시간지원 데이터의 효율적인 관리를 위한 이동 방법

윤 홍 원†

요 약

본 논문에서는 시간지원 데이터를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리한 저장 구조를 기반으로 하는 네 가지 데이터 이동 방법을 제안하였다. 제안한 데이터 이동 방법은 시간단위에 의한 이동 방법, LST-GET(Least valid Start Time-Greatest valid End Time)에 의한 이동 방법, AST-AET(Average valid Start Time-Average valid End Time)에 의한 이동 방법, 그리고 Min-Overlap에 의한 이동 방법이 있다. 각각의 이동 방법에서는 세그먼트의 경계값, 각 세그먼트에 저장되는 개체 버전 등을 정의하였다. 제안한 이동 방법에 대해서 사용자 질의에 대한 평균 응답 시간을 측정하였다. 실험결과, LLT(Long Lived Tuples)가 없는 경우에는 LST-GET에 의한 이동 방법, 그리고 AST-AET에 의한 이동 방법이 시간단위에 의한 이동 방법보다 성능이 우수하였다. LLT가 있는 경우에는 LST-GET에 의한 이동 방법의 성능이 저하되었다. AST-AET에 의한 이동 방법은 시간단위에 의한 이동 방법과 LST-GET에 의한 이동 방법보다 질의에 대한 성능이 우수하였다. Min-Overlap에 의한 이동 방법은 질의에 대한 평균 응답 시간에서 AST-AET에 의한 이동 방법과 비슷한 결과를 보였고, 공간 이용율 측면에서는 AST-AET에 의한 이동 방법보다 효율적이었다.

Migration Method for Efficient Management of Temporal Data

Hongwon Yun†

ABSTRACT

In this paper we proposed four data migration methods based on time segmented storage structure including past segment, current segment, and future segment. The migration methods proposed in this paper are the Time Granularity migration method, the LST-GET (Least valid Start Time-Greatest valid End Time) migration method, the AST-AET (Average valid Start Time-Average valid End Time) migration method, and the Min-Overlap migration method. In the each data migration method we define the dividing criterion among segments and entity versions to store on each segment. We measured the response time of queries for the proposed migration methods. When there are no LLTs (Long Lived Tuples), the average response time of AST-AET migration method and LST-GET migration method are smaller than that of Time Granularity migration method. In case of existing LLT, the performance of the LST-GET migration method decreased. The AST-AET migration method resulted in better performance for queries than the Time Granularity migration method and the LST-GET migration method. The Min-Overlap migration method resulted in the almost equal performance for queries compared with the AST-AET migration method, in case of storage utilization more efficient than the AST-AET.

키워드 : 시간지원 데이터베이스(temporal database), 데이터 이동(data migration)

1. 서 론

시간지원 데이터베이스(Temporal Database)는 시간에 따라 변하는 개체의 정보를 기록하고 개체의 현재 상태, 과거 상태는 물론, 계획되어 있는 미래 상태에 대해서 질의할 수 있는 특징을 가지고 있다. 시간지원 데이터베이스의 응용 분야는, 주식 시세 관리, 회계 관리, 은행의 입출금 관리 등과 같은 재무 관리 분야가 있고, 개인 신상의 기록, 환자의 진료 기록, 재고 관리 등과 같은 기록 관리 분야가 있다. 또한, 계획 관리 분야로써 항공 예약, 기차 예약, 호텔 예약, 프로젝트 관리 등이 있고, 과학적 응용 분야로써 실험에서 발생하는 온도 변화의 추적 등이 있다[1, 2].

시간지원 데이터베이스와 관련된 연구에서는 시간지원 데이터 모델과 시간지원 데이터의 액세스 방법에 관한 연구가 많았으며[11-15] 최근에는 시간지원 데이터의 저장 기술에 관한 연구가 차츰 늘어나고 있다[3-5]. 과거에 유효했던 데이터와 현재에 유효한 데이터로 분리할 수 있는 시간지원 데이터의 특성을 고려하여 시간지원 데이터를 과거 세그먼트와 현재 세그먼트로 분리하는 저장 구조에 관한 연구가 있었다[3, 7]. 이들 연구에서는 과거 세그먼트의 구조를 제안하거나 과거 세그먼트와 현재 세그먼트를 분리하는 기준을 제안하였으나 미래 세그먼트는 고려하지 않았다.

† 정 회 원 : 신라대학교 컴퓨터정보공학부 교수
논문접수 : 2001년 6월 28일, 심사완료 : 2001년 9월 20일

시간지원 데이터의 시간적 특성을 고려하면 유효한 기간에 따라 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리할 수 있다. 시간지원 데이터를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리한 구조(본 논문에서는 분리 저장 구조라고 부른다)는 분리하지 않은 저장 구조보다 시간지원 질의에 대해서 빠른 응답을 보일 뿐만 아니라[9], 오래된 개체 버전을 제3의 저장 매체에 저장할 수 있으므로 저장 비용의 측면에서 효율적이다.

시간지원 데이터의 분리 저장 구조에서는 시간이 경과하면 어느 시점에서는 세그먼트 사이에 데이터의 이동이 일어나야 한다. 세그먼트 사이에 데이터 이동이 필요한 분리 저장 구조에서는 데이터를 이동하는 방법에 따라 사용자 질의에 대한 성능의 차이가 나타나게 된다. 본 논문에서는 분리 저장 구조를 기반으로 하는 네 가지 데이터 이동 방법을 제안하고 실험을 통해서 데이터 이동 방법들의 성능을 분석한다.

본 논문의 구성은 다음과 같다. 2장에서는 시간지원 데이터의 분리 저장 구조와 관련 있는 기존의 연구에 대해서 살펴본다. 3장에서는 분리 저장 구조를 기반으로 하는 데이터 이동 방법 네 가지를 제안한다. 4장에서는 제안한 데이터 이동 방법에 대해서 사용자 질의에 대한 성능을 평가하고 마지막으로, 5장에서 결론을 맺는다.

2. 관련 연구

본 장에서는 시간지원 데이터베이스와 관련된 기존의 분리 저장 방법과 데이터 이동 방법에 대해서 살펴 본다. Ahn과 Snodgrass는 시간지원 데이터에 대한 검색 속도를 향상시키고 공간을 효율적으로 활용하기 위해서 시간지원 데이터의 저장 장소를 현재 저장소와 이력 저장소로 나누고, 이력 저장소의 구조에 대해서 역 체인, 접근 리스트, 클러스터링, 스택, 그리고 셀루러 체인 등 다섯 가지 저장 구조를 제안하였다[10]. 이 저장 구조에서는 시간지원 데이터의 저장 장소를 현재 저장소와 이력 저장소로 분리해서, 현재 저장소에는 현재 유효한 데이터를 저장하고 이력 저장소에는 과거에 유효했던 데이터를 저장한다. 그리고 각각의 데이터를 현재 버전(open version), 이력 버전(closed version)이라고 부른다. Ahn과 Snodgrass가 제안한 저장 구조들은 모두 과거에 유효했던 개체 버전과 현재 유효한 개체 버전만을 고려하고 있으며 미래에 대한 계획이나 예측 등으로 인해서 발생할 수 있는 미래에 유효한 개체 버전은 고려하지 않았다.

Sarda는 이력 데이터베이스 관리 시스템이 수행할 역할로써 이력 릴레이션을 과거, 현재, 그리고 미래 세그먼트로 분리해서 관리하는 방안을 개념적으로 제안하였다[8]. Sarda의 제안에 의하면, 과거 세그먼트에는 유효 끝 시간이 현재 시간보다 작은 개체 버전을 저장하고, 현재 세그먼트에는 유효 시작 시간이 현재 시간보다 작거나 같고 유효 끝 시간이 현재 시간보다 크거나 같은 개체 버전을 저장한다. 또한, 미래 세그먼트에는 유효 시작 시간이 현재 시간보다 큰 개체 버전을 저장한다. Sarda는 시간지원 데이터에 과거 데이터, 현재 데이터뿐만 아니라, 개인이나 조직체의 미래에 대한 계획을 나타내는 미래 데이터를 포함시키고 있다. Sarda는 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장될 개체 버전을 분리하기 위해서 현재 시간을 사용하는 개념적인 분리 방안을 제안하였으나 구체적인 방법은 언급하지 않았다.

Sarda는 이력 릴레이션을 과거, 현재, 미래 세그먼트로 분리해서 관리해야 한다고 제안하면서 시간지원 데이터의 이동에 대해서 언급하고 있는데 이 연구에 의하면, 미래 세그먼트에서 현재 세그먼트로의 데이터 이동은 현재 시간이 미래 세그먼트에 들어있는 개체 버전의 유효 시작 시간과 같아지면 일어난다. 마찬가지로 현재 세그먼트에 들어있는 개체 버전 중에서 유효 끝 시간이 현재 시간과 같아지면 현재 세그먼트에서 과거 세그먼트로 데이터 이동이 발생한다.

Sarda가 언급한 데이터 이동 방식대로 한다면, 각 개체 버전이 가지고 있는 유효 시간 단위마다 각 세그먼트에 들어있는 개체 버전의 유효성을 검사하고 그 세그먼트에서 더 이상 유효하지 않은 개체 버전은 다른 세그먼트로 이동해야 한다. Sarda는 시간단위마다 세그먼트에 들어있는 개체 버전의 유효성을 검사하고 이동하는 부하가 어느 정도인지는 구체적으로 다루지 않고 개념적 수준에서 데이터 이동에 대해서 언급하였다.

Kouramajian은 시간지원 데이터의 저장 장소를 광 디스크와 자기 디스크로 구분해서 오래된 과거에 유효했던 개체 버전은 광 디스크에 저장하고 가까운 과거에 유효했던 개체 버전과 현재 유효한 개체 버전은 자기 디스크에 저장한다. Kouramajian은 광 디스크와 자기 디스크에 저장될 개체 버전을 구분하기 위해서 현재 유효한 개체 버전 중에서 가장 작은 유효 시작 시간을 세그먼트를 분리하는 경계값으로 사용하였다[3]. 경계값보다 작은 유효 끝 시간을 가지는 개체 버전은 대부분 광 디스크에 저장하고 참조한다. 유효 시작 시간이 경계값보다 작고 유효 끝 시간이 경계값보다 큰 개체 버전은 광 디스크와 자기 디스크 양쪽 모두에 저장하고 참조한다. 또한, 유효 시작 시간이 경계값보다 큰 개체 버전은 자기 디스크에 저장하고 참조한다. Kouramajian은 시간지원 데이터를 분리하면서 과거에 유효했던 개체 버전과 현재 유효한 개체 버전만을 고려하였으며 미래에 유효한 개체 버전은 다루지 않았다.

Kouramajian은 시간지원 데이터의 저장 장소를 광 디스크와 자기 디스크로 구분해서 먼 과거에 유효했던 개체 버전은 광 디스크에 저장하고 가까운 과거에 유효했던 개체 버전과 현재 유효한 개체 버전은 자기 디스크에 저장하는 방법을 제안하면서, 광 디스크와 자기 디스크에 저장될 개

체 버전을 구분하는 경계값을 제시하였다. 현재 유효한 개체 버전 중에서 가장 작은 유효 시작 시간을 경계값으로 하고 이 경계값을 T_m이라 한다.

T_m보다 작은 유효 끝 시간을 가지는 개체 버전은 대부분 광 디스크에 저장하고 참조한다. T_m보다 유효 시작 시간은 작고, 유효 끝 시간은 T_m보다 큰 개체 버전은 광 디스크와 자기 디스크 양쪽 모두에 저장하고 참조한다. 유효 시작 시간이 T_m보다 큰 개체 버전은 자기 디스크에 저장하고 참조한다. 광 디스크에만 저장되는 과거 개체 버전은 이력 버전이라고 하고, 광 디스크와 자기 디스크 양쪽에 모두 저장되는 개체 버전은 걸친 버전(spanned version)이라고 하였다. 또한, 자기 디스크에만 저장하는 현재 버전을 걸치지 않은 버전(non-spanned version)이라고 구분하였다.

Kouramajian의 분리 방법에서는 현재 시간에 유효한 개체 버전 중에서 가장 작은 유효 시작 시간을 경계값으로 하기 때문에 현재 세그먼트가 지나치게 커지게 된다. 일반적으로 질의의 상당한 부분은 현재 세그먼트를 검색하는 현재 질의라고 보면 이러한 방법은 검색 속도를 향상시키기 어렵다. 또한, Kouramajian은 미래 데이터에 대해서는 다루지 않았다.

본 논문에서는 Sarda가 개념적으로 언급한 바 있는 시간지원 데이터의 분리 방법을 일반화한 시간단위 이동 방법을 제안하고, Kouramajian이 제안한 분리 방법의 개념을 확장해서 미래 데이터까지 처리하는 데이터 이동 방법을 구체화하여 제안한다. 또한, 시간지원 데이터의 평균 경계값 특성을 이용하는 데이터 이동 방법을 제안하고, 과학적 응용에서 발생하는 시간지원 데이터에 적합한 데이터 이동 방법을 제안한다. 제안하는 네 가지 이동 방법에 대해서 사용자 질의에 대한 성능을 평가한다.

3. 시간지원 데이터의 이동 방법

3장에서는 시간지원 데이터의 이동 방법 네 가지를 살펴본다. 이동 방법별로 경계값을 정의하고 각 세그먼트에 저장되는 개체 버전을 정의한다.

3.1 시간단위에 의한 이동 방법

시간지원 데이터의 유효 시간과 현재 시간의 관계에 따라 과거에 유효했던 개체 버전, 현재 유효한 개체 버전, 미래에 유효한 개체 버전으로 나눌 수 있다. 현재 시간을 기준으로 어떤 개체 버전의 유효 끝 시간이 현재 시간보다 작으면 과거 세그먼트에 저장하고, 유효 시작 시간이 현재 시간보다 작거나 같고 유효 끝 시간이 현재 시간보다 크면 현재 세그먼트에 저장한다. 그리고 유효 시작 시간이 현재 시간보다 큰 개체 버전은 미래 세그먼트에 저장한다. 과거 세그먼트에 저장되는 개체 버전의 집합을 P, 현재 세그먼트에 저장되는 개체 버전을 C, 그리고 미래 세그먼트에 저장되는 개체 버전을 F라고 하면, 다음과 같이 정의할 수 있다.

- $P = \{ E_{ij} \mid E_{ij}.V_e \le now \}$
- $C = \{ E_{ij} \mid E_{ij}.V_s \le now < E_{ij}.V_e \}$
- $F = \{ E_{ij} \mid E_{ij}.V_s > now \}$

위 식에서 now는 현재 시간, E_{ij}는 개체 E_i의 j번째 버전, $E_{ij}.A$는 개체 버전 E_{ij}의 모든 애트리뷰트, $E_{ij}.V_s$는 개체 버전 E_{ij}의 유효 시작 시간, $E_{ij}.V_e$는 개체 버전 E_{ij}의 유효 끝 시간을 나타낸다. 본 논문의 이후에도 이 기호들은 같은 의미로 사용한다.

과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 들어있는 개체 버전의 집합에서 각 세그먼트의 개체 버전이 가지고 있는 유효 시간 간격의 집합을 각각 I_p, I_c, 그리고 I_f 라고 하고 다음과 같이 정의한다.

- $I_p = \{ i_{p1}, i_{p2}, \cdots, i_{pl} \}$
- $I_c = \{ i_{c1}, i_{c2}, \cdots, i_{cm} \}$
- $I_f = \{ i_{f1}, i_{f2}, \cdots, i_{fn} \}$

I_p의 최소값을 $MinStart_{ip}$, 최대값을 $MaxEnd_{ip}$라고 하고, I_c의 최소값을 $MinStart_{ic}$, 최대값을 $MaxEnd_{ic}$라고 한다. 그리고 I_f의 최소값을 $MinStart_{if}$, 최대값을 $MaxEnd_{if}$라고 한다. I_p, I_c, 그리고 I_f의 lifespan을 각각 $L(I_p)$, $L(I_c)$, 그리고 $L(I_f)$라고 하고 다음과 같이 정의한다.

- $L(I_p) = [MinStart_{ip}, MaxEnd_{ip})$
- $L(I_c) = [MinStart_{ic}, MaxEnd_{ic})$
- $L(I_f) = [MinStart_{if}, MaxEnd_{if})$

데이터 이동 프로세서가 현재 세그먼트를 검사하는 범위는 $MinStart_{ic}$에서 $MaxEnd_{ic}$까지이고, 미래 세그먼트를 검사하는 범위는 $MinStart_{if}$에서 $MaxEnd_{if}$까지가 된다. 데이터 이동 프로세서는 현재 세그먼트와 미래 세그먼트를 차례로 검사하게 되므로 그 범위가 $MinStart_{ic}$에서 $MaxEnd_{if}$까지가 된다. 이것을 *lifespan*으로 나타내면 다음과 같다.

- $L(L(I_c) \cup L(I_f)) = [MinStart_{ic}, MaxEnd_{if}]$

과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트의 시간 범위는 각각 $L(I_p)$, $L(I_c)$, 그리고 $L(I_f)$ 이고, 이것은 시점 질의와 범위 질의가 들어왔을 때 검색 대상이 되는 세그먼트를 정하는 기준이 된다. 또한, $L(I_c)$와 $L(I_f)$는 데이터의 유효성 검사의 대상이 되는 현재 세그먼트와 미래 세그먼트의 시간 범위가 된다.

시간지원 데이터의 시간단위마다 미래 세그먼트와 현재 세그먼트에 저장되어 있는 개체 버전을 검사해서 그 세그먼트에서 더 이상 유효하지 않은 데이터는 다른 세그먼트로 옮겨야 하는데, 이때 기준이 되는 시간단위는 크기에 따라서 일반적으로 초, 분, 시간, 일, 주, 월, 년 중에 하나가 될 수 있다. 이러한 데이터 이동의 기준이 되는 시간단위는 시간지원 데이터의 시간단위가 되는데, 시간단위에 의한 이동 방법에서는 매 시간단위가 세그먼트를 검사하는 시기가 된다.

3.2 LST-GET(Least valid Start Time-Greatest valid End Time)에 의한 이동 방법

현재 시각에 유효한 개체 버전의 유효시간 집합에서 유효 시작 시각이 가장 작은 개체 버전의 유효 시작 시각을 *LST*, 유효 끝 시각이 가장 큰 개체 버전의 유효 끝 시각을 *GET*라고 하고 다음과 같이 정의한다.

- $LST = \min \{ E_{ij}.V_s \mid E_{ij}.V_s \le now < E_{ij}.V_e \}$
- $GET = \max \{ E_{ij}.V_e \mid E_{ij}.V_s \le now < E_{ij}.V_e \}$

*LST*와 *GET*를 기준으로 시간지원 데이터를 과거 데이터, 현재 데이터, 그리고 미래 데이터로 나누어서 저장한다. 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장되는 개체 버전의 집합을 각각 *P*, *C*, *F*, 그리고 각 세그먼트에 중복해서 저장되는 개체 버전의 집합을 *PC*, *CF*라 하고 다음과 같이 정의한다.

- $P = \{ E_{ij} \mid E_{ij}.V_e < LST \}$
- $C = \{ E_{ij} \mid (E_{ij}.V_s \ge LST) \wedge (E_{ij}.V_e < GET) \}$
- $F = \{ E_{ij} \mid E_{ij}.V_s \ge GET \}$
- $PC = \{ E_{ij} \mid E_{ij}.V_s < LST \wedge LST < E_{ij}.V_e < GET \}$
- $CF = \{ E_{ij} \mid (LST \le E_{ij}.V_s < GET) \wedge (E_{ij}.V_e > GET) \}$

데이터 이동은 옮김과 복사로 나누어진다. *LST* 또는 *GET*에 걸치는 개체 버전은 경계값을 중심으로 양쪽 세그먼트에 중복해서 저장해 두는 복사를 하게 된다. 각 세그먼트 사이에 데이터를 옮기고 복사하는 경우는 다음과 같이 분류할 수 있다.

- 미래 세그먼트에서 현재 세그먼트로 옮김
- 현재 세그먼트에서 과거 세그먼트로 옮김
- 미래 세그먼트에서 현재 세그먼트로 복사
- 현재 세그먼트에서 과거 세그먼트로 복사

개체 버전을 복사한 뒤에 지우기 과정이 필요한 옮김은 미래 세그먼트에 현재 세그먼트로 옮김과 현재 세그먼트에서 과거 세그먼트로 옮김이 있다. 옮김과 복사의 대상이 되는 개체 버전은 (그림 1)과 (그림 2)와 같다.

데이터의 이동 방향	해당 개체 버전
미래 세그먼트 → 현재 세그먼트	$(E_{ij}.V_s \ge LST) \wedge (E_{ij}.V_e < GET)$
현재 세그먼트 → 과거 세그먼트	$E_{ij}.V_e < LST$

(그림 1) 옮김의 대상이 되는 개체 버전

데이터의 이동 방향	해당 개체 버전
미래 세그먼트 → 현재 세그먼트	$(LST \le E_{ij}.V_s < GET) \wedge (E_{ij}.V_e > GET)$
현재 세그먼트 → 과거 세그먼트	$(E_{ij}.V_s < LST) \wedge (E_{ij}.Ve < GET)$

(그림 2) 복사의 대상이 되는 개체 버전

3.3 AST-AET(Average valid Start Time-Average valid End Time)에 의한 이동 방법

3.2 절에서 살펴본 LST-GET에 의한 이동 방법에서 경계값인 *LST*와 *GET*를 기준으로 해서 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 구분하는 경우에 일어날 수 있는 두 가지 문제점은 다음과 같다.

첫째, LLT가 있으면 확장된 현재 영역의 범위가 커져서 현재 세그먼트에 들어가는 데이터의 수가 지나치게 많아질 수 있다. 이것은 현재 시점에 유효한 데이터에 대한 질의가 비교적 많다는 질의의 특성에 비추어 보면, 질의에 대한 응답 시간이 늘어날 수 있다.

둘째, LLT가 있으면 현재 세그먼트에 들어있는 데이터가 많아지므로 데이터를 이동하는 시점에 다른 세그먼트로 이동해야 하는 이동 대상 데이터 수가 지나치게 많아져서 데이터 이동에 따른 부담을 분산시키지 않고 집중시키는 문제점을 가지고 있다.

*LST*와 *GET*를 이용한 이동 방법을 사용함으로써 현재 세그먼트의 크기가 지나치게 커지는 것을 막기 위해서 유효 시작 시각의 평균은 과거 세그먼트와 현재 세그먼트를 구분하는 경계값으로 사용하고, 유효 끝 시각의 평균은 현재 세그먼트와 미래 세그먼트를 구분하는 경계값으로 쓸 수 있다. 이 경계값을 사용하는 경우에, 아주 긴 수명을 가지는 개체 버전의 수가 적으면 아주 긴 수명의 개체 버전을 제외하고 일반적인 개체 버전의 시간 간격을 반영한 경계값을 구하게 되는 효과를 가져오고, 아주 긴 수명을 가지는 데이터가 대다수이면 아주 긴 수명의 데이터 특성대로 경계값을 구할 수 있으므로 현재 세그먼트의 크기가 지나치게 커지는 것을 막을 수 있다.

과거 세그먼트와 현재 세그먼트를 나누는 경계값으로 사용하는 유효 시작 시각의 평균값과 유효 끝 시각의 평균값을 이용하는 데이터 이동 방법을 제안한다. 현재 시각에 유효한 개체 버전의 집합을 *C*, *C*에 들어가는 각 개체 버전이 가지고 있는 유효 시간 간격의 집합을 *I*라고 하면, *I*와 *C*는 각각 다음과 같다.

- $C = \{ E_{ij} \mid E_{ij}.V_s \le now < E_{ij}.V_e \}$
- $I = \{ c_1, c_2, \cdots, c_n \}$

개체 버전의 시작 유효 시간이 현재 시간 보다 작고 유효 종료시간이 현재 시간보다 큰 개체 버전을 *Lc*라 하고, 유효 시작시간이 현재 시간보다 작거나 같고 유효 종료시간이 현재 시간보다 큰 개체 버전의 집합을 *Rc*라 하면, 각각 다음과 같이 정의한다.

- $Lc = \{ E_{ij} \in C \mid (E_{ij}.V_s < now) \wedge (E_{ij}.V_e > now) \}$
- $Rc = \{ E_{ij} \in C \mid (E_{ij}.V_s \le now) \wedge (E_{ij}.V_e > now) \}$

*Lc*에 들어가는 모든 개체 버전의 유효 시간 간격의 시

간 범위를 $T(Lc)$ 라고 하고, 집합 Rc에 들어가는 모든 개체 버전의 유효 시간 간격의 시간 범위를 $T(Rc)$ 라고 하고, 이것을 각각 다음과 같이 정의한다.

- $T(Lc) = \bigcup_{[vs, ve] \in Lc} [V_s, V_e]$
- $T(Rc) = \bigcup_{[vs, ve] \in Rc} [V_s, V_e]$

AST를 구하는데 대상이 되는 개체 버전이 가지고 있는 유효 시작시간의 집합을 $S(I)$라고 하고, AET를 구하는데 대상이 되는 개체 버전이 가지고 있는 유효 종료시간의 집합을 $E(I)$라고 하고, $S(I)$와 $E(I)$를 다음과 같이 정의한다.

- $S(I) = \{V_s \mid \exists V_s \in T(Lc)\ such\ that\ [V_s, V_e] \in Lc\}$
- $E(I) = \{V_e \mid \exists V_e \in T(Rc)\ such\ that\ [V_s, V_e] \in Lc\}$

$S(I)$에 들어가는 모든 유효 시작시간의 합을 $SUM(S)$라 하고 다음과 같이 정의한다.

- $SUM(S) = \bigcup_{Vs \in Lc} S(I)$

모든 유효 시작시간의 평균을 나타내는 AST는 다음과 같다.

- $AST = \frac{SUM(S)}{n}$

$E(I)$ 에 들어가는 모든 유효 종료시간의 합을 $SUM(E)$라 하고 다음과 같이 정의한다.

- $SUM(E) = \bigcup_{Ve \in Rc} E(I)$

모든 유효 종료시간의 평균을 나타내는 AET는 다음과 같다.

- $AET = \frac{SUM(E)}{n}$

경계값 AST는 과거 세그먼트와 현재 세그먼트에 들어갈 개체 버전을 구분하는 경계값이 되고, AET는 현재 세그먼트와 미래 세그먼트를 구분하는 경계값으로 사용된다. 각 개체 버전이 가지고 있는 유효 시작시간과 유효 종료시간을 경계값과 비교해서 해당하는 세그먼트에 저장하게 된다.

과거 세그먼트, 현재 세그먼트, 리고 미래 세그먼트에 그 저장되는 개체 버전의 집합의 집합을 각각 P, C, F, 그리고 두 세그먼트에 중복해서 저장되는 개체 버전을 PC, CF라 하고 다음과 같이 정의한다.

- $P = \{E_{ij} \mid E_{ij}.V_e < AST\}$
- $C = \{E_{ij} \mid (E_{ij}.V_s \geq AST) \wedge (E_{ij}.V_e < AET)\}$
- $F = \{E_{ij} \mid E_{ij}.V_s \geq AET\}$
- $PC = \{E_{ij} \mid (E_{ij}.V_s < AST) \wedge (AST < E_{ij}.V_e\ AET)\}$
- $CF = \{E_{ij} \mid (AST \leq E_{ij}.V_s < AET) \wedge (E_{ij}.V_e > AET)\}$

AST-AET에 의한 데이터 이동 방법에서 데이터 이동은 옮김과 복사로 구분할 수 있다. AST-AET에 의한 이동에서 각 세그먼트 사이에 개체 버전을 옮기거나 복사하는 경우는 (그림 3)과 같다. 옮김은 이동 대상이 되는 개체 버전을 다른 세그먼트에 쓰고 원래의 세그먼트에서 지우는 과정을 포함한다. 복사는 서로 다른 두 세그먼트에 개체 버전이 동시에 존재하는 것이므로 이동 대상이 되는 개체 버전을 다른 세그먼트에 쓰면 된다. 옮김과 복사는 미래 세그먼트에서 현재 세그먼트로 발생하거나, 현재 세그먼트에서 과거 세그먼트로 발생한다.

데이터의 옮김과 복사	해당 개체 버전
미래 세그먼트 → 현재 세그먼트로 옮김	$(E_{ij}.V_s \geq AST) \wedge (E_{ij}.V_e < AET)$
현재 세그먼트 → 과거 세그먼트로 옮김	$(E_{ij}.V_e < AST)$
미래 세그먼트 → 현재 세그먼트로 복사	$(AST \leq E_{ij}.V_s < AET) \wedge (E_{ij}.V_e > AET)$
현재 세그먼트 → 과거 세그먼트로 복사	$(E_{ij}.V_s < AST) \wedge (AST < E_{ij}.V_e < AET)$

(그림 3) 개체 버전의 옮김과 복사

3.4 Min-Overlap에 의한 이동 방법

시간지원 데이터의 보기가 되는 응용은 비즈니스 응용(business application)과 과학적 응용 (scientific application)으로 분류할 수 있다. 시간단위에 의한 이동 방법, LST-GET에 의한 이동 방법, 그리고 AST-AET에 의한 이동 방법 등이 대상으로 하는 시간지원 데이터의 시간적 특성은 일반적인 비즈니스 응용을 근간으로 하고 있다. 과학적 응용에서 보기가 되는 전형적인 시간적 특성은 물리 실험을 들 수 있다. 이러한 실험에 대한 반응으로 나타나는 시간지원 데이터는 일정한 시간 동안 군집을 이루고 이산적인 특성을 가지고 있다. 본 절에서는 과학적 응용에 적합한 데이터 이동 방법을 제안한다.

과거 세그먼트와 현재 세그먼트에 저장되는 개체 버전을 구분하는 기준이 되는 시점을 LB(*Left Bound with min overlap*), 현재 세그먼트와 미래 세그먼트에 저장되는 개체 버전을 구분하는 기준이 되는 시점을 RB(*Right Bound with min overlap*)라 하고, 경계값으로 사용되는 LB와 RB를 정의하도록 한다. 경계값 LB와 RB를 기준으로 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장되는 개체 버전의 집합의 집합을 각각 P, C, F, 그리고 두 세그먼트에 중복해서 저장되는 개체버전을 각각 PC, CF라 하고 각각의 집합을 정의하면 다음과 같다.

- $P = \{E_{ij} \mid E_{ij}.V_e < LB\}$
- $C = \{E_{ij} \mid (E_{ij}.V_s \geq LB) \wedge (E_{ij}.V_e < RB)\}$
- $F = \{E_{ij} \mid E_{ij}.V_s \geq RB\}$
- $PC = \{E_{ij} \mid (E_{ij}.V_s < LB) \wedge (LB < E_{ij}.V_e < RB)\}$
- $CF = \{E_{ij} \mid (LB \leq E_{ij}.V_s < RB) \wedge (E_{ij}.V_e > RB)\}$

과거 세그먼트가 포함하는 모든 개체 버전을 EP, 현재 세그먼트가 포함하는 모든 개체 버전을 EC, 미래 세그먼트가 포함하는 모든 개체 버전을 EF라 하면 각 세그먼트에 저장되는 개체 버전은 다음과 같이 정의할 수 있다.

- $EP = P \bigcup_{All} PC$
- $EC = PC \bigcup_{All} C \bigcup_{All} CF$
- $EF = CF \bigcup_{All} F$

EP, EC, 그리고 EF에 들어있는 개체 버전의 유효 시간 간격의 집합을 각각 I_P, I_C, 그리고 I_f라고 하고 다음과 같이 나타낸다.

- $I_P = \{ip_1, ip_2, \cdots, ip_l\}$
- $I_c = \{ic_1, ic_2, \cdots, ic_m\}$
- $I_f = \{if_1, if_2, \cdots, if_n\}$

세그먼트를 구분하는 기준이 되는 시점을 구하기 위해서 쓰게 될 함수를 정의하면 다음과 같다.

- $O_{Ic}(t) = |\{i_c \in I_c \mid i_c.V_s < t < i_c.V_e\}|$
- $O_{If}(t) = |\{i_f \in I_f \mid i_f.V_s < t < i_f.V_e\}|$
- $S_{Ic}(t) = |\{i_c \in I_c \mid i_c.V_s = t\}|$
- $E_{Ic}(t) = |\{i_c \in I_c \mid i_c.V_e = t\}|$
- $S_{If}(t) = |\{i_f \in I_f \mid i_f.V_s = t\}|$
- $E_{If}(t) = |\{i_f \in I_f \mid i_f.V_e = t\}|$

과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트 각각에 들어가는 시간 간격의 집합에서 임의의 시점 t를 포함하는 시간 범위를 각각 $R(I_p)$, $R(I_c)$, 그리고 $R(I_f)$ 라고 하고, 이것을 정의하면 다음과 같다.

- $R(I_p) \bigcup_{[v_s, v_e] \in I_p} [V_s, V_e]$
- $R(I_c) \bigcup_{[v_s, v_e] \subset I_c} [V_s, V_e]$
- $R(I_f) \bigcup_{[v_s, v_e] \in I_f} [V_s, V_e]$

$R(I_p)$, $R(I_c)$, 그리고 $R(I_f)$에 들어가는 유효 시작 시각의 집합을 $S(I_p)$, $S(I_c)$, 그리고 $S(I_f)$로 나타내고, 유효 끝 시각의 집합을 $E(I_p)$, $E(I_c)$, 그리고 $E(I_f)$로 나타내고 이것을 정의하면 다음과 같다.

- $S(I_p) = \{I_p.V_s \mid \exists V_s \in R(I_p) such\ that\ [V_s, V_e] \in I_p\}$
- $E(I_p) = \{I_p.V_e \mid \exists V_e \in R(I_p) such\ that\ [V_s, V_e] \in I_p\}$
- $S(I_c) = \{I_c.V_s \mid \exists V_s \in R(I_c) such\ that\ [V_s, V_e] \in I_c\}$
- $E(I_c) = \{I_c.V_e \mid \exists V_e \in R(I_c) such\ that\ [V_s, V_e] \in I_c\}$
- $S(I_f) = \{I_f.V_s \mid \exists V_s \in R(I_f) such\ that\ [V_s, V_e] \in I_f\}$
- $E(I_f) = \{I_f.V_e \mid \exists V_e \in R(I_f) such\ that\ [V_s, V_e] \in I_f\}$

I_c의 임의의 시점에서 왼쪽으로 가장 가까운 유효 시작 시각을 $nsleft(I_c.t)$라고 하고, 이것은 다음과 같이 정의한다.

- $nsleft(I_c, t) = \max\{I_c.V_s \mid I_c.V_s \in S(I_c)\}$

I_c의 임의의 시점부터 왼쪽으로 가장 가까운 유효 시작 시각에서 겹치는 유효 시간 간격의 개수는 $O_{Ic}(nsleft(I_c.t))$로 나타낼 수 있다.

새로운 LB를 구하는데 해당하는 시간 영역은, 현재 사용하고 있는 $LB+1$에서 새 LB를 구하려는 시점 t까지가 된다. 새로운 LB를 구하는데 해당하는 시간 영역을 $S(Ic)$라고 하고 다음과 같이 정의한다.

- $S(Ic) = [\ LB+1,\ t\ ]$

새로운 LB를 구하는 시점을 t라고 하고, t에서 왼쪽으로 가장 가까운 유효 시작 시각은 $nsleft(Ic.t)$이다. $S(I_c)$에서 나오는 모든 유효 시작 시각은 $nsleft(I_c.t)$, $nsleft(nsleft(I_c.t))$, …가 된다. $nsleft(I_c.t)$를 통해서 나온 바로 앞의 유효 시작 시각을 차례대로 st_1, st_2, …라고 하면, 값의 범위는 $LB+1 < st_1, st_2, \cdots < t$이고, 이 유효 시작 시각 각각의 시점에서 겹치는 시간 간격의 개수는 $O_{Ic}(st_1)$, $O_{Ic}(st_2)$, $O_{Ic}(st_3)$, …로 나타낼 수 있다.

이 중에서 가장 최소값이 되는 유효 시작 시각이 과거 세그먼트와 현재 세그먼트를 구분하는 경계값 LB가 되고, 이를 정의하면 다음과 같다.

- $LB = \min\{st_i \mid O_{Ic}(st_1), O_{Ic}(st_2), \cdots, O_{Ic}(st_n)\}$

I_f의 임의의 시점에서 오른쪽으로 가장 가까운 유효 끝 시각을 $neright(I_f.t)$라고 하고, 다음과 같이 정의한다.

- $neright(I_f.t) = \min\{I_f.V_e \mid I_f.V_e \in E(I_f)\}$

I_f의 임의의 시점부터 오른쪽으로 가장 가까운 유효 끝 시각에서 겹치는 유효 시간 간격의 개수는 $O_{If}(neright(I_f.t))$로 나타낼 수 있다. 새로운 RB를 구하는 시점을 t라고 하면 새로운 RB를 구하는데 해당되는 시간 영역은, 시점 t에서 LB까지의 시간 길이를 t를 중심으로 접었을 때 LB가 만나는 시점을, RB를 구하기 위한 구간의 끝 시점으로 한다. 시점 t에서 LB까지의 길이 l은 $t-LB$가 된다. 새로운 RB를 구하는데 해당되는 영역은 시점 $t+1$에서 $t+l$인데, 이를 $S(I_f)$라고 하고 다음과 같이 정의한다.

- $S(I_f) = [\ t+1,\ t+l\ ]$

새로운 RB를 구하는 시점을 t라고 하면, $t+1$에서 오른쪽으로 가장 가까운 유효 끝 시각은 $neright(I_f.t)$이다. $S(I_f)$에서 나오는 모든 유효 끝 시각은 $neright(I_f.t)$, $neright(neright(I_f.t))$, …가 된다. $neright(I_f.t)$를 통해서 나온 바로 앞의 유효 끝 시각을 차례대로 et_1, et_2, …라고 하면, 값의 범위는 $t+1 < et_1, et_2, \cdots < t+l$이고, 이 유효 끝 시각 각각

의 시점에서 겹치는 시간 간격의 개수는 $O_{lf}(et_1)$, $O_{lf}(et_2)$, $O_{lf}(et_3)$, …로 나타낼 수 있다.

이 중에서 가장 최소값이 되는 유효 끝 시각이 현재 세그먼트와 미래 세그먼트를 구분하는 경계값 RB가 되고, 이를 정의하면 다음과 같다.

- $RB = \min\{et_i \mid O_{lf}(et_1), O_{lf}(et_2), \cdots, O_{lf}(et_n)\}$

작업	세그먼트	해당하는 개체 버전
옮김	미래 세그먼트 → 현재 세그먼트	$(E_{ij}.V_s \geq LB) \wedge (E_{ij}.V_e < RB)$
	현재 세그먼트 → 과거 세그먼트	$E_{ij}.V_e < LB$
복사	미래 세그먼트 → 현재 세그먼트	$(LB \leq E_{ij}.V_s < RB) \wedge (E_{ij}.V_e > RB)$
	현재 세그먼트 → 과거 세그먼트	$(E_{ij}.V_s < LB) \wedge (E_{ij}.V_e < RB)$

(그림 4) 개체 버전의 옮김과 복사

데이터를 이동하는 과정에서 발생하는 작업은 옮김과 복사로 나눌 수 있다. (그림 4)는 Min-Overlap에 의한 데이터 이동 방법에서 데이터를 이동할때 수행하는 옮김, 복사의 대상이 되는 개체 버전 등을 나타내고 있다.

(그림 5)는 3장에서 제안한 이동 방법들의 데이터 이동 과정을 일반화하여 나타낸 것이다. 데이터 이동 알고리즘에서 BCF는 현재 세그먼트와 미래 세그먼트의 경계값을 나타내고 BPC는 과거 세그먼트와 현재 세그먼트의 경계값을 나타낸다. 시간단위에 의한 이동 방법에서는 BCF는 now와 같고 BPC는 $now+1$과 같다.

```
Algorithm Migration( )
while (not end-of-file ( FutureSegment )) do
    EntityVersion ← read FutureSegment record
    if EntityVersion.Vs BCF then
        add ( Buffer, EntityVersion )
    if EntityVersion.Ve BCF then
        add ( FutBuffer, EntityVersion )
    end if
    else add ( FutBuffer, EntityVersion )
    end if
    if ( Buffer is full ) then
        write ( CurrentSegment, Buffer )
    end if
    if ( FutBuffer is full ) then
        write ( FutureSegment, FutBuffer )
    end if
end while
Buffer ← null
while (not end-of-file ( CurrentSegment )) do
    EntityVersion ← read CurrentSegment record
    if EntityVersion.Vs BPC then
        add ( Buffer, EntityVersion )
    if EntityVersion.Ve BPC then
        add ( CurBuffer, EntityVersion )
    end if
    else add ( CurBuffer, EntityVersion )
    end if
    if ( Buffer is full ) then
        write ( PastSegment, Buffer )
    end if
    if ( CurBuffer is full ) then
        write ( CurrentSegment, CurBuffer )
    end if
end while
```

(그림 5) 데이터 이동 알고리즘

4. 성능 평가

본 장에서는 분리 저장 구조를 기반으로 하는 네 가지 데이터 이동 방법에 대해서 질의에 대한 성능을 평가한다. 먼저, 실험을 위한 환경과 방법을 살펴보고 실험에 대한 결과를 분석하도록 한다.

4.1 실험 방법

분리 저장 구조에서 시간단위에 의한 이동 방법, LST-GET에 의한 이동 방법, AST-AET에 의한 이동 방법, 그리고 Min-Overlap에 의한 이동 방법 사이에 성능을 비교하기 위해서 사용자 질의에 대한 평균 응답 시간을 측정해서 비교한다. 실험에 사용할 데이터는 TimeIT를 이용해서 만들었다. TimeIT는 미국 애리조나 대학의 TimeCenter에서 개발한 시간지원 질의의 연산 알고리즘을 테스트하기 위한 시험 도구이다. 실험을 위한 알고리즘을 C언어로 작성하였고 선(SUN) 사의 울트라 스팍 (ULTRA SPARC)에서 수행하였다.

비즈니스 응용 등과 같은 일반적인 시간지원 데이터의 특성을 근간으로 하는 데이터 이동 방법인 시간단위에 의한 이동 방법, LST-GET에 의한 이동 방법, 그리고 AST-AET에 의한 이동 방법을 대상으로 실험에 사용할 데이터는 다음과 같다.

일반적인 데이터의 유효 시간의 길이는 30에서 50사이에 단위가 1인 크기로 만들었고 실제 상황에서 시간지원 데이터의 유효 시간 길이가 균등 분포와 유사하므로 각각의 길이는 균등 분포로 생성하였다. LLT는 일반적인 튜플의 10배 정도 큰 것으로 가정하고 LLT의 길이는 300에서 500사이의 길이를 가지는 것으로 하고 길이의 분포는 균등 분포로 만들었다. LLT의 비율은 0%, 1%, 3%, 5%, 7%, 그리고 9%로 각각 생성하였으며 LLT는 상대적으로 유효 시간이 긴 데이터를 말하는 것이므로 LLT의 비율이 10%가 넘는 것은 실험에서 고려하지 않는다.

Min-Overlap에 의한 이동 방법은 일정한 시간 동안 군집을 이루는 시간지원 데이터에 적합한 이동 방법이므로, 모의 실험 데이터는 주기적으로 군집을 이루는 핫스팟 분포를 이용해서 생성하였다. 하나의 군집에는 1% 내지 2% 정도 튜

플이 들어가도록 하였고, LLT는 1%와 5%로 각각 만들었고 튜플의 길이는 균등 분포로 생성하였다. 이 실험 데이터는 LLT의 비율을 1%와 5%로 각각 생성하였다. 실험에서는 시게이트(Seagate)사에서 나온 3.5 인치 ST34501N의 하드 디스크 성능 인수값을 사용하였다.

사용자 질의의 종류는 시점 질의와 범위 질의로 구분하였으며 실험에서는 다음과 같이 세가지 경우로 구분해서 실험하였다. 시점 질의는 과거, 현재, 그리고 미래 세그먼트 중에서 한 세그먼트만 검색 대상이 되고, 범위 질의는 한 개, 두 개, 또는 세 개의 세그먼트가 검색 대상이 될 수 있다. 시간지원 질의는 10%, 20%, 30%, 40%, 그리고 50%로 나누고 각 시간지원 질의의 비율 내에서 과거, 현재, 미래, 과거와 현재, 현재와 미래, 과거와 현재와 미래 세그먼트 각각에 균등하게 (1/6씩) 시간지원 질의가 주어지도록 하였다.

임의의 질의를 수행하는 시간은 해당하는 릴레이션을 찾기 위한 한 번의 임의 탐색 시간, 릴레이션의 한 세그먼트가 차지하고 있는 트랙 만큼의 순차 탐색 시간, 회전 지연 시간, 그리고 세그먼트에 들어있는 튜플의 전송시간의 합으로 계산할 수 있다. 실험에서 사용자 질의에 대한 평균 응답 시간은, 질의를 수행하는데 걸리는 시간과 데이터 이동으로 인한 질의당 평균 지연 시간의 합으로 나타내었다.

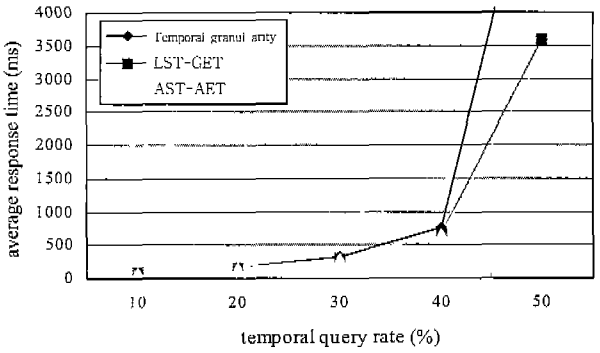
4.2 실험 결과

(그림 6)은 분리 저장 구조에서 시간지원 질의를 10%에서 50%까지 바꾸어 가면서 LLT의 비율이 0%인 경우에 시간단위에 의한 이동, LST-GET에 의한 이동, 그리고 AST-AET에 의한 이동 방법에서 사용자 질의에 대한 평균 응답 시간을 나타낸 것이다. 이 그림의 범례에서 Temporal granularity는 시간 단위에 의한 이동 방법을 뜻하고, LST-GET와 AST-AET는 각각 LST-GET에 의한 데이터 이동 방법과 AST-AET에 의한 이동 방법을 나타낸다.

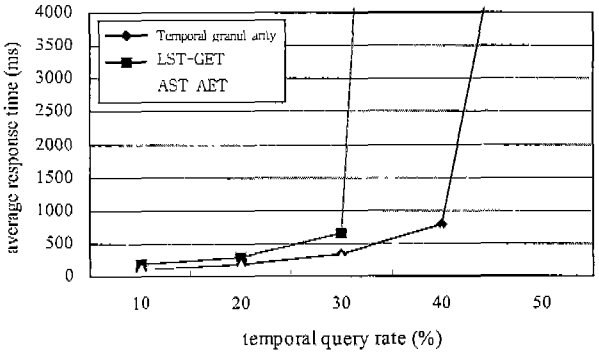
(그림 6)에서 시간지원 질의가 10%인 경우에는 세 가지 데이터 이동 방법의 사용자 질의에 대한 평균 응답 시간이 거의 비슷하게 나타난다. LLT가 없으므로 세 가지 데이터 이동 방법에서 현재 세그먼트의 크기가 비슷하기 때문에 성능의 차이가 나타나지 않는다. 세 가지 이동 방법에서 시간지원 질의의 비율이 증가하면서, 세그먼트의 크기가 큰 과거 세그먼트를 접근하는 회수가 증가하므로 데이터 이동으로 인한 부하가 큰 시간단위에 의한 이동 방법의 성능이 저하된다. 이러한 현상은 시간지원 질의가 50%인 경우에 더욱 현저히 나타나서, 시간단위에 의한 이동 방법이 LST-ET에 의한 이동 방법과 AST-AET에 의한 이동 방법보다 질의에 대한 성능이 약 50%로 감소하였다. AST-AET에 의한 이동 방법과 LST-GET에 의한 이동 방법에서는 LLT가 없으므로 현재 세그먼트의 크기가 비슷하고, 시간단위에 의한 이동 방법보다 데이터 이동 부하가 작기 때문에 거의 비슷한 성능을 보인다.

(그림 7)은 LLT가 있는 경우에 분리 저장 구조에서 시간지원 질의의 비율을 바꾸어 가면서 데이터 이동 방법별로 사용자 질의에 대한 응답 시간을 나타낸 것이다. LLT가 있으면 LST-GET에 의한 이동 방법에서는 LST가 작아지고 GET가 커지므로 현재 세그먼트에 들어갈 개체 버전을 포함하는 범위가 늘어나서 현재 세그먼트의 크기가 급격히 커진다. 반면에, 시간단위에 의한 이동 방법이나 AST- ET에 의한 이동 방법은 현재 세그먼트의 크기 변화가 비교적 작다. 현재 세그먼트를 검색하는 현재 질의가 많은 경우에, LLT가 있으면 현재 세그먼트의 크기가 커지는 LST-GET에 의한 이동 방법이, 현재 세그먼트의 크기 변화가 작은 시간단위에 의한 이동 방법과 AST-AET에 의한 이동 방법보다 질의에 대한 응답 속도가 저하된다.

시간지원 질의가 30% 이상인 경우에는 시간단위에 의한 이동 방법과 AST-AET에 의한 이동 방법이 성능의 격차를 벌리면서 차츰 AST-AET에 의한 이동 방법이 좋은 성능을 보이고 있다. 시간단위에 의한 이동 방법에서는 시간지원 질의의 비율이 증가하면서 범위 질의가 증가하게 되고, 따라서 크기가 큰 과거 세그먼트를 검색하는 비율이 높아지고, 매 초마다 발생하는 데이터를 이동 부하가 크기 때문에 질의에 대한 성능의 저하를 가져왔다. AST-AET에 의한 이동 방법은 데이터를 이동하는 빈도가 시간단위에 의한 이동 방법보다 작으므로 데이터 이동 부하가 작은 AST-AET에 의한 이동 방법이 상대적으로 우수한 성능을 보였다.

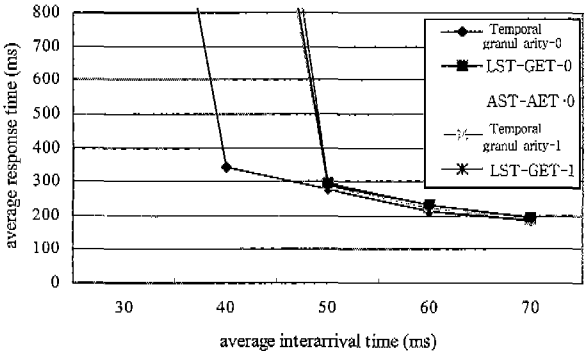


(그림 6) LLT가 없는 경우 질의에 대한 평균 응답 시간



(그림 7) LLT가 있는 경우 질의에 대한 평균 응답 시간

(그림 8)은 사용자 질의가 도착하는 평균 시간 간격의 변화에 대해서 시간단위에 의한 이동 방법, LST-GET에 의한 이동 방법, 그리고 AST-AET에 의한 이동 방법의 평균 응답 시간을 비교한 것이다. 그림의 범례에서 이동 방법 뒤에 붙어있는 0은 LLT가 없는 경우이고, 1은 LLT가 있는 경우를 나타낸다. LLT가 있는 경우에는 질의의 평균 도착 시간 간격이 작아짐에 따라 LST-GET에 이동 방법의 성능 저하가 가장 일찍 나타났다. 이것은 LLT가 있는 경우에는 LST-GET에 의한 이동 방법에서 현재 세그먼트의 크기가 급격히 커지기 때문에 성능의 저하가 가장 일찍 나타났다. LLT가 없는 경우에는 질의의 평균 도착 시간 간격이 작아짐에 따라 시간단위에 의한 이동 방법이 가장 오랫동안 견디다가 가장 나중에 급격한 성능의 저하를 보였다. 이것은 LLT가 없는 시간단위에 의한 이동 방법이 현재 세그먼트의 크기가 가장 작기 때문이다.

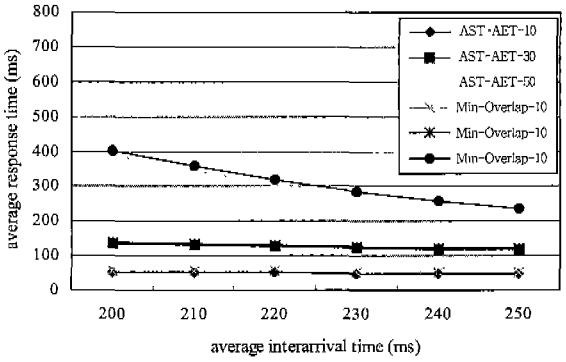


(그림 8) 질의 평균 도착 시간의 변화에 따른 평균 응답 시간

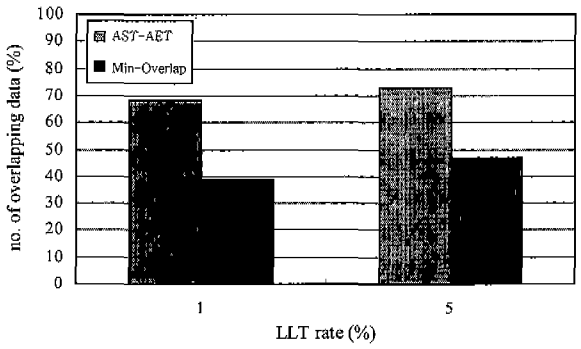
Min-Overlap에 의한 데이터 이동 방법은 과학적 응용에서 발생하는 일정한 시간 동안 군집을 시간지원 데이터를 대상으로 하기 때문에 실험에 사용한 데이터는 핫스팟 분포를 따르게 하였고, LLT를 1%와 5%로 생성한 데이터를 가지고 실험을 하였다. Min-Overlap에 의한 이동 방법의 성능을 비교하기 위한 대상으로 4.2 절에서 살펴 본 것처럼 데이터 이동 방법 중에서 전반적으로 우수한 성능을 보인 AST-AET에 의한 이동 방법을 선택하였다.

(그림 9)는 분리 저장 구조에서 LLT가 1%이고 시간지원 질의가 10%, 30%, 그리고 50%인 경우에 AST-AET에 의한 이동 방법과 Min-Overlap에 의한 이동 방법에 대해서 사용자 질의에 대한 응답 시간을 측정한 것이다. 두 가지 이동 방법은 시간지원 질의의 비율이 같은 경우에 사용자 질의에 대한 응답 시간이 거의 비슷하게 나타나고 있다. 이것은 AST-AET에 의한 이동 방법과 Min-Overlap에 의한 이동 방법은 현재 세그먼트의 크기가 비슷하고 데이터 이동 빈도가 비슷하기 때문이다.

사용자 질의에 대한 성능 측면에서 보면, AST-AET에 의한 이동 방법과 Min-Overlap에 의한 이동 방법에서 비슷한 성능을 보이지만, Min-Overlap에 의한 방법은 두 세그먼트에 중복되는 개체 버전의 수를 작게 하므로 공간 이용율의 측면에서 효율적이다. (그림 10)은 현재 세그먼트에 대해서 중복 저장되는 개체 버전의 수를 백분율로 나타낸 것이다. Min-Overlap에 의한 이동 방법은 군집을 이루는 실세계의 데이터를 한 세그먼트에 저장함으로써 서로 시간적으로 연관성이 있는 개체 버전을 묶을수 있기 때문에 실세계의 특징을 그대로 반영하는 특징을 가지고 있다.



(그림 9) Min-Overlap에 의한 이동 방법과 AST-AET에 의한 이동 방법의 사용자 질의에 대한 평균 응답 시간



(그림 10) Min-Overlap에 의한 이동 방법과 AST-AET에 의한 이동 방법의 중복 저장되는 데이터 수

5. 결 론

본 논문에서는 분리 저장 구조를 기반으로 하는 네 가지 데이터 이동 방법을 제안하였다. 제안한 데이터 이동 방법은 시간단위에 의한 이동 방법, LST-GET에 의한 이동 방법, AST-AET에 의한 이동 방법, 그리고 Min-Overlap에 의한 이동 방법이 있다.

시간단위에 의한 이동 방법에서 각 세그먼트에 저장되는 개체 버전을 정의하고 각 세그먼트에서 데이터의 유효성을 검사하고 이동하는 과정을 보였다. 시간단위에 의한 이동 방법에서 데이터 이동에 따른 부하가 지나치게 커지는 경향을 개선하기 위해서 LST-GET에 의한 이동 방법을 제안하였다. LST-GET에 의한 데이터 이동 방법에서 LST와 GET를 정의하고 경계값과 유효 시간과의 관계를 통해서 각 세그먼트에 저장되는 개체 버전을 정의하였다. LST-

ET에 의한 데이터 이동 방법은 LLT가 있는 경우에 현재 세그먼트가 커지므로 검색 속도가 저하되는데, 이 문제점을 해결하기 위해서 AST-AET에 의한 이동 방법을 제안하였다. AST-AET에 의한 이동 방법에서 AST와 AET를 정의하고 AST와 AET를 경계값으로 각 세그먼트에 저장되는 개체 버전을 정의하였다.

과학적 응용에서 발생하는 군집을 이루는 시간지원 데이터에 적합한 이동 방법으로써 Min-Overlap에 의한 이동 방법을 제안하였다. Min-Overlap에 의한 이동 방법에서 시간지원 데이터를 구분하는 경계값 LB와 RB를 정의하고 각 세그먼트에 저장되는 개체 버전을 정의하였다. LB와 RB를 구하는 방법을 보이고, 세그먼트 사이에 데이터를 이동하는 과정을 보였다.

제안한 이동 방법에 대해서 사용자 질의에 대한 평균 응답 시간을 측정한 결과, LLT가 없는 경우에는 LST-GET에 의한 이동 방법, 그리고 AST-AET에 의한 이동 방법이 데이터 이동 부하가 큰 시간단위에 의한 이동 방법보다 성능이 우수하였다. LLT가 있는 경우에는 현재 세그먼트의 크기가 커지는 LST-GET에 의한 이동 방법의 성능이 저하되었다. AST-AET에 의한 이동 방법은 시간지원 질의 비율의 변화와 LLT 비율의 변화를 통한 실험에서 시간단위에 의한 이동 방법과 LST-GET에 의한 이동 방법보다 질의에 대한 성능이 우수하였다.

과학적 응용에서 발생하는 군집을 이루는 시간지원 데이터에 적합한 이동 방법으로 제안한 Min-Overlap에 의한 이동 방법은, 질의에 대한 평균 응답 시간에서 AST-AET에 의한 이동 방법과 비슷한 결과를 보였다. Min-Overlap에 의한 이동 방법은 세그먼트 사이에 중복해서 저장되는 데이터 수를 적게 하므로 공간 이용을 측면에서는 AST-ET에 의한 이동 방법보다 효율적이었다.

참 고 문 헌

[1] G. Ozsoyoglu and R. T. Snodgrass, "Temporal and Realime Databases : A Survey," IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.4, pp.511-532, August, 1995.

[2] C. S. Jensen and R. T. Snodgrass. "Temporal Data Management," Technical Report (TR-17), TimeCenter, June, 1997.

[3] V. Kouramajian, "Temporal Databases : Access Structures, Search Methods, Migration Strategies, and Declustering Techniques," Ph. D. Dissertation, The University of Texas at Arlington, 1994.

[4] G. Ozsoyoglu and R. T. Snodgrass, "Temporal and Realime Databases : A Survey," IEEE Transactions on Knowledge and Data Engineering, Vol.7, No.4, pp.511-532, August, 1995.

[5] T. Zurek. "Optimal Interval Partitioning for Temporal Databases," Proceedings of The 3rd Basque International Workshop on Information Technology(BIWIT'97), pp.140-147, July, 1997.

[6] J. Kim and M. Kim, "On Effective Data Clustering in Bitemporal Databases," Fourth International Workshop on Temporal Representation and Reasoning (TIME-97), pp. 54-61, Florida, May, 1997.

[7] I. Ahn and R. T. Snodgrass, "Partitioned Storage for Temporal Databases," Information Systems, Vol.13, No.4, pp. 369-391, 1988.

[8] A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. T. Snodgrass(eds.), *Temporal Databases : Theory, Design, and Implementation*, pp.137-138, Benjamin/cummings, Redwood City, CA, 1994.

[9] H. Yun and K. Kim, "Experimenting with Segmentation and Non-segmentation Methods for Storing Temporal Data," Proceeding of CNDS'98, pp.113-118, San Diego, California, January, 1998.

[10] I. Ahn and R. T. Snodgrass, "Partitioned Storage for Temporal Databases," Information Systems, Vol.13, No.4, pp. 369-391, 1988.

[11] M. H. Bohlen, R. Busatto, and C. S. Jensen, "Point versus Interval-Based Temporal Data Models," Proceedings of 14th International Conference on Data Engineering, pp. 192-200, Orlando, Florida, February, 1998.

[12] K. Norvag, "The Persistent Cache : Improving OID Indexing in Temporal Object-Oriented Database Systems," Proceedings of the 25th International Conference on VLDB, pp.66-77, Scotland, UK, September, 1999.

[13] B. Moon, I. Lopez, and V. Immanuel, "Scalable Algorithm for Large Temporal Aggregation," 16th International Conference on Data Engineering, pp.145-156, San Diego, California, March, 2000.

[14] J. Linan, S. Betty, L. David, and B. Manuel, "The BT-tree : A Branch and Temporal Access Method," Proceedings of the 26th International Conference On VLDB, pp.451-460, Cairo, Egypt, September, 2000.

[15] J. Lee and R. Elmasri, "A Temporal Algebra for an ER-Based Temporal Data Model," Proceedings of 17th International Conference on Data Engineering, pp.33-40, Heidelberg, Germany, April 2001.

[16] S. Giedrius, S. J. Christian, R. T. Snodgrass, "Adaptable Query Optimization and Evaluation in Temporal Middleware," Proceedings of the 2001 ACM SIGMOD, pp.127-138, Santa Barbara, California, May, 2001.

윤 홍 원

e-mail : hwyun@silla.ac.kr

1986년 부산대학교 계산통계학과 졸업(학사)

1990년 한국외국어대학교 경영정보대학원 전자계산학과(이학사)

1998년 부산대학교 대학원 전자계산학과 (이학박사)

1996년~현재 신라대학교(구·부산여자대학교) 컴퓨터정보공학부 조교수

관심분야 : 데이터베이스 시스템, 시간 데이터베이스, 멀티미디어 시스템, XML