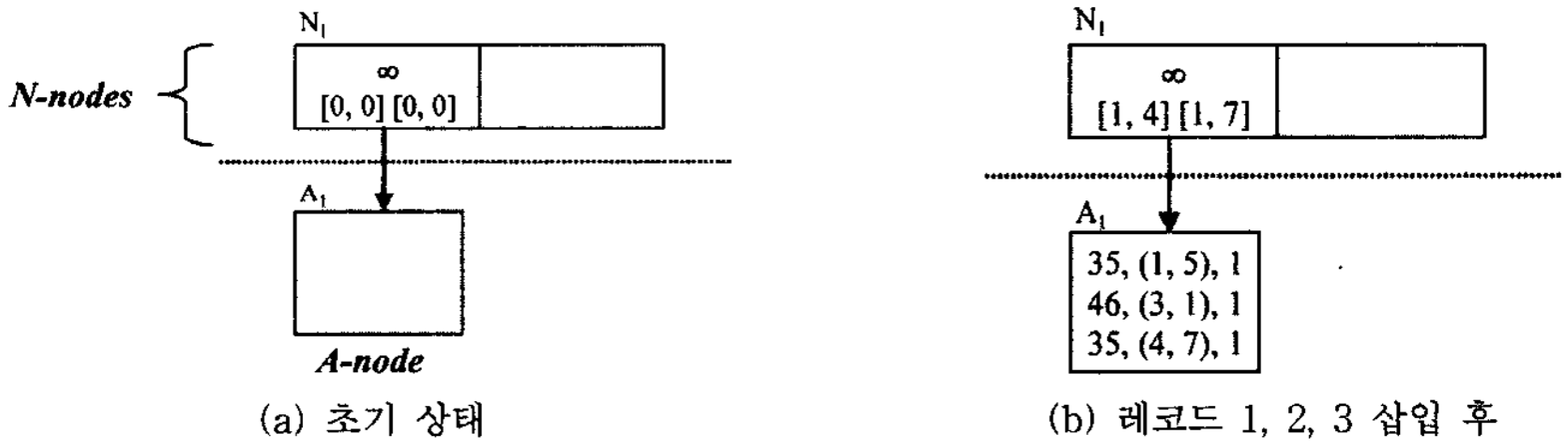
표 1 예제 시간 데이타

No	선택 조건 애트리뷰트		시간 구간	접속 기기 수
	타입	영역		
1	1	5	[3, 5]	3
2	3	1	[4, 6]	1
3	4	7	[3, 5]	6
4	5	3	[7, 9]	9
5	2	7	[8, 9]	20
6	5	4	[5, 6]	8
7	7	8	[6, 8]	6

3.3 알고리즘

3.3.1 삽입

레코드를 삽입할 때는, 먼저 N-node로 구성된 경로를 결정하고, 새로운 레코드를 저장할A-node를 선택한다. 경로를 결정할 때, 다음 방문할 자식 노드는 N-node의 T-value들로부터 결정된다(이 과정은 B^+-tree의 검색과 유사하다). 또한, 삽입 과정에서, 선택 조건 애트리뷰트들의 값의 범위 *I*도 갱신된다. 다음은 N-node에서의 삽입 알고리즘을 나타낸다. N_c는 현재의 N-node이며, D는 새로운 레코드이다. 레코드 D는



(a) 초기 상태 (b) 레코드 1, 2, 3 삽입 후

(c) 레코드 4, 5 삽입 후 (d) 레코드 6, 7 삽입 후

그림 4 ITA-tree 생성 과정

T-value K, 선택 조건 애트리뷰트 A, 집계값 V로 구성된다.

```
Algorithm: N-Insert(N-node Nc, Record D)
  if (Nc is an internal N-node) {
      find a record E whose T-value range includes D.K ;
      update E.I as the union of the E.I and D.A ;
      call N-Insert(E.C, D) ;
  } else {
      find a record X whose T-value range includes D.K ;
      update X.I as the union of the E.I and D.A ;
      call A-Insert(E.C, D) ;   // insert D into the A-node
  }
```

A-node에서의 삽입은 B^+-tree에서와 유사하다. 새로운 레코드의 T-value와 같은 T-value를 가진 레코드가 존재하면, 기존 레코드의 집계값만 갱신한다. 존재하지 않으면, 새로운 레코드가 A-node에 삽입된다. 예를 들어, 그림 4(b)에서는 동일한 T-value를 가진 레코드가 존재하지 않았으므로, 3개의 레코드가 A_1에 추가되었다. 다음은 A-node에서의 삽입 알고리즘을 나타낸 것이다. D는 새로운 레코드이며, N_c는 D가 삽입될 A-node이다. *agg*는 기존 집계값과 새 레코드의 집계값으로부터 집계값을 계산하는 집계 함수이다.

```
Algorithm: A-Insert(A-node Nc, Record D)
  if (there is a record X whose T-value K equals D.K in Nc) {
     update the existing record's aggregate value V as agg(X.V, D.V);
  } else {
     insert the new record, {D.K, D.A, D.V} into Nc;
  }
```

3.3.2 분할

A-node와 N-node의 분할은 B^+-tree의 분할과 거의 같다. 차이점은, (1) 루트 노드는 가장 큰 T-value 값을 가지는 레코드를 포함하여, 적어도 2개의 레코드를 가진다 (2) A-node를 분할할 때, 부모 N-node 레코드에 있는 선택 조건 애트리뷰트의 값의 범위도 갱신한다. 복잡하지 않으므로 이 알고리즘들에 대한 자세한 설명은 생략한다.

3.3.3 집계 처리

시간 집계의 조건 절은 선택 조건 애트리뷰트들의 값의 범위 R과 시간 구간 I ($[t_s, t_e]$)로 구성된다. 시간집계를 처리하려면 먼저 주어진 조건을 만족하는 모든 N-node 경로를 방문한다. 경로를 찾기 위해서는, N-node에 있는 레코드의 T-value와 선택 조건 애트리뷰트들의 값의 범위를 주어진 조건과 비교한다. 값의 범위가 겹치는지 비교하는 것은 간단하다. 시간 구간 비교를 위해서는 ITA-tree의 T-value를 사용한다. N-node에서 T-value는 T-value 범위의 상한선이다. 주어진 시간 구간 I에 교차하는 시간 구간들의 T-value 'αβ'는 $\alpha \leq I.t_e$ 이고 $\beta \geq I.t_s$이다. 또한, 이러한 T-value들은 그림 3에서 보인 영역들에만 존재한다(y=x 라인 아래에는 데이타가 존재할 수 없으므로 질의 영역을 사각형으로 생각할 수 있다). 따라서, N-node에서는 T-value 범위가 질의 영역과 겹치고, 선택 조건 애트리뷰트들의 값의 범위가 주어진 값의 범위와 겹칠 때 자식 노드를 방문하면 된다.

이러한 방문을 통해, A-node에서 부분 결과가 생성되며, 최종 결과는 이러한 부분 결과들을 병합하여 생성한다. 예를 들어, 주어진 조건에서 시간 구간은 [7, 9]이고 애트리뷰트 범위는([1, 5], [3, 5])라고 하자. 그러면, 그림 4(d)에서 조건을 만족하는 경로는 I_3-I_2-A_2와 I_3-I_2-A_3이다. 이 두 경로로부터의 부분 결과는 (<[5, 6], 1>), (<[7, 9], 1>)이며 이들은 (<[5], 1>, <[7], 0>) (<[7], 1>, <[10], 0>)로 표현될 수 있다. 이들을 병합하면 최종 결과는 (<[5], 1>, <[10], 0>)이 된다. 다음은 집계 처리를 위한 알고리즘 *I-ComputeAgg*를 나타낸다. 주어진 조건은 R과 I로 구성되며, R은 선택 조건 애트리뷰트들의 값의 범위, I는 시간 구간이다.

```
Algorithm: N-ComputeAgg (N-node Nc, range-condition attributes' range R, time interval I)
  if (Nc is an internal N-node) {
     for (each record E in Nc whose time interval intersects I and E.I intersects R) {
        call N-ComputeAgg(E.C, R, I) and acquire a partial result; }
     merge partial results and return the merged result;
  } else {
     for (each record E in Nc whose time interval intersects I and E.I intersects R) {
        visit the A-node pointed by E.C and acquire partial result; }
     merge partial results and return the merged result;
  }
```

4. elTA-tree: 핫-스팟 해결을 위한 개선된 ITA-tree

4.1 핫-스팟 문제

ITA-tree에서 레코드를 T-value에 따라 인덱싱할 때, 핫-스팟 문제가 발생할 수 있다. 즉, 동일한 시간-구간을 가지는 많은 레코드들은 동일한 T-value 값을 가지게 된다. 기존의 B^+-tree에서는 단순히 디스크 페이지들

을 연결하는 체인(chain) 기법으로 쉽게 해결할 수 있기 때문에 이 문제는 중요하지 않았다. 그러나, ITA_tree에서 체인 기법은 효율적이지 않다. 디스크 페이지들을 연결하게 되면, 디스크 접근 회수가 아주 많아지게 된다. 만약 동일한 T-value를 가진 레코드들을 동일한 노드에 저장하고 이 노드를 위한 하나의 T-value만 저장하면, 노드에 T-value를 따로 저장할 필요가 없게 된다 (내부 노드에서 이 T-value는 '범위'가 아니라 하나의 '값'이다. B^+-tree에서는 내부 노드의 키 값은 키 값의 범위를 의미한다). 그리고, 동일한 T-value 값을 가지는 노드들로 B-tree와 같은 트리 구조를 생성할 수 있다. 따라서, 디스크 공간을 절약할 수 있으며, 집계 처리를 위한 디스크 접근 회수를 줄일 수 있다.

본 절에서는 핫-스팟을 유발하는 데이타를 효과적으로 관리하기 위한 새로운 노드 타입인 RA-node(Range-condition attributes-based Aggregate node)를 제안한다. RA-node의 레코드에는 T-value는 저장되지 않고 선택 조건 애트리뷰트만 저장된다. 핫-스팟 문제가 발생하면, 핫-스팟 문제를 유발한 T-value를 가지는 레코드들을 추출하여 RA-node를 생성한다. 본 논문에서는, 핫-스팟을 인식하는 한계를 노드 용량의 50%로 설정하였다. 이후에는, RA-node의 T-value와 동일한 T-value를 가지는 새 레코드들은 이 RA-node에 삽입된다.

4.2 eITA-tree의 구조

향상된 트리 구조인 *eITA-tree(extended ITA-tree)*는 추가적으로 RA-node들을 가진다. RA-node의 각 레코드는 선택 조건 애트리뷰트들과 집계 값을 저장한다. 자식 노드가 RA-node임을 표시하기 위해 말단 N-node도 수정된다. 말단 N-node에는 *Flag*가 추가된다. *Flag*의 값이 T(True)이면 자식 노드는 RA-node이다. *Flag*의 값이 F(False)이면 자식 노드는 A-node이다. *Flag*의 값이 T(True)인 레코드(자식 노드가 RA-node)를 *상승(promoted) 레코드*라 하며, RA-node를 생성하고 상승 레코드를 만드는 과정을 *상승(promotion)*이라 한다. 상승 레코드의 자식 노드는 RA-node이며 T-value는 RA-node에 있는 레코드들의 T-value이다. 비상승 레코드의 자식 노드는 A-node이며, T-value는 값이 아닌 T-value 범위의 상한선이다.

그림 5는 eITA-tree의 예제이다. 그림 4(d)에 표 2의 레코드 8, 9를 삽입하여 RA-node가 생성되는 과정을 나타낸다.

레코드 8을 삽입할 때 공간이 없으므로 A_2 가 분할되어야 한다. 이 때, T-value가 35인 레코드가 A-node 용량의 50% 이상을 차지하고 있다. 이 레코드들을 추출하여 RA-node를 생성한다. 그림 5에는 새로운 RA-node들 RA_1, RA_2, RA_3가 있다. RA-node들의 레코드

표 2 시간 데이타 예제

No	선택 조건 애트리뷰트		시간 구간	접속 기기 수
	타입	영역		
8	3	6	[3, 5]	7
9	2	8	[3, 5]	4

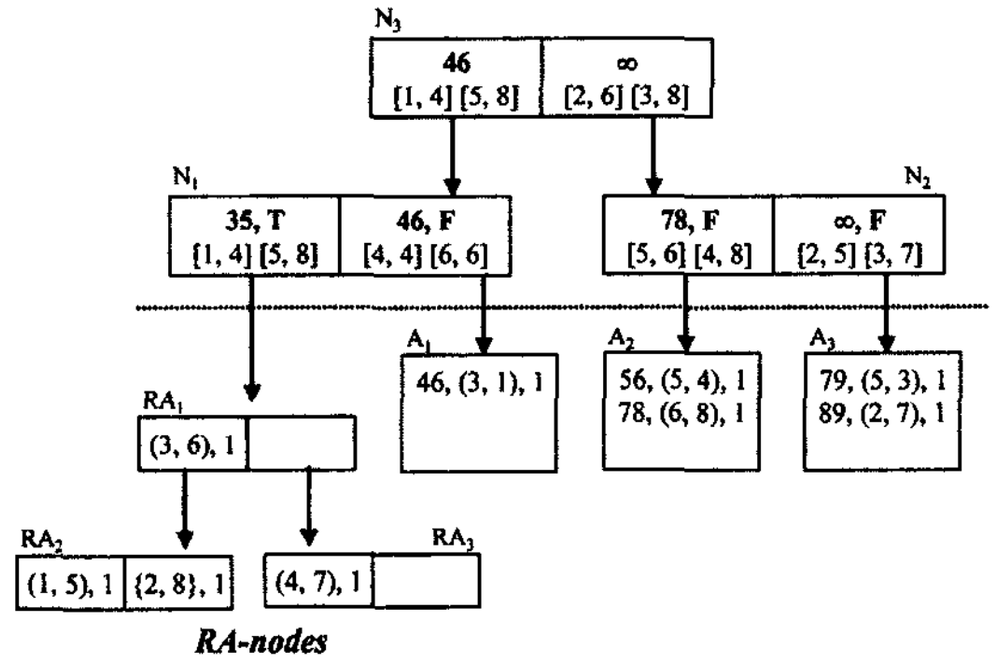


그림 5 eITA-tree 생성 과정

에는 선택 조건 애트리뷰트들과 집계 값들이 저장된다 (RA_3의 "1"은 선택 조건 애트리뷰트(4, 7)의 집계값이다). 그리고, RA-node들을 가리키는 새로운 레코드가 말단 N-node N_1에 삽입되었다. 이 레코드의 Flag는 T이며, 새로 생성된 RA-node들의 루트 RA_1을 가리킨다. 이러한 과정을 *상승*이라 하며, RA_1을 가리키는 N_1의 새로운 레코드를 *상승 레코드*라 한다. 이후에 추가되는 T-value가 35인 레코드들은 RA-node들에 삽입된다. 따라서, 표 2의 레코드 9는 RA-node RA_2에 삽입되었다.

4.3 수정된 알고리즘

4.3.1 삽입

새로이 RA-node와 상승 레코드가 추가되었으므로, 3.3절의 삽입 알고리즘을 약간 수정해야 한다. 새 레코드와 동일한 T-value를 가지는 상승 레코드가 존재할 경우, 그 레코드가 가리키는 RA-node에 새 레코드를 추가한다. 그렇지 않은 경우에는 ITA-tree의 삽입 알고리즘과 동일하다. 자세한 알고리즘은 생략한다.

4.3.2 분할

RA-node의 분할은 B-tree의 분할과 유사하다. RA-node는 선택 조건 애트리뷰트들의 범위에 따라 분할된다.

3.3절에서 설명했듯이, A-node의 분할은 T-value에 따라 이루어지며 B^+-tree의 분할과 유사하다. 그러나, 핫-스팟 문제가 있을 때는 RA-node를 생성한다. 즉, A-node를 분할할 때, 동일한 T-value를 가지는 레코드의 수가 한계치 K' 이상이면, 해당되는 레코드들을 모두 추출하여 RA-node들을 생성한다. 다음은 A-node를 분할하는 알고리즘을 기술한 것이다.

```
Algorithm: A-Split
  for (each distinct T-value K) {
    calculate the number of records, num(K); }
  if (there is a T-value K' whose num(K') is greater than half
  of the capacity of the A-node) {
      extract a set of record whose T-values equals K' and
      construct new RA-nodes using them;
  } else {
      split the A-node into two A-nodes to have
      approximately the same number of records;
  }
```

내부 N-node의 분할은 ITA-tree에서와 동일하다. 그러나, 말단 N-node는 상승 레코드와 비상승 레코드를 동시에 가질 수 있으므로, 상승 레코드를 고려하여 분할해야 한다. 상승 레코드는 비상승 레코드가 가리키는 A-node가 분할될 때 생성되며, 그 비상승 레코드의 앞에 삽입된다. 즉, 말단 N-node에 상승 노드가 있다면, 그 노드에는 반드시 상승 노드의 T-value를 포함하는 T-value 범위를 가지는 비상승 레코드가 존재해야만 한다. 따라서, 말단 N-node가 분할되었을 때, 새로운 노드의 마지막 레코드가 상승 레코드라면 비상승 레코드가 추가되어야만 한다. 이 비상승 레코드의 T-value는 마지막 상승 레코드의 T-value를 포함할 수 있는 T-value 범위로 설정된다. 그리고 새로운 A-node가 할당되며, 여기에는 분할된 다른 말단 N-node의 첫번째 비상승 노드가 가리키는 A-node에서 T-value 범위에 속하는 레코드들을 추출하여 저장한다. 예를 들어, 그림 5에서 말단 N-node I_2가 I_2와 I_4로 분할되었고, RA_1을 가리키는 상승 레코드는 I_4에 저장되었다고 하자. 그러면, 새로운 비상승 레코드 E를 I_4에 삽입하고, E의 T-value는 상승 레코드를 포함할 수 있는 값 35로 설정한다. 또한, 새로운 A-node A_4를 E에 할당(E.C가 가리킴)하고, A_1에서 범위 E.K (35 이하)에 속하는 레코드들을 추출하여 A_4에 저장한다. 다음은 말단 N-node의 분할 알고리즘 *N-Deep-Split*를 나타낸다.

```
Algorithm: N-Deepest-Split
  split node N into N1 and N2 to have approximately same
  number of records;
  if (the last record X in N1 is a promoted record) {
      insert a new unpromoted record E into N1;
      allocate a new A-node and set E.C as the new A-node;
      set E.K to X.K;
      select the first unpromoted record E' from N2;
      extract records that covered by X.K from an A-node that
      is indicated by E'.C;
      insert extracted records into the A-node that is indicated
      by E.C;
  }
```

5. 분 석

이 장에서는 ITA-tree의 공간 복잡도를 분석한다. 기존에 유효 시간과 하나 이상의 선택 조건 애트리뷰트를 지원하는 시간 집계 처리 기법은 존재하지 않았으므로, SB-tree를 간단히 확장한 eSB-tree와 비교하도록 한다.

SB-tree는 레코드가 삽입될 때 불변 구간에 따라 집계값을 저장한다. 일반 시간 집계를 처리하기 위해서는, 각 불변 구간마다 선택 조건 애트리뷰트들과 집계값들이 저장되어야 한다. 따라서 선택 조건 애트리뷰트들과 집계값들을 B-tree에 저장하고, 이러한 B-tree를 SB-tree에 있는 레코드마다 할당한다. 또한, SB-tree의 노드에 저장되는 레코드에는 하위 노드들의 각 선택 조건 애트리뷰트들의 범위를 저장한다. 이렇게 시간 구간마다 선택 조건 애트리뷰트들과 집계값을 저장하도록 확장된 SB-tree를 *eSB-tree*라 한다. 그림 6은 eSB-tree의 예로써 표 3의 예제 데이타로부터 생성되었다.

본 논문에서는 최악 경우(worst-case) 공간 복잡도와 공간 효율성의 예측 가능성(predictability)을 분석한다. 공간 효율성의 예측 가능성은 디스크-기반 접근 방법들을 평가하는데 필요한 중요한 기준이다[14].

표 3 시간 데이타 예

No	선택 조건 애트리뷰트		시간 구간	접속 기기 수
	타입	지역		
1	1	5	[3, 5]	3
2	3	1	[4, 6]	1
3	4	7	[3, 5]	6
4	5	3	[7, 9]	9
5	2	7	[8, 9]	20

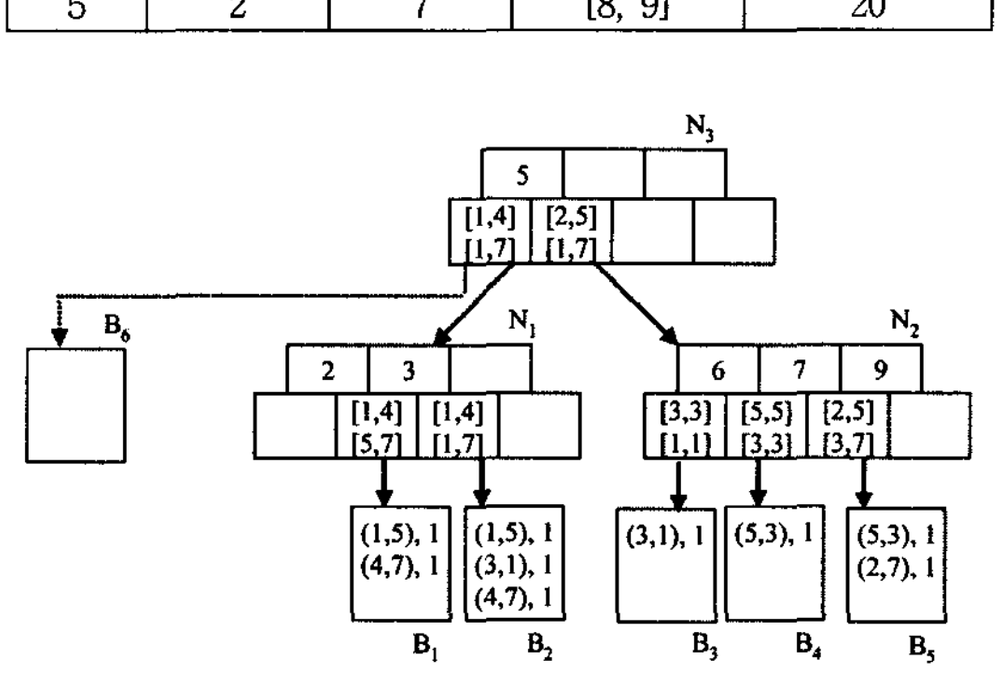


그림 6 eSB-tree 예

정리1. 최악 경우, eSB-tree에서 한 레코드를 삽입할 때 $(B-2)+2*(h-1)*(B-1)$개의 복제가 필요하다. B는 노드에 저장 가능한 최대 원소의 수(degree)이며, h는 트리의 높이이다.

증명: 새로운 레코드가 eSB-tree에 삽입된다고 하자. 최악의 경우는 새 레코드의 시간구간이 루트 노드의 첫 번째와 마지막 레코드의 시간 구간과는 교차하고, 나머지 레코드의 시간 구간은 포함할 때이다. 이 때, 첫 번째와 마지막 레코드 외의 레코드가 가리키는 B-tree들에 새 레코드를 복제한다. 노드에 저장 가능한 최대 원소의 수는 B이므로 (B-2)개의 레코드가 복제된다. 루트 노드의 첫 번째와 마지막 레코드의 시간 구간은 새 레코드의 시간 구간과 교차하므로 이 레코드로부터 시작되는 경로들의 자식 노드를 방문한다. 자식 노드에서도 레코드들의 시간 구간들과 새 레코드의 시간 구간을 비교한다. 이 때, 최악의 경우는 노드에서 첫 번째 혹은 마지막 레코드(둘 다 교차하는건 루트 노드에서만 가능)의 시간 구간이 새 레코드의 시간 구간과 교차하고 나머지 (B-1)개 레코드의 시간 구간은 포함되는 것이다. 이 때, (B-1)개의 레코드가 복제되며 시간 구간이 교차한 레코드가 가리키는 자식 노드를 따라서 동일한 과정을 반복하게 된다. 즉, 경로 상의 노드들에서는 (B-1)개의 복제가 발생한다. 루트를 제외하면 각 경로 상의 노드는 (h-1)개이다. 따라서, 두 경로에서는 2*(h-1)*(B-1)개의 복제가 생긴다. 루트 노트의 복제까지 합하면, 최악의 경우 한 개의 레코드가 삽입될 때 생기는 복제 레코드는 (B-2)+2*(h-1)*(B-1)개 이다. ■

N-node의 구조는 B-tree와 유사하므로, 레코드가 N개일 때 트리의 높이 h는 O(log N)에 비례한다. 따라서, 하나의 레코드는 O(B*log N)만큼 복제되며, eSB-tree의 최악 경우 공간 복잡도는 O(B*N*log N)이 된다.

ITA-tree와 eITA-tree에서는 복제가 필요 없다. 즉, 최악 경우에도 항상 하나의 레코드만 저장된다. 따라서, 최악 경우 공간 복잡도는 O(N)이다.

eSB-tree에서 노드에 저장된 각 시간 구간은 T-value와 집계 값을 저장할 B-tree에 연결된다. 따라서, 하나의 시간 구간을 위해서는 최소한 하나의 디스크 페이지가 필요하다. 시간 분포가 조밀하지 않을 경우, 많은 디스크 페이지에는 몇 개의 레코드들만 존재하게 된다. 따라서, 조밀하지 않은 시간 분포일 경우 최소 공간 활용도를 예측할 수 없다. 즉, eSB-tree의 최소 공간 사용량을 예측할 수 없다.

핫-스팟 문제가 있을 때, ITA-tree에서는 체인 기법을 사용해야 하므로 A-node의 공간 활용도를 정확하게 예측하기는 어렵다. 그러나, 체인된 A-node에서 공간 활용도를 예측할 수 없는 것은 가장 마지막 노드이며, 이러한 노드들의 수는 말단 N-node에 있는 레코드들의 수와 같다. 따라서, 평균적인 공간 활용도는 예측할 수 있으며, ITA-tree는 50%를 보장한다.

eITA-tree에서는 공간 활용도를 예측할 수 없는 노드가 있으나, 이들의 수를 노드의 종류에 따라 예측할 수 있다. 따라서, 최소 공간 활용도를 예측할 수 있다. A-node와 RA-node에 있는 레코드의 수는 최소한 A-node 용량의 33%이다(정리 2). N-node의 최소 공간 활용도는 평균 50%이다. N-node들의 구조는 B+-tree와 유사하므로, N-node에서는 50% 공간 활용도가 보장된다. 공간 활용도를 예측할 수 없는 A-node들도 존재할 수 있으나, 최악 경우에도 이러한 A-node들의 수는 예측될 수 있다. 이러한 특성들로 인해, 최악 경우의 eITA-tree의 공간 사용량을 예측할 수 있다.

정리2. RA-node나 공간 사용량이 예측 가능한 A-node에 있는 레코드의 수는 A-node 저장량의 33% 이상이다.

증명: A-node는 레코드들의 T-value들에 따라서 분할되며, 동일한 T-value들도 존재할 수 있다. 분할된 공간 사용량이 예측 가능한 A-node들은 최소한 33% 이상의 공간을 사용한다. 예측할 수 없는 A-node는 두 가지 경우에서 발생한다. 첫 번째는 핫-스팟 문제를 해결하기 위해 A-node에서 레코드를 추출하여 RA-node들을 생성할 때이다. 두 번째는 말단 N-node 분할에서 비상승 레코드를 추가하기 위해 새로운 A-node를 생성하고 기존의 A-node에 있는 레코드들을 추출한 경우이다. 먼저 첫 번째 경우를 살펴보자. 레코드가 추출된 A-node의 레코드 수는 예측할 수 없다. 이러한 추출은 핫-스팟 문제가 발생했을 때이며, 추출된 레코드들은 새로운 RA-node에 삽입되었다. 따라서, A-node와 새로운 RA-node에 있는 레코드의 수는 A-node의 저장량과 동일하다. 따라서, 평균 공간 활용량은 약 50%이다. 두 번째 경우는 말단 N-node를 분할할 때 발생한다. 이 경우에는, 예측할 수 없는 공간 활용도를 가진 A-node의 수는 예측할 수 있다(정리3). 따라서, RA-node와 예측 가능한 A-node의 최소 공간 활용도는 30%이다. ■

정리3. N-node 분할에서 생성된 공간 활용도를 예측할 수 없는 A-node의 수는 최악 경우 $2*(N_d-1)$이다. 여기서, N_d는 말단 N-node의 수이다.

증명: 말단 N-node의 분할 후에, 새로 생성된 N-node의 마지막 레코드가 상승 레코드라면, 비상승 레코드를 그 노드에 삽입한다. 새로운 A-node를 추가된 레코드에 할당하며, 이 A-node에는 분할된 다른 말단 N-node의 첫 번째 비상승 레코드가 가리키는 A-node로부터 추출된 레코드를 삽입한다. 이 분할에서, 새로운

A-node가 생성되었으며, 두 A-node의 공간 활용도는 예측할 수 없게 되었다. 즉, 말단 N-node가 분할될 때마다 최대 2개의 A-node들의 공간 활용도를 예측할 수 없게 된다. 말단 N-node는 말단 N-node가 분할될 때마다 하나씩 증가하므로, 말단 N-node의 개수가 N_d라면, 노드 분할은 N_d-1번 발생한 것이다. 따라서, 공간 활용도를 예측할 수 없는 A-node의 수는 최악 경우 2*(N_d-1)이다. ■

표 4 공간 사용량과 활용도 분석

	최악 경우 공간 복잡도	예측 불가능한 공간 활용도의 노드 수
eSB-tree	O(B*N*log N)	예측 불가능
ITA-tree	O(N)	말단 N-node의 레코드 수
eITA-tree	O(N)	2 * (말단 N-node의 수 - 1)

6. 성능 실험

본 논문에서는 디스크 사용량과 집계 처리 시간을 기준으로 eSB-tree, ITA-tree, eITA-tree의 성능을 비교한다.

실험을 위해 먼저 시간 데이타를 생성하고, eSB-tree, ITA-tree, eITA-tree를 생성한다. 모든 애트리뷰트는 0 이상의 정수 값을 가진다. 실험에 사용되는 패러미터로는 선택 조건 애트리뷰트의 값의 범위, 시간 구간의 범위, 레코드 수와 선택 조건 애트리뷰트의 수가 있다. 디스크 사용량에 대한 실험은 선택 조건 애트리뷰트와 시간 구간의 밀도를 달리 하여 수행하였다. 실험에 사용된 밀도에 따른 조합은 *wRwI*, *nRwI*, *wRnI*이다. 여기서, *wR*은 넓은 애트리뷰트 범위로써 [0, 1000000]이다. *nR*은 좁은 애트리뷰트 범위로써 [0, 10]이다. 마찬가지로, *wI*와 *nI*는 넓은 시간 구간과 좁은 시간 구간을 의미하며 각각 [0, 1000000], [0, 10]이다. 레코드의 수는 50000, 100000, 200000, 250000, 300000, 350000, 400000, 450000, 500000를 대상으로 한다. 시간 구간과 선택 조건 애트리뷰트의 값은 균일 분포를 이룬다고 가정한다. 또한, 시간 구간의 시작 시간과 끝 시간도 균일 분포를 이룬다고 가정한다. 선택 조건 애트리뷰트의 수는 2, 4, 6개로 하였다(d2, d4, d6으로 표시). 그림 7, 8, 9는 선택 조건 애트리뷰트의 수를 2로 고정한 것이다. 애트리뷰트와 포인터의 크기는 모두 4바이트로 한다.

먼저, 디스크 사용량에 관하여 알아본다. 그림 7은 조밀하지 않은 시간 구간에서의 디스크 사용량을 나타낸다. ITA-tree와 eITA-tree가 eSB-tree에 비해 디스크를 적게 사용한다는 것을 알 수 있다. ITA-tree와 eITA-tree는 조밀하지 않은 시간 구간에서 거의 같은 디스크 페이지를 사용함을 알 수 있다. 레코드 수가 500000일 때, ITA-tree(eITA-tree)는 eSB-tree가 사용하는 디스크 페이지의 1/100 정도로 적은 디스크 페이지를 사용함을 알 수 있다.

그림 8은 시간 구간은 조밀하고, 선택 조건 애트리뷰트는 조밀하지 않은 경우에 대한 디스크 사용량을 나타낸다. 시간 구간의 범위가 [0,10]이므로 100개의 T-

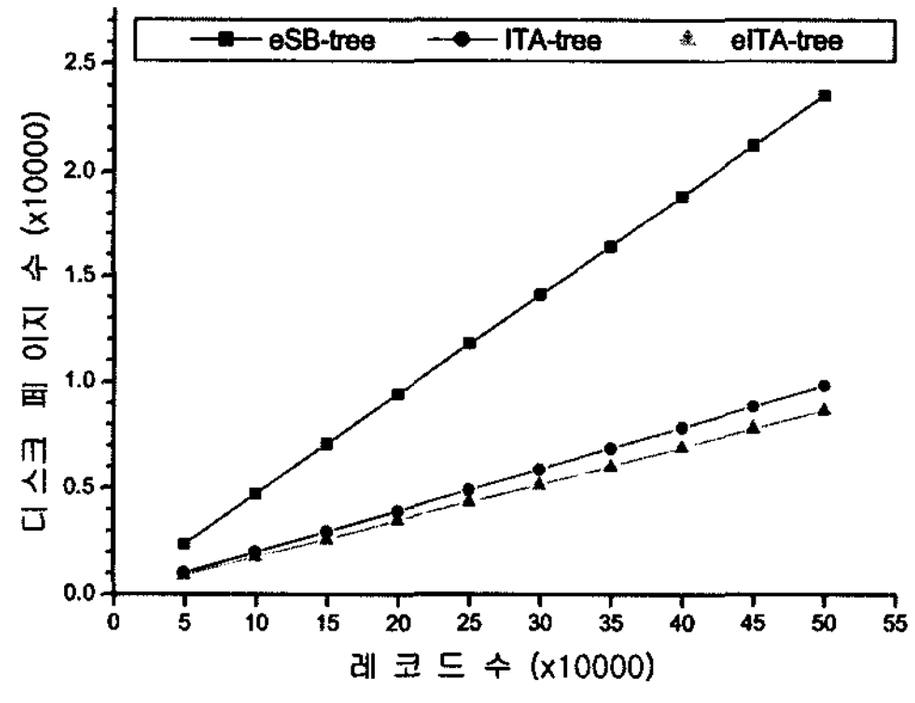
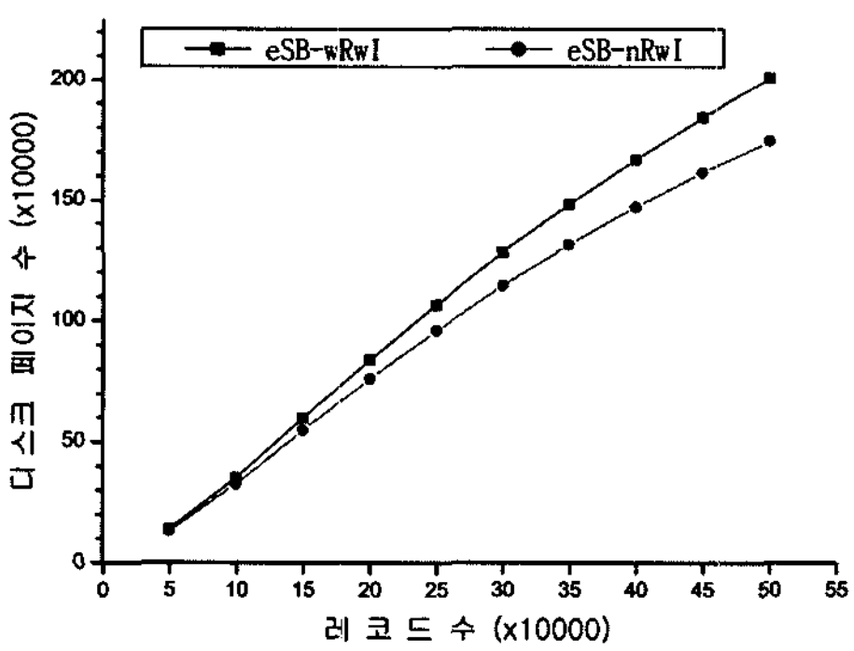


그림 8 조밀한 시간 구간과 조밀하지 않은 선택 조건 애트리뷰트에 대한 디스크 사용량



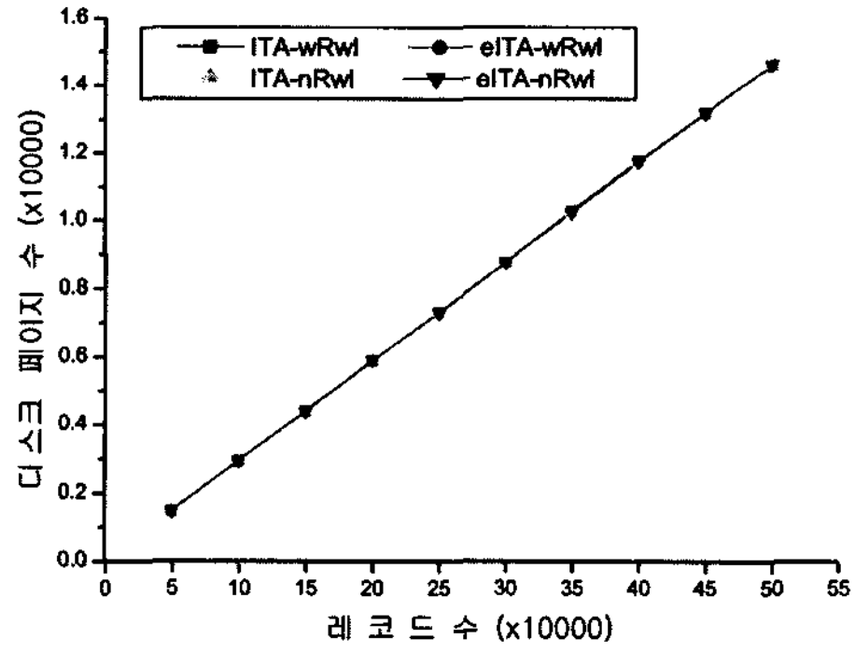


그림 7 조밀하지 않은 시간 구간에 대한 디스크 사용량

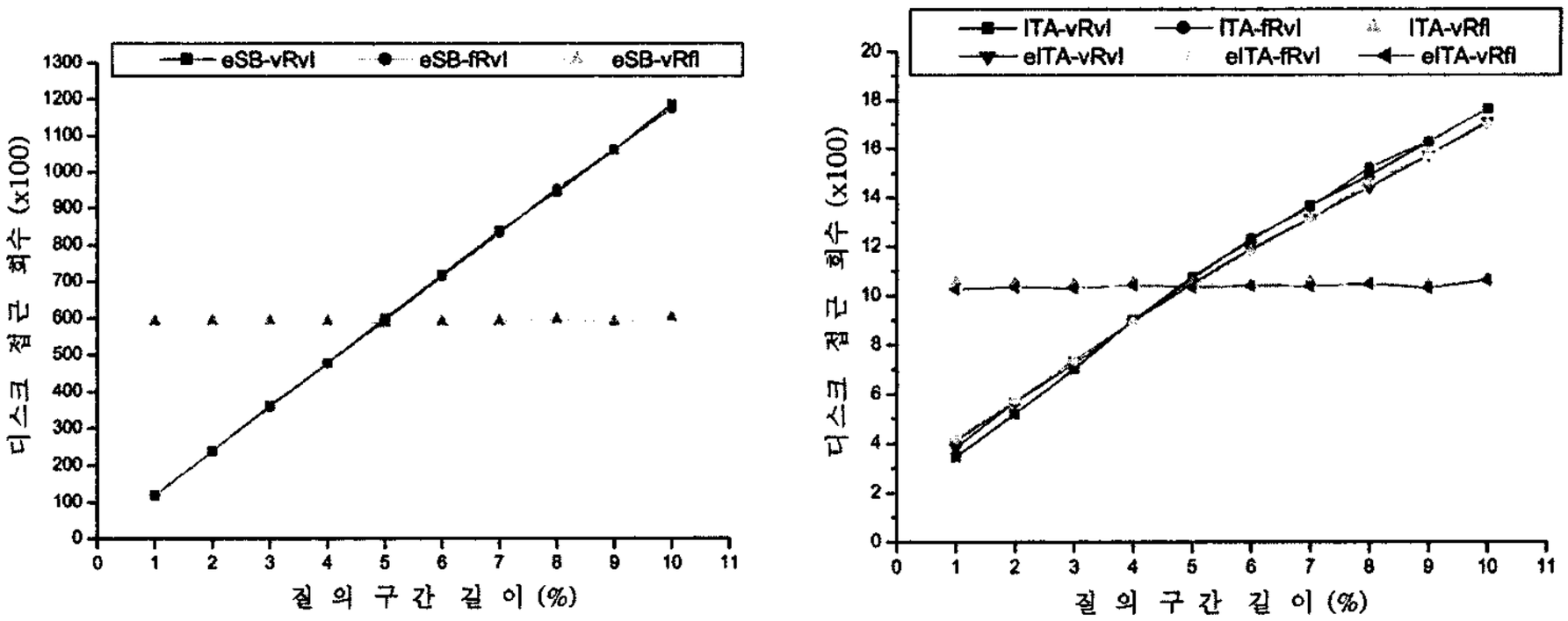


그림 9 시간 집계 처리에 필요한 디스크 접근 회수

value들이 존재한다. 따라서, 레코드가 500000개일 때, 500개의 레코드가 동일한 T-value를 가지게 된다. eITA-tree와 ITA-tree는 eSB-tree가 사용하는 디스크 페이지의 1/4 정도로 적은 디스크 페이지를 사용함을 알 수 있다.

그림 9는 일반 시간 집계 처리에 필요한 디스크 접근 회수를 나타낸다. 레코드 수는 50000으로 하였으며, 시간 구간과 선택 조건 애트리뷰트는 조밀하지 않게 하였다. 조건의 애트리뷰트 범위와 시간 구간의 길이는 전체 범위의 1, 2, 3, 4, 5, 6, 7, 8, 9, 10%로 하였다. 실험에 사용된 조합은 3가지가 있다. *vRvI*는 애트리뷰트 범위와 시간 구간을 동일하게 변화시키는 경우이며, *fRvI*는 애트리뷰트 범위는 5%로 고정시킨 경우이며, *vRfI*는 시간 구간을 5%로 고정시킨 경우이다. 결과에서 eSB- tree가 eITA-tree와 ITA-tree보다 60배나 많이 디스크에 접근함을 알 수 있다. eITA-tree와 ITA-tree는 거의 동일하게 디스크에 접근함을 알 수 있다. 그리고, 디스크 접근 회수에 큰 영향을 미치는 것은 시간 구간이며, 애트리뷰트 범위에 의한 영향은 미미하다는 것을 알 수 있다.

그림 10은 선택 조건 애트리뷰트 개수에 대한 디스크 사용량을 나타낸다. 시간 구간과 선택 조건 애트리뷰트는 조밀하지 않다. 디스크 사용량은 선택 조건 애트리뷰트의 개수에 비례하여 증가함을 알 수 있다. 레코드 개수가 500000일 때, eITA-tree와 ITA-tree는 eSB-tree가 사용하는 디스크 페이지의 1/100 정도를 사용함을 알 수 있다.

그림 11은 조밀한 시간 구간과 조밀하지 않은 선택 조건 애트리뷰트에 대한 디스크 사용량을 나타낸다. eITA-tree와 ITA-tree가 eSB-tree의 디스크 사용량의 1/3 정도로 적은 디스크를 사용함을 알 수 있으며, eITA-tree가 ITA-tree보다 약간 많은 디스크를 사용함을 알 수 있다.

그림 12는 선택 조건 애트리뷰트 개수가 다를 때 일반 시간 집계를 처리하는 데 필요한 디스크 접근 회수를 나타낸다. 레코드 개수는 50000, 시간 구간과 선택 조건 애트리뷰트는 조밀하지 않게 하였다. 조건의 애트리뷰트 범위와 시간 구간의 길이는 동시에 변화시키면서 실험을 수행하였다. eSB-tree의 디스크 접근 회수는 여러 상황에서 거의 동일하며, eITA-tree와 ITA-tree보다는 30배 더 많다는 것을 알 수 있다.

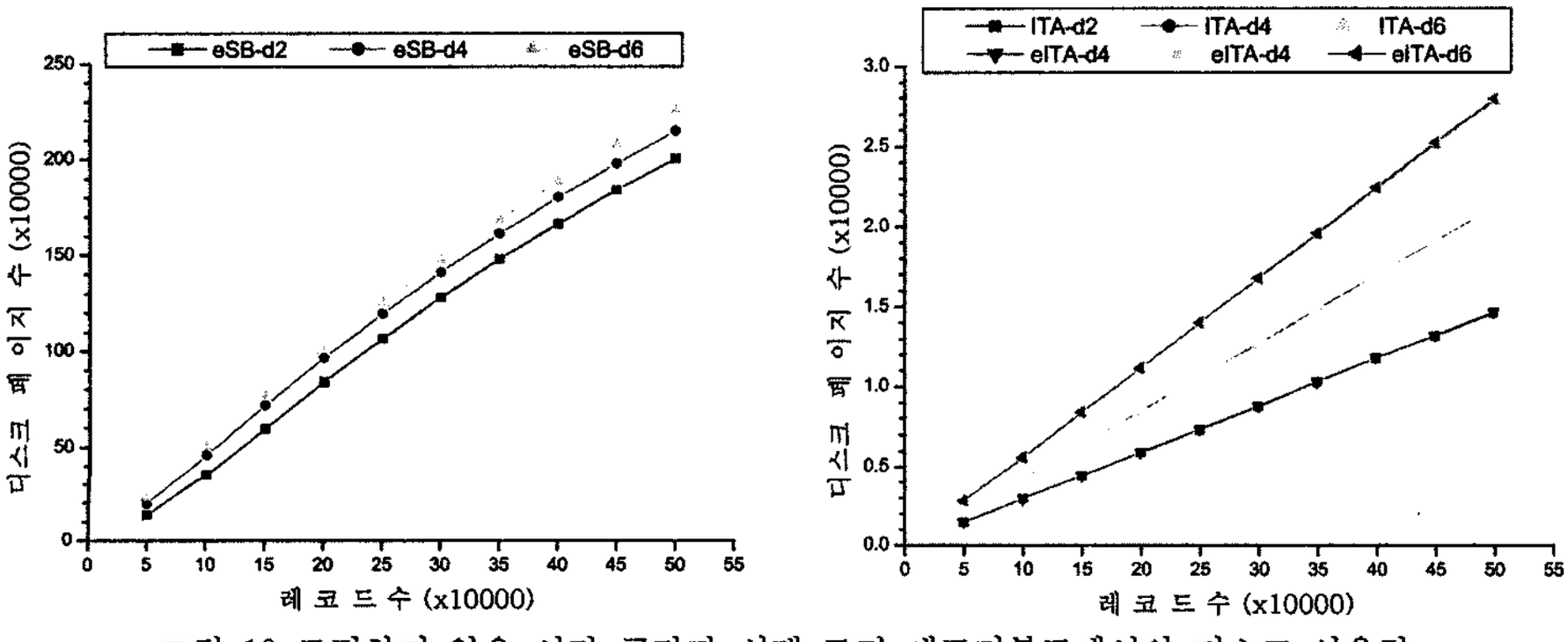


그림 10 조밀하지 않은 시간 구간과 선택 조건 애트리뷰트에서의 디스크 사용량

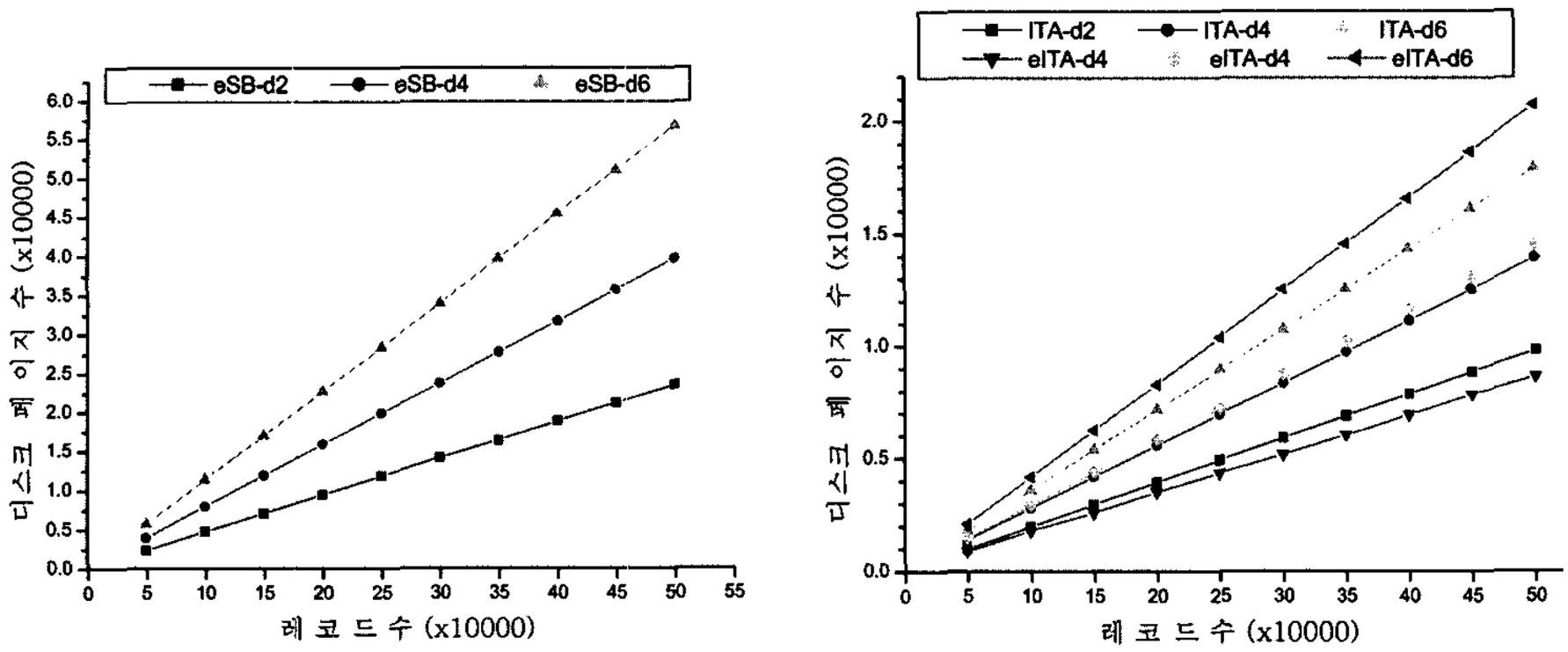


그림 11 조밀한 시간 구간과 조밀하지 않은 선택 조건 애트리뷰트에 대한 디스크 사용량

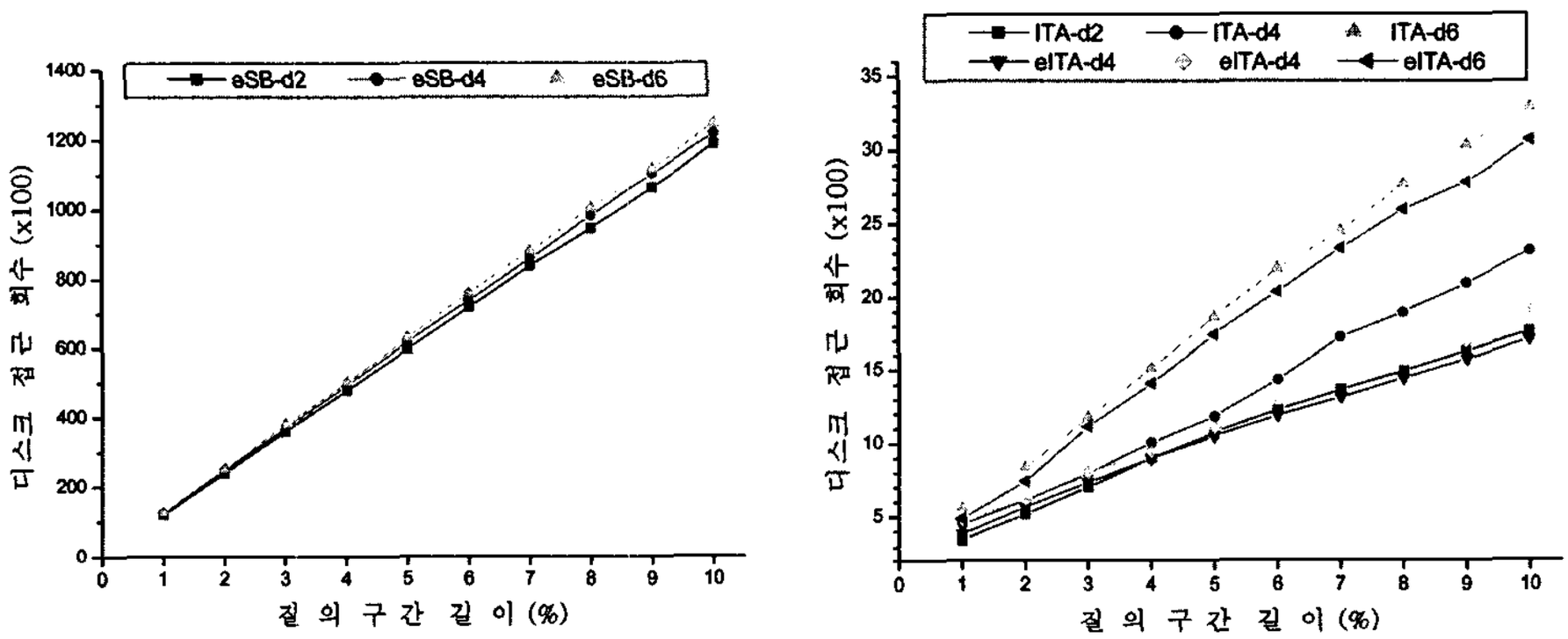


그림 12 시간 집계 처리에 필요한 디스크 접근 회수

7. 결 론

집계 연산은 릴레이션 전체 혹은 그 일부를 구성하는 레코드들에 적용되어 전체 값을 계산하거나 대표 값을 선택하는 연산으로, 다양한 응용 분야에서 많이 사용된다. 그러나, 시간 애트리뷰트를 포함하는 시간지원 데이타베이스에서의 집계 연산은 시간 애트리뷰트를 고려하지 않은 기존의 집계 연산과는 큰 차이가 있다. 따라서, 기존의 집계 연산 처리 기법은 시간 집계 연산에는 사용될 수 없다.

본 논문에서는 시간 구간과 하나 이상의 선택 조건 애트리뷰트를 포함하는 일반 시간 집계 연산(*General Temporal Aggregation*)을 처리하는 기법을 제안하였다. 기존의 연구들은 시간 애트리뷰트만 고려하거나, 하나의 선택 조건 애트리뷰트와 단순 증가하는 트랜잭션 시간만 지원하는 문제점이 있다. 본 논문에서는 일반 시간 집계를 처리할 수 있는 새로운 구조로써 *ITA-tree*와 이를 이용한 처리 기법을 제안하였다. 또한, 핫-스팟 문제를 극복할 수 있는 개선된 방법으로 *eITA-tree*를 제안하였다. ITA-tree는 균형화된 트리로서 효율적으로 레코드를 삽입하고 시간 집계 연산을 처리할 수 있다. 본 논문에서는 분석과 실험을 통해 질의 처리 시간과 디스크 사용량을 측정하였으며, 일반 시간 집계 연산을 처리할 수 있도록 SB-tree를 간단히 확장한 eSB-tree와 비교하였다. 결과에서, ITA-tree가 eSB-tree보다 우수한 성능을 보였다. 또한, 핫-스팟 문제가 있을 때는 eITA-tree가 ITA-tree보다 우수한 디스크 활용도를 보였다.

향후, 질의에 따라 선택 조건 애트리뷰트들을 여러 가지로 조합하고 각각의 T-value에 따라 여러 ITA-tree로 관리할 때, 시간 집계 연산을 어떻게 처리할 것인가에 대한 연구가 필요하다. 또한, 시간지원 데이타베이스가 대형화되어 병렬 시스템이나 분산 시스템이 활용되고 있으므로, 이러한 환경에서 제안된 방법들을 활용할 수 있도록 확장하는 방안에 대한 연구가 필요하다.

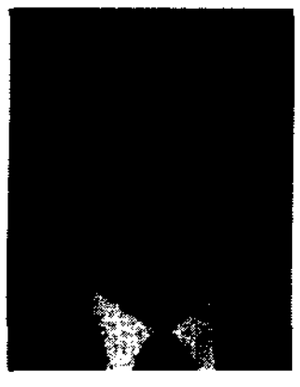
참 고 문 헌

[1] G.Ozsoyoglu and R.T.Snodgrass. *Temporal and Real-Time Databases: A Survey*. IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.4, pages 513-532, 1995.

[2] H.Gregersen, and C.S.Jensen. *Temporal Entity-Relationship Models - A Survey.* Technical Report R-96-2039, Aalborg University, 1996.
[3] J.Melton. *SQL/Temporal.* July 1996. (ISO/IEC JTC 1/SC 21/WG 3 DBL-MCI-0012.)
[4] R.T.Snodgrass, I.Ahn, G.Ariav and et al. *The TSQL2 Temporal Query Language.* Kluwer Academic Publishers, 1995.
[5] 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.
[6] J.S.Kim and M.H.Kim. *An Effective Data Clustering Measure for Temporal Selection and Projection Queries.* Decision Support Systems, Vol. 30, No. 1, pages 33-50, 2000.
[7] 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, pages 282-292, 1994.
[8] B.Salzberg and V.J.Tsotras. *A Comparison of Access Methods for Time Evolving Data.* ACM Computing Surveys, Vol. 31, Issue. 2, pages 158-221, 1997.
[9] V.J.Tsotras, B.Gopinath and G.W.Hart. *Efficient Management of Time-Evolving Databases.* IEEE Transactions of Knowledge and Data Engineering, Vol. 7, No. 4, pages 591-608, 1995.
[10] K.Torp, C.S.Jensen and R.T.Snodgrass. *Effective Timestamping in Databases. Journal of Very Large Data Bases.* Vol. 8, Issue. 3, pages 267-288, 2000.
[11] V.J.Tsotras, N.Kangelaris. *The Snapshot Index: An I/O-optimal access method for timeslice queries.* Journal of Information Systems, Vol. 20, No. 3, pages 237-260, 1995.
[12] J.A.G.Gendrano, B.C.Huang, J.M.Rodrigue, B.Moon and R.T.Snodgrass. *Parallel Algorithms for Computing Temporal Aggregates.* International Conference on Data Engineering, pages 418-427, 1999.
[13] 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.
[14] N.Kline, and R.T.Snodgrass. *Computing Temporal Aggregates.* International Conference on Database Engineering, pages 222-231, 1995.
[15] B.Moon, I.F.V.Lopez and V.Immanuel. *Efficient Algorithms for Large Temporal Aggregation.* IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 3, pages 744-759, 2003.
[16] 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.
[17] P.A.Tuma. *Implementing Historical Aggregates in TempIS.* Master's Thesis, Wayne State University, Nov. 1992.
[18] X.Ye and J.A.Keane. *Processing Temporal Aggregates in Parallel.* International Conference on Systems, Man, and Cybernetics, pages 1373-1378, 1997.
[19] J.Yang and J.Widom, *Incremental Computation and Maintenance of Temporal Aggregates.* International Conference on Data Engineering, pages 51-60, 2001.
[20] D.Zhang, A.Markowetz, V.Tsotras, D.Gunopulos and B.Seeger. *Efficient Computation of Temporal Aggregates with Range Predicates.* Principles of Database Systems, 2001.
[21] B.Seeger and H.P.Kriegal. *Techniques for Design and Implementation of Efficient Spatial Access Methods.* International Conference on Very Large Data Bases, pages 360-371, 1988.
[22] H.K.Park, S.T.kang, M.H.Kim and K.W.Min. *On Indexing Method for Current Positions of Moving Objects.* Journal of the Korea Open GIS Association,Vol. 5, No. 1, P. 65-74. 2003, 6.
[23] J.W.Song, K.Y.Whang, Y.K.Lee, M.J.Lee and S.W. Kim. *Spatial Join Processing Using Corner Transformation.* IEEE Transactions of Knowledge and Data Engineering, Vol. 11, No. 4, pages 688-695, 1999.
[24] V.Gaede and O.Gunther. *Multidimensional Access Methods. ACM Computing Surveys.* Vol. 30, Issue. 2, pages 170-231, 1998.

강 성 탁
1996년 한국과학기술원 전산학과(학사) 1998년 한국과학기술원 전산학과(석사) 2004년 한국과학기술원 전산학전공(박사). 현재 디지털아리아 책임연구원. 관심분야는 Temporal Database, Spatio-Temporal Database, Data Warehouse, OLAP

정 연 돈
1994년 2월 고려대학교 전산과학과 학사 1996년 2월 한국과학기술원 전산학과 석사. 2000년 8월 한국과학기술원 전자전산학과 전산학전공 박사. 2000년 9월~2001년 8월 한국과학기술원 정보전자연구소 Post-Doc. 연구원. 2001년 9월~2003년 2월 한국과학기술원 전자전산학과 전산학전공 연구교수. 2003년 3월~2006년 2월 동국대학교 컴퓨터공학과 교수. 2006년 3월~현재 고려대학교 컴퓨터·통신공학부 교수 관심분야는 XML, Mobile/Broadcast Databases, Sensor Networks, Spatial Databases, Database Systems 등



김 명 호

1982년 서울대학교 컴퓨터 공학과(학사) 1984년 서울대학교 컴퓨터 공학과(석사) 1989년 Michigan State Univ. 전산학(박사). 1989년~현재 한국과학기술원 전산학전공 교수. 1995년 Univ. of Virginia 방문 교수. 관심분야는 Database, Distributed Systems, Workflow, Multimedia, OLAP, Data Warehouse