시간지원 데이타의 특성을 고려한 AST-AET 데이타 이동 기법

(AST-AET Data Migration Strategy considering Characteristics of Temporal Data)

윤 홍 원 † 김 경 석 ††
(Hongwon Yun) (Gyongsok Gim)

요 약 본 논문에서는 시간지원 데이타를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리한 저장 구조를 기반으로 하는 AST-AET(Average valid Start Time-Average valid End Time) 데이타 이동 방법을 제안한다. 제안한 AST-AET 데이타 이동 방법에서의 AST와 AET를 정의하고 각 세그먼트에 저장되는 개체 버전을 정의한다. AST와 AET를 계산하는 방법과 이동 대상이 되는 개체 버전을 검색하고 이동하는 과정을 보인다. 또한, 제안하는 AST-AET 데이타 이동 방법과 기존의 LST-GET (Least valid Start Time-Greatest valid End Time) 데이타 이동 방법의 사용자 질의에 대한 평균 응답 시간을 비교한다.

실험 결과에 의하면, LLT(Long Lived Tuples)가 없을 때는 현재 세그먼트의 크기가 비슷하기 때문에 두 이동 방법의 평균 응답 시간이 비슷하였다. 그러나 LLT가 있을 때에는 LST-GET 데이타 이동 방법의 현재 세그먼트 크기가 커지기 때문에, AST-AET 데이타 이동 방법의 평균 응답 시간이 LST-GET데이타 이동 방법보다 작았다. 또한, 시간지원 질의의 평균 도착시간을 변화시키면 AST-AET 데이타 이동 방법의 평균 응답 시간이 LST-GET 데이타 이동 방법보다 전반적으로 작았다.

Abstract In this paper, we propose AST-AET(Average valid Start Time-Average valid End Time) data migration strategy based on the storage structure where temporal data is divided into a past segment, a current segment, and a future segment. We define AST and AET which are used in AST-AET data migration strategy and also define entity versions to be stored in each segment. We describe methods to compute AST and AET, and processes to search entity versions for migration and move them. We compare average response times for user queries between AST-AET data migration strategy and the existing LST-GET(Least valid Start Time-Greatest valid End Time) data migration strategy.

The experimental results show that, when there are no LLTs(Long Lived Tuples), there is little difference in performance between the two migration strategies because the size of a current segment is nearly equal. However, when there are LLTs, the average response time of AST-AET data migration strategy is smaller than that of LST-GET data migration strategy because the size of a current segment of LST-GET data migration strategy becomes larger. In addition, when we change average interarrival times of temporal queries, generally the average response time of AST-AET data migration strategy is smaller than that of LST-GET data migration strategy.

1. 서 론

시간지원 데이타베이스(Temporal Databases)에 관한 연구는 시간지원 데이타 모델과 시간지원 질의어[1,2,3,4,5,6]에 집중되었으며, 10여년 전부터 시간지원 데이타의 인덱싱[7,8,9,10]에 관한 연구가 활발하게 진행되고 있으며 최근에는 시공간 데이타베이스 등으로 연구 분

† 정 회 원 : 신라대학교 컴퓨터정보공학부 교수
hwyun@silla.ac.kr
†† 종신회원 : 부산대학교 정보컴퓨터공학부 교수
gimgs@asadal.cs.pusan.ac.kr
논문접수 : 2001년 1월 31일
심사완료 : 2001년 7월 9일

야가 확장되고 있다[11,12,13].

시간지원 데이타 모델과 시간지원 질의어, 그리고 시간지원 인덱스 구조에 관한 연구가 많은 반면에 시간지원 데이타의 저장 방법과 데이타 이동(data migration)에 관한 연구는 많지 않다. 그 동안 구현된 시간지원 데이타베이스 시스템은 기존의 연구용이나 상업용 데이타베이스 관리 시스템에 시간지원 질의어를 처리하는 전처리기를 두는 방식을 주로 사용하고 있으며, 시간지원 데이타의 특성을 고려하지 않고 기존의 데이타 저장 방법대로 시간지원 데이타를 저장하고 있다[14,15].

지금까지 시간지원 데이타베이스에서 다루어 온 시간지원 데이타는 과거에 유효했던 개체 버전과 현재 유효한 개체 버전이 대부분이었으나 최근에는 미래의 특정 기간 동안 유효한 데이타를 고려한 연구 결과가 나오고 있다[15,16,17,19]. 최근에는 경영 정보 시스템이나 의사 결정 지원 시스템 등의 활용이 많아지면서 미래의 계획에 해당하는 데이타의 양이 많아지고, 정보 기술의 발달로 인해서 미래에 대한 예측 데이타가 많이 발생하고 있다. 지금까지 시간지원 데이타베이스와 관련된 연구에서는 미래에 유효한 데이타에 대해서 비교적 관심이 적었는데 본 논문에서는 미래 데이타를 고려한다.

시간지원 데이타의 시간적 특성을 고려하면 과거에 유효했던 개체 버전은 과거 세그먼트에 저장하고, 현재 유효한 개체 버전은 현재 세그먼트에 저장하고, 그리고 미래에 유효한 개체 버전은 미래 세그먼트에 저장할 수 있다. 이와 같이 시간지원 데이타를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리한 저장 구조에서는 시간지원 질의에 대해서 빠른 응답을 보일 뿐만 아니라[15,18], 오래된 개체 버전은 비용과 효율을 고려하여 적절한 제 3의 저장 장치에 저장할 수 있다[8]. 한편, 시간지원 데이타를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리해서 저장하는 분리 저장 구조에서는 시간의 경과에 따라서 세그먼트 사이에 데이타의 이동이 필요하게 된다.

본 논문에서는 시간지원 데이타베이스에서 질의에 대한 검색 속도를 향상시키기 위해서 분리 저장 구조를 기반으로 하는 시간지원 데이타의 이동 방법을 제안한다. 제안하는 이동 방법에서는 각 세그먼트의 경계값을 정의하고 각 세그먼트에 저장되는 개체 버전을 정의한다. 또한, 경계값을 구하는 방법과 이동 대상이 되는 개체 버전을 검색하고 이동하는 과정을 보인다. 그리고 실험을 통해서 본 논문에서 제안하는 데이타 이동 방법과 기존의 데이타 이동 방법 사이에 성능을 비교 분석한다. 본 논문의 구성은 다음과 같다. 2장에서는 시간지원 데이타의 분리 저장 구조와 데이타 이동과 관련된 기존의 연구에 대해서 살펴본다. 3장에서는 시간지원 데이타의 분리 저장 구조를 기반으로 하는 데이타 이동 방법을 제안하고, 4장에서는 기존의 LST-GET 데이타 이동 방법과 본 논문에서 제안하는 AST-AET 데이타 이동 방법에 대해서 질의에 대한 성능을 평가한다. 마지막으로, 5장에서는 결론과 향후 연구 과제에 대해서 살펴본다.

2. 관련 연구

이 장에서는 시간지원 데이타베이스와 관련된 기존의 저장 구조와 시간지원 데이타의 특성을 고려한 분리 저장 구조에 대해서 살펴보고, 시간지원 데이타의 이동에 관한 기존의 연구에 대해서 살펴본다.

Ahn과 Snodgrass는 시간지원 데이타에 대한 검색 속도를 향상시키고 공간을 효율적으로 활용하기 위해서 시간지원 데이타의 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[20].

Sarda는 HDBMS(Historical DBMS)가 수행할 역할로써 이력 릴레이션(historical relation)을 과거, 현재, 그리고 미래 세그먼트로 분리해서 관리하는 방안을 개념적으로 제안하였다[21]. Sarda의 제안에 의하며, 과거 세그먼트에는 유효 종료시간이 현재 시간보다 작은 개체 버전을 저장하고 현재 세그먼트에는 유효 시작시간이 현재 시간보다 작거나 같고 유효 종료시간이 현재 시간보다 크거나 같은 개체 버전을 저장한다. 또한, 미래 세그먼트에는 유효 시작시간이 현재 시간보다 큰 개체 버전을 저장한다.

Kouramajian은 [8]에서 시간지원 데이타의 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 디스크과 자기 디스크에 저장될 개체 버전을 구분하기 위해서 현재 유효한 개체 버전 중에서 가장 작은 유효 시작시간을 경계값으로 사용한다.

Yun은 [15,18,19]에서 시간지원 데이타의 특성을 고려하여 세그먼트로 분리하여 저장하는 저장 방법들을 제안하였다. Yun은 분리 저장 구조에서 데이타 이동 프로세서가 데이타를 이동하기 위한 시기를 결정하는 방법과 데이타를 이동하는 과정을 제안하였으며, LST-GET 데이타 이동 방법의 데이타 이동 알고리즘과 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장

되는 개체 버전에 대해서 정의하였다[19].

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 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장될 개체 버전을 분리하기 위해서 현재 시간을 사용하는 개념적인 방안을 제안하였다. 그러나, 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트의 분리 구조를 관리하는 구체적인 방법은 언급하지 않았다. Kouramajian은 시간지원 데이타를 분리하면서 과거에 유효했던 개체 버전과 현재 유효한 개체 버전만을 고려하였으며 미래에 유효한 개체 버전은 다루지 않았다.

Sarda는 [21]에서 시간지원 데이타의 이동에 대해서 언급하고 있는데 이 연구에 의하면, 미래 세그먼트에서 현재 세그먼트로의 데이타 이동은 현재 시간이 미래 세그먼트에 들어있는 개체 버전의 유효 시간 시간과 같아지면 일어난다. 마찬가지로 현재 세그먼트에 들어있는 개체 버전 중에서 유효 종료시간이 현재 시간과 같아지면 현재 세그먼트에서 과거 세그먼트로 데이타 이동이 발생한다.

Kouramajian은 광 디스크와 자기 디스크에 저장될 개체 버전을 구분하는 경계값을 제시하고 있는데, 현재 유효한 개체 버전 중에서 가장 작은 유효 시작시간을 경계값으로 하고 있다[8]. 경계값보다 작은 유효 종료시간을 가지는 개체 버전은 대부분 광 디스크에 저장하고 참조한다. 경계값보다 유효 시작시간은 작고, 유효 종료시간은 경계값보다 큰 개체 버전은 광 디스크와 자기 디스크 양쪽 모두에 저장하고 참조한다. 유효 시작시간이 경계값보다 큰 개체 버전은 자기 디스크에 저장하고 참조한다.

Yun은 시간지원 데이타의 유효 시간과 현재 시간과의 관계에 따라 과거에 유효했던 개체 버전, 현재 유효한 개체 버전, 미래에 유효한 개체 버전으로 나누고 현재 시간을 기준으로 데이타를 즉시 이동하는 방법을 제안하였다[15,18]. 또한, 현재 시간에 유효한 개체 버전의 유효 시간 집합에서 유효 시작시간이 가장 작은 개체 버전의 유효 시작시간과, 현재 시간에 유효한 개체 버전의 유효 시간 집합에서 유효 종료시간이 가장 큰 개체 버전의 유효 종료시간을 경계값으로 하는 LST-GET 데이타 이동 방법을 제안하였다[19].

본 논문에서는 시간지원 데이타의 평균 경계값 특성을 이용한 AST-AET 데이타 이동 방법(줄여서, AST-AET 이동 방법이라고 부른다)을 제안한다. Kouramajian 이 제안한 데이타 이동 방법을 확장해서 미래 데이타를 처리하는 LST-GET 데이타 이동 방법(줄여서, LST-GET 이동 방법이라고 부른다)의 문제점을 제시하고, 과거, 현재, 그리고 미래 세그먼트를 구분하는 경계값으로 사용하는 *AST*와 *AET*를 정의한다. 데이타 이동에서 옮김과 복사의 대상이 되는 개체 버전과 데이타 이동 알고리즘을 정의하고, 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 저장되는 개체 버전을 정의한다. 또한, 본 논문에서 제안하는 AST-AET 이동 방법과 기존의 LST-GET 이동 방법에 대해서 사용자 질의에 대한 평균 응답시간을 비교한다.

3. 분리 저장 구조에 기반한 AST-AET 이동 방법

이 장에서는 시간지원 데이타의 시간적 특성을 고려한 분리 저장 구조를 보이고 LST(Least valid Start Time)-GET(Greatest valid End Time) 이동 방법의 문제점을 살펴본다. LST-GET 이동 방법의 문제점을 해결하기 위해서 유효 시작시간의 평균과 유효 종료시간의 평균을 사용하는 AST-AET 이동 방법을 제안한다. 먼저, *AST*(*Average valid Start Time*) 와*AET* (*Average valid End Time*)를 정의하고 데이타 이동 과정을 보인 뒤에, 각 세그먼트에 저장되는 개체 버전을 정의한다.

3.1 분리 저장 구조

시간지원 데이타의 시간적 특성을 고려해서 과거 세그먼트, 현재 세그먼트, 미래 세그먼트로 분리한 저장 구조는 그림 1과 같다. 분리 저장 구조에서 DB관리자는 결정기를 통해서 데이타 이동 방법과 세그먼트 사이의 경계값을 결정하는 방법을 선택할 수 있다. 선택할 수 있는 데이타 이동 방법은 LST-GET 이동 방법과 AST-AET 이동 방법이 있으며, 경계값을 결정하는 방법은 각 이동 방법마다 세그먼트를 검색하는 방법과 테이블을 참조하는 방법이 있다. DB관리자가 세그먼트 검색 방법을 선택한 경우에는 검사기가 현재 세그먼트와 미래 세그먼트를 검사하여 이동 방법에 따라 경계값을 계산하고, 테이블을 참조하는 방법을 선택한 경우에는 누적기가 유효 시작시간과 유효 종료시간을 누적하여 이동 방법에 따라 경계값을 계산한다. 결정기가 이동 시기를 결정하면 경계값을 결정하는 방법에 따라 검사기 또는 누적기를 참조하여 경계값을 찾고 데이타 이동을 위해서 실행기를 수행시킨다. 실행기는 데이타 이동 방

법에 따라 세그먼트 사이에 개체 버전을 옮기거나 복사한다.

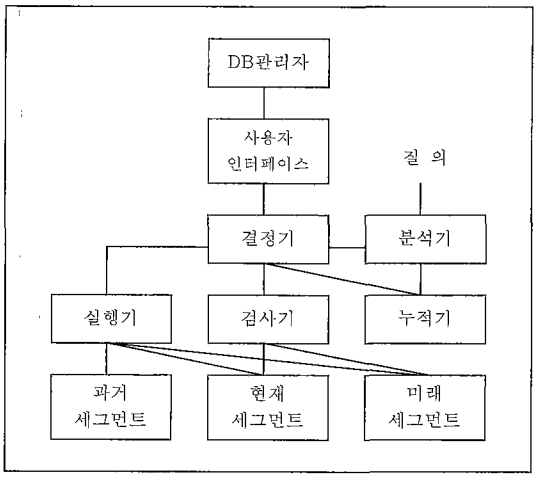

그림 1 분리 저장 구조

3.2 LST- GET 이동 방법의 문제점과 AST-AET 이동 방법의 기본 개념

LST-GET 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, 두 개 또는 세 개의 세그먼트에 걸쳐지면서 상대적으로 유효 시간이 긴 LLT(Long Lived Tuples)가 있으면 확장된 현재 영역의 범위가 커져서 현재 세그먼트에 들어가는 데이타의 수가 지나치게 많아질 수 있다. 이것은 현재 시점에 유효한 데이타에 대한 질의가 비교적 많다는 질의의 특성에 비추어 보면, 질의에 대한 응답 시간이 늘어날 수 있다.

둘째, LLT가 있으면 현재 세그먼트에 들어있는 데이타가 많아지므로 데이타를 이동하는 시점에 다른 세그먼트로 이동해야 하는 대상 데이타 수가 지나치게 많아져서 데이타 이동에 따른 부담을 분산시키지 못하고 집중시키는 문제점을 가지고 있다.

*LST*와 *GET*를 이용한 이동 방법을 사용함으로써 현재 세그먼트의 크기가 지나치게 커지는 것을 막기 위해서 각 개체 버전이 가지고 있는 유효 시작시간과 유효 종료시간의 평균을 각각 이용하는 방법을 생각해 볼 수 있는데, 이것이 AST-AET 이동 방법의 기본 개념이다. 유효 시작시간의 평균은 과거 세그먼트와 현재 세그먼트를 구분하는 경계값으로 사용하고, 유효 종료시간의 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, 개체 버전의 대다수가 아주 긴 수명을 가지고 있다면 아주 긴 수명의 데이타 특성대로 경계값을 구할 수 있으므로 현재 세그먼트의 크기가 지나치게 커지는 것을 막을 수 있다.

3.3 AST와 AET의 정의

과거 세그먼트와 현재 세그먼트를 나누는 경계값으로 사용하는 유효 시작시간의 평균값을 *AST*라 하고 현재 세그먼트와 미래 세그먼트를 나누는 경계값으로 사용하는 유효 종료시간의 평균값을 *AET*라 부르기로 한다. 현재 시간에 유효한 개체 버전의 집합을 *C*라 하면 다음과 같다.

$$C = \{\ E_{ij} \mid E_{ij}.V_s \ \ now < E_{ij}.V_e\ \}$$

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

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

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

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

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

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

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

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

모든 유효 시작시간의 평균을 나타내는 *AST*는 다음

과 같다.

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

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

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

모든 유효 종료시간의 평균을 나타내는 AET는 다음과 같다.

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

경계값 AST는 과거 세그먼트와 현재 세그먼트에 들어 갈 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.

```
GetAST()
begin
/* for all E_ij in Future Segment and Current Segment */
  ASTpre <- AST;  AST <- Null;  ASTsum <- n <- 0;
  while (not eof (EntityVersion <- read (FutSeg))) do
   if ( EnitityVersion.V_s  now < EntityVersion.V_e ) ∧
      (EntityVersion.V_s > ASTpre) then
       ASTsum <- ASTsum + EntityVersion.V_s;
       n <- n +1;
   end if
  end while
    while (not eof (EntityVersion <- read (CurSeg))) do
     if ( EnitityVersion.V_s  now < EntityVersion.V_e ) ∧
        (EntityVersion.V_s > ASTpre) then
         ASTsum <- ASTsum + EntityVersion.V_s;
         n <- n +1;
     end if
  end while
  if  (ASTsum = Null) then
      AST <- Notfound;
  else AST <- ASTsum / n;
  end if
  return AST;
end
```

그림 2 AST를 구하는 알고리즘

3.4 AST와 AET를 구하는 방법

현재 시간이 AST와 AET의 사이에 놓여 있다가 시간이 흐름에 따라 현재 시간이 AET와 같아지면 새로운 AST와 AET를 구하게 된다. 새로운 AST와 AET를 구하는 방법은 LST-GET 이동 방법에서 살펴 본 바와 같이 세그먼트를 검색하는 방법과 AST-AET 테이블을 참조하는 방법이 있다.

1) 세그먼트를 검색하는 방법

세그먼트를 검색하는 방법에서는 현재 시간에 유효한 모든 개체 버전을 검색해서 유효 시작시간을 더한 다음에 개체 버전의 개수로 나눔으로써 AST를 구할 수 있다. 모든 유효 종료시간을 더한 다음에 개체 버전의 개수로 나눈 결과는 AET가 된다. 그림 2는 세그먼트를 검색해서 AST를 구하는 과정을 나타낸 것이다. AST보다 큰 유효 시작시간을 가지는 개체 버전 가운데 현재 시간에 유효한 개체 버전의 유효 시작시간을 모두 더하고, 더하는 대상이 된 개체 버전의 개수로 나누어서 평균을 구함으로써 AST를 구할 수 있다. 그림 3은 AET보다 큰 유효 종료시간을 가지는 모든 개체 버전의 유효 종료시간의 평균을 구하는 과정을 나타낸 것이다.

```
GetAET()
/* for all E_ij in Future Segment and Current Segment */
AETpre <- AET;  AET <- Null;  AETsum <- n <- 0;
begin
  while (not eof (EntityVersion <- read (FutSeg))) do
   if ( EnitityVersion.V_s  now < EntityVersion.V_e ) ∧
      (EntityVersion.V_e > AETpre) then
       AETsum <- AETsum + EntityVersion.V_e;
       n <- n +1;
   end if
  end while
    while (not eof (EntityVersion <- read (CurSeg))) do
     if ( EnitityVersion.V_s  now < EntityVersion.V_e ) ∧
        (EntityVersion.V_e > AETpre) then
         AETsum <- AETsum + EntityVersion.V_e;
         n <- n +1;
     end if
  end while
  if  (AETsum = Null) then
      AET <- Notfound;
  else AET <- AETsum / n;
  end if
  return AET;
end
```

그림 3 AET를 구하는 알고리즘

2) 테이블을 참조하는 방법

AST-AET 테이블을 참조하는 방법은 새로 삽입되거나 갱신되는 시간지원 데이타의 개수와 유효 시작시간과 유효 종료시간을 누적해서 AST-AET 테이블에 기록해 두었다가 새로운 경계값을 구하는 시점에서 이 값들을 참조하는 것이다.

*AST*와 *AET*와 관련된 테이블은 현재 AST-AET테이블과 예상 AST-AET테이블을 유지한다. 현재 AST-AET테이블은 현재 사용하고 있는 *AST*와 *AET*를 유지하고 있다. 예상 AST-AET테이블은 새로운 경계값을 구하는데 필요한 정보를 유지하고 있다.

과거 세그먼트와 미래 세그먼트에 들어갈 개체 버전을 구분하는데 사용하는 *AST*와 현재 세그먼트와 미래 세그먼트에 들어 갈 개체 버전을 구분하는데 사용하는 *AET*를 구하기 위해서 예상 AST-AET테이블에는 유효 시작시간과 유효 종료시간을 따로 누적한다. 새로운 경계값을 구하는 시점에서 그때까지 누적된 유효 시작시간을 누적된 개체 버전의 개수로 나누어서 새로운 경계값 *AST*와 *AET*를 구할 수 있다. AST-AET테이블에서 누적 유효 시간과 누적 유효 종료시간을 유지하는 방법을 살펴보면 다음과 같다.

E_{ij}가 데이타베이스에 저장되는 시점을 $E_{ij}.t$라고 하면 $NewE_{ij}$의 집합은 다음과 같이 정의한다.

$$NewE = \{E_{ij} \mid (E_{ij}.V_s < E_{ij}.t < E_{ij}.V_e)\}$$

유효 시작시간과 유효 종료시간을 누적하기 위한 대상이 되는 개체 버전의 집합 S를 정의하면 다음과 같다.

$$\begin{aligned} S = \{E_{ij} \mid & ((E_{ij}.V_s < AST) \wedge (AST < E_{ij}.V_e)) \\ & \vee ((AST \le E_{ij}.V_s < AET) \wedge (AET \le E_{ij}.V_e)) \\ & \vee ((E_{ij}.V_s \le AET) \wedge (AET < E_{ij}.V_e)) \\ & \wedge E_{ij} \in NewE \} \end{aligned}$$

S에 들어가는 개체 버전의 유효 시작시간의 누적 합계를 $C(Vs)$라 하면 다음과 같이 정의된다.

$$C(Vs) = \bigcup_{Eij,\ V_s \in S} E_{ij}.V_S$$

S에 들어가는 개체 버전의 유효 종료시간의 누적 합계를 $C(Ve)$라 하면 다음과 같이 정의된다.

$$C(Ve) = \bigcup_{Eij,\ V_e \in S} E_{ij}.V_e$$

S에 들어가는 모든 개체 버전의 개수를 n이라고 하면 *AST*와 *AET*를 정의하면 각각 다음과 같다.

$$AST = C(Vs) / n$$

$$AET = C(Ve) / n$$

AST-AET테이블을 이용하는 방법에서는 *AST*와 *AET*에 오차가 생길 수 있다. 현재 시간에 유효한 개체 버전의 유효 시작시간과 유효 종료시간 정보가 누적되었다 할지라도 현재 시간이 흐르면서 어떤 개체 버전의 유효 시간이 현재 시간을 벗어날 수 있다. 현재 시간이 *AET*와 같아져서 새로운 경계값을 구하는 시점에는 그동안 누적되었던 개체 버전 가운데서 어떤 개체 버전은 더 이상 현재 시간에 유효하지 않는 것도 있을 수 있다. 새로운 경계값을 갱신하는 시점에서 그 동안 누적된 개체 버전의 유효 시간에는 오차가 있을 수 있으나 테이블을 이용하는 방법에서는 경계값을 구하는데 오차를 허용한다.

3.5 데이타 이동 과정

과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트에 들어가는 개체 버전의 집합과 두 세그먼트에 중복해서 저장되는 개체 버전은 다음과 같이 정의한다.

- 과거 세그먼트에 들어가는 개체 버전의 집합
  $= \{E_{ij} \mid E_{ij}.V_e < AST\} = P$
- 현재 세그먼트에 들어가는 개체 버전의 집합
  $= \{E_{ij} \mid (E_{ij}.V_s \ge AST) \wedge (E_{ij}.V_e < AET)\} = C$
- 미래 세그먼트에 들어가는 개체 버전의 집합
  $= \{E_{ij} \mid E_{ij}.V_s \ge AET\} = F$
- 과거 세그먼트와 현재 세그먼트에 중복해서 들어가는 개체 버전의 집합
  $= \{E_{ij} \mid (E_{ij}.V_s < AST) \wedge (AST < E_{ij}.V_e \le AET)\}$
  $= PC$
- 현재 세그먼트와 미래 세그먼트에 중복해서 들어가는 개체 버전의 집합
  $= \{E_{ij} \mid (AST \le E_{ij}.V_s < AET) \wedge (E_{ij}.V_e > AET)\}$
  $= CF$

AST-AET 이동 방법에서 데이타 이동은 옮김과 복사로 구분할 수 있다. AST-AET에 의한 이동에서 각 세그먼트 사이에 개체 버전을 옮기거나 복사하는 경우는 표 1과 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. 미래 세그먼트에서 현재 세그먼트로 데이타를 이동하는 과정과 현재 세그먼트에서 과거 세그먼트로 데이타를 이동하는 과정

표 1 개체 버전의 옮김과 복사

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

은 표 1과 같다. 표 2는 각 세그먼트에 저장되는 개체 버전을 나타낸 것이다.

표 2 AST-AET 이동 방법에서 과거 세그먼트, 현재 세그먼트, 미래 세그먼트에 저장되어 있는 개체 버전

과거 세그먼트	현재 세그먼트	미래 세그먼트
$P \vee PC$	$C \vee CF \vee PC$	$F \vee CF$

4. 성능 평가

3장에서는 분리 저장 구조를 기반으로 한 AST-AET 이동 방법을 제안하였다. 이 장에서는 AST-AET 이동 방법에 대해서 사용자 질의에 대한 평균 응답시간을 비교한다. 먼저, 4.1절에서는 실험 환경을 살펴보고, 4.2절에서는 실험에 대한 결과를 분석하도록 한다.

4.1 실험 환경

분리 저장 구조에서 LST-GET 이동 방법과 AST-AET 이동 방법 사이에 성능을 비교하기 위해서 그림1과 같은 실험 모델에서 사용자 질의에 대한 평균 응답 시간을 측정한다. 모의 실험에 사용할 데이타는 TimeIT (the Time-Integrated Testbed)를 이용해서 만들었다. TimeIT는 미국 애리조나 대학의 TimeCenter에서 개발한 시간지원 질의의 연산 알고리즘을 테스트하기 위한 시험도구이다[22].

표 3 모의 실험에 사용한 하드 디스크의 성능 매개변수와 값

설 명	값
평균 탐색 시간, 읽기/쓰기(Average Seek, Read/Write(msec))	7.7 / 8.7
트랜간 탐색 시간, 읽기/쓰기(Track-To-Track Seek, Read/Write (msec))	0.98 / 1.24
평균 지연 시간(Average Latency(msec))	2.99
데이타 전송율(Transfer Rate(Bytes/sec))	16,777,216
트랙 크기 (Track Size (Bytes))	115,078

모의 실험을 위한 알고리즘은 C언어로 작성하였고 모의 실험은 선(SUN)사의 울트라 스팍(ULTRA SPARC)에서 수행하였다. 모의 실험에 사용할 데이타는, 일반적인 데이타의 유효 시간의 길이는 30에서 50 사이에 단위가 1인 크기로 만들었고 각각의 길이는 균등 분포(uniform distribution)로 생성하였다. LLT(Long Lived Tuples)는 일반적인 튜플의 10배 정도 큰 것으로 가정하고 LLT의 길이는 300에서 500 사이의 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 데이타를 말하는 것이므로 LLT의 비율이 10%가 넘는 것은 실험에서 고려하지 않는다. 모의 실험에서는 시게이트(Seagate)사에서 나온 최근 제품인 3.5 인치 ST34501N의 하드디스크 성능 인수값을 사용하였다[표 3].

사용자 질의의 종류는 시점 질의와 범위 질의로 구분하였으며 모의 실험에서는 다음과 같이 세 가지 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.

모의 실험에서 사용자 질의에 대한 평균 응답 시간을 구하는 식에 적용한 가정은 다음과 같다.

첫째, 한 블록은 한 릴레이션의 튜플만을 포함한다. 둘째, 릴레이션은 비시간지원 키(non-temporal key)로 색인되어 있으므로 시간지원 사용자 질의는 릴레이션의 모든 튜플을 검색한다. 셋째, 사용자 질의의 평균 응답 시간에는 데이타 이동시의 지연과 질의 처리의 대상이 되는 릴레이션의 크기만을 고려한다.

분리 저장 구조에서 사용자 질의에 대한 평균 응답 시간을 계산하는데 사용하는 기호와 의미는 그림 4와 같다.

임의의 질의를 수행하는 시간은 해당하는 릴레이션을 찾기 위한 한 번의 임의 탐색 시간 (random seek), 릴레이션의 한 세그먼트가 차지하고 있는 트랙 만큼의 순차 탐색 시간(sequential seek), 회전 지연 시간(rotational latency), 그리고 세그먼트에 들어있는 튜플의 전송 시간의 합으로 계산할 수 있다. 따라서, 한 개의 질의의 수행 시간, T_{query}는 다음과 같이 나타낼 수 있다.

$$T_{query} = T_{rand}^{r\_seek} + S_{seg} \times (T_{seq}^{r\_seek} + T_{latn}) + T_{trans}$$

한 세그먼트가 차지하는 공간을 트랙의 수로 나타내면 다음과 같다.

$$S_{seg} = (N_{tupl} \times S_{tupl}) / S_{track}$$

한 세그먼트를 전송하는데 걸리는 시간, T_{trans}는 다음과 같다.

$$T_{trans} = (N_{tupl} \times S_{tupl}) / T_{trans}^{rate}$$

T_{query} : 한 개의 질의를 수행하는데 걸리는 시간
$T_{rand}^{r\_seek}$: 디스크 평균 탐색 시간 (읽기)
$T_{rand}^{w\_seek}$: 디스크 평균 탐색 시간 (쓰기)
T_{latn} : 디스크 평균 지연 시간
T_{trans} : 데이터 전송 시간
$T_{seq}^{r\_seek}$: 트랙간 탐색 시간 (읽기)
$T_{seq}^{w\_seek}$: 트랙간 탐색 시간 (쓰기)
S_{tupl} : 튜플의 크기
N_{tupl} : 튜플의 개수
T_{mig} : 데이터 이동에 걸리는 시간
T_{scan} : 이동 대상 세그먼트를 검색하는데 걸리는 시간
T_{write}^{move} : 이동 대상 튜플을 다른 세그먼트에 쓰는 시간
T_{write}^{back} : 이동한 튜플을 지우는 시간
T_{delay} : 질의당 평균 지연 시간
Q_{avg} : 사용자 질의에 대한 평균 응답 시간

그림 4 사용자 질의에 대한 평균 응답 시간을 구하는데 사용하는 기호와 의미

데이타를 이동하는데 걸리는 시간은 이동 대상 세그먼트를 검색하는 시간, 이동 대상 튜플을 다른 세그먼트에 쓰는 시간, 그리고 이동한 튜플을 원래의 세그먼트에서 지우는 시간을 합해서 나타낼 수 있으며, 다음과 같다.

$$T_{mig} = T_{scan} + T_{write}^{move} + T_{write}^{back}$$

이동 대상 세그먼트를 검색하는데 걸리는 시간 T_{scan}은 다음과 같다.

$$T_{scan} = T_{rand}r\_seek + S_{seg} \times (T_{seq}^{r\_seek} + T_{latn}) + T_{trans}$$

이동 대상인 튜플을 다른 세그먼트에 쓰는 시간과 이동한 튜플을 원래의 세그먼트에서 지우는 시간은 다음과 같이 나타낼 수 있다.

$$T_{write}^{move} = T_{rand}w\_seek + S_{seg} \times (T_{seq}^{w\_seek} + T_{latn}) + T_{trans}$$

데이타를 이동하는 시간(T_{mig}) 동안 도착한 질의의 수를 N_{Tmig}라고 하면, T_{mig} 동안 도착한 각 질의의 대기 시간의 합은 다음과 같다.

$$\begin{aligned} SUM_{Tmig}^{Q} &= \sum_{i=1}^{N_{Tmig}} (1 - \frac{i}{N_{Tmig}}) \times T_{mig} \\ &= \frac{1}{N_{Tmig}} (\frac{N_{Tmig}(N_{Tmig}-1)}{2} \times T_{mig}) \\ &= (\frac{N_{Tmig}-1}{2}) \times T_{mig} \end{aligned}$$

데이타를 이동하기 전에 질의 대기 큐에 기다리고 있던 모든 질의는 T_{mig} 동안 대기하게 된다.

따라서, 데이타 이동으로 인한 질의당 평균 대기 시간은 다음과 같다.

$$T_{avg} = \frac{SUM_{Tmig}^{Q} + (N_{QUEUE} \times Tmig)}{Q_{arrival} \times c}$$

데이터를 이동하는 동안에 도착한 질의의 수 N_{Tmig}은 $\frac{Qarrival}{1,000} \times T_{mig}$이고, $Q_{arrival}$은 질의 도착율로서 초당 평균 도착 질의의 수를 나타낸다.

따라서, 분리 저장 구조에서 사용자 질의에 대한 평균 응답 시간은, 질의를 수행하는데 걸리는 시간과 데이타 이동으로 인한 질의당 평균 지연 시간의 합으로 나타낼 수 있다.

$$Q_{avg} = T_{query} + T_{delay}$$

4.2 성능 분석

그림 5는 분리 저장 구조에서 시간지원 질의를 10%에서 50%까지 바꾸어 가면서 LLT의 비율이 0%인 경우에 LST-GET에 의한 이동과 AST-AET 이동 방법에서 사용자 질의에 대한 평균 응답 시간을 나타낸 것이다. 그림 7에서 시간지원 질의가 10% 인 경우에는 두 가지 데이타 이동 방법의 사용자 질의에 대한 평균 응답 시간이 거의 비슷하게 나타난다. LLT가 없으므로 두 가지 데이타 이동 방법에서 현재 세그먼트의 크기가 비슷하기 때문에 성능의 차이가 크게 나타나지 않는다.

그림 6은 LLT가 있는 경우에 분리 저장 구조에서 시간지원 질의의 비율을 바꾸어 가면서 LST-GET 이동 방법과 AST-AET 이동 방법에 대해서 사용자 질의에

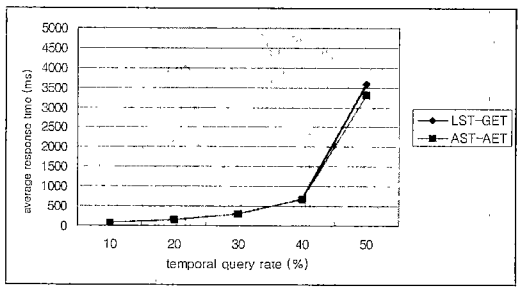

그림 5 LLT가 없는 경우 질의에 대한 평균 응답 시간

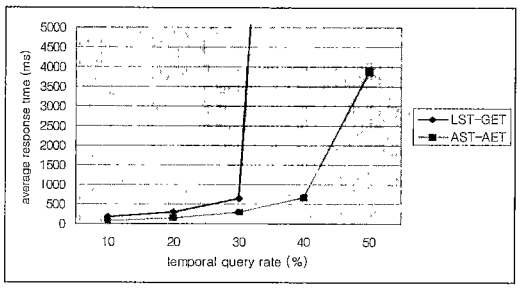

그림 6 LLT가 있는 경우 질의에 대한 평균 응답 시간

대한 응답 시간을 나타낸 것이다. LLT가 있으면 LST-GET 이동 방법에서는 *LST*가 작아지고 *GET*가 커지므로 현재 세그먼트에 들어갈 개체 버전을 포함하는 범위가 늘어나서 현재 세그먼트의 크기가 급격히 커진다. 반면에, AST-AET 이동 방법은 현재 세그먼트의 크기 변화가 비교적 작다. 현재 세그먼트를 검색하는 현재 질의가 많은 경우에 LLT가 있으면 현재 세그먼트의 크기가 커지는 LST-GET 이동 방법이 현재 세그먼트의 크기 변화가 작은 AST-AET 이동 방법보다 질의에 대한 응답 속도가 급속히 저하된다.

바로 앞에서 LST-GET 이동 방법은 LLT가 있는 경우에 현재 세그먼트의 크기가 커져서 질의에 대한 응답 속도가 급속히 저하되는 것을 보았다. 그림7는LLT의 변화에 대해서 LST-GET 이동 방법의 응답 시간을 나타낸 것이다. LLT가 0%에서 1%로 바뀌면서 현저한 성능의 감소를 나타내는 것을 보이고 있다. 그림8은 AST-AET 이동 방법이 LLT의 변화에 비교적 안정적으로 반응하는 것을 나타내고 있다. AST-AET 이동 방법이 유효 시작시간과 유효 종료시간의 평균을 이용하기 때문에 LLT의 비율의 변화에 대해서 현재 세그먼트의 크기가 급격한 변화를 보이지 않으므로 비교적 안정된 성능을 나타낸다.

그림 9는 LST-GET 이동 방법과 AST-AET 이동 방법 각각에 대해서 LLT 비율의 변화에 대해서 서로 질의에 대한 응답 시간을 비교한 것이다. 이미 그림 7에서 본 것처럼 LST-GET 이동 방법은 LLT가 있는 경우와 없는 경우에 대해서 민감한 반응을 보이지만, AST-AET 이동 방법은 질의에 대해서 안정된 성능을 보이고 있다.

그림 10은 사용자 질의가 도착하는 평균 시간 간격의 변화에 대해서 LST-GET 이동 방법과 AST-AET 이동 방법의 평균 응답 시간을 비교한 것이다. 그림 10의

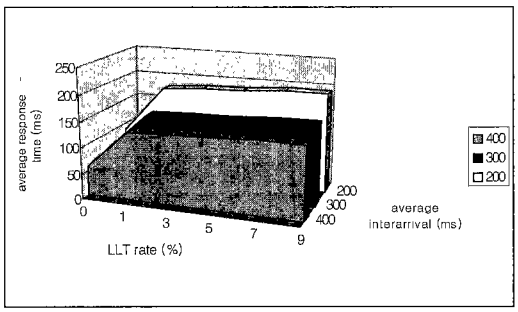

그림 7 LST-GET 이동 방법에서 LLT 비율의 변화에 따른 질의에 대한 평균 응답 시간

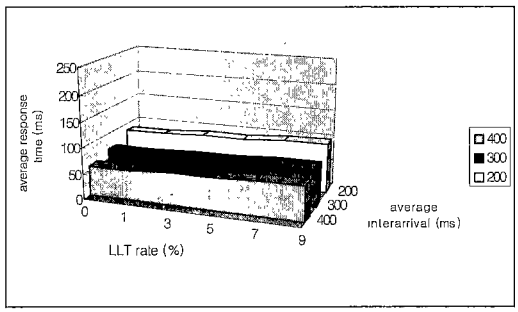

그림 8 AST-AET 이동 방법에서 LLT 비율의 변화에 따른 질의에 대한 평균 응답 시간

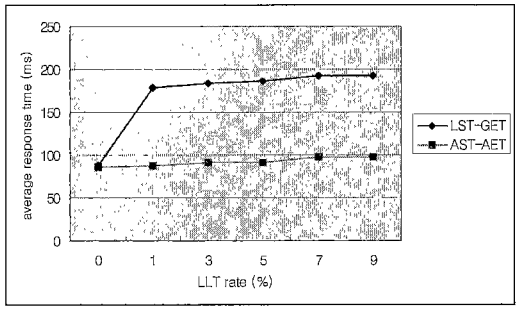

그림 9 LLT 비율의 변화에 따른LST-GET 이동 방법과 AST-AET 이동 방법의 질의에 대한 평균 응답 시간

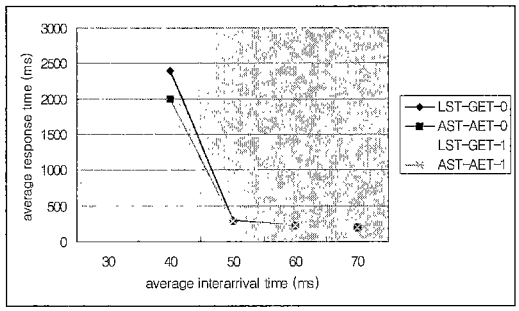

그림 10 질의 평균 도착 시간의 변화에 따른 평균 응답 시간

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-GET 이동 방법의 성능 저하가 가장 일찍나타났다. 이것은 LLT가 있는 경우에는 LST-GET 이

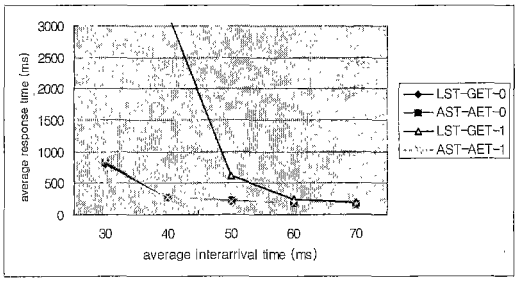

그림 11 질의 평균 도착 시간의 변화에 따른 평균 응답 시간

동 방법에서 현재 세그먼트의 크기가 급격히 커지기 때문에 성능의 저하가 가장 일찍 나타났으며 현격한 차이로 인하여 그림의 영역에 넣을 수가 없었다. 그림 11은 성능의 차이를 그림의 영역으로 표현하기 위해서 일반적인 개체 버전의 수명을 5에서 10사이로 조절하여 실험한 결과를 나타낸 것이다.

그림 11은 일반적인 개체 버전의 수명이 5, 6, ..., 10인 경우에, LLT가 있는 경우와 LLT가 없는 경우에 대해서 질의에 대한 평균 응답 시간을 나타낸 것이다. LLT가 있는 경우에는 LST-GET 이동 방법이 현재 세그먼트의 크기가 다른 AST-AET 이동 방법보다 크기 때문에 질의의 평균 도착 시간이 작아짐에 따라 성능의 저하가 빨리 나타났다. LLT가 없는 경우의 LST-GET 이동 방법과 LLT의 유무에 상관없이 AST-AET 이동 방법이 가장 나중에 성능의 저하가 나타났다.

5. 결 론

시간지원 데이타의 시간적 특성을 고려하면 과거에 유효했던 개체 버전은 과거 세그먼트에 저장하고, 현재 유효한 개체 버전은 현재 세그먼트에 저장하고, 그리고 미래에 유효한 개체 버전은 미래 세그먼트에 저장할 수 있다. 본 논문에서는 시간지원 데이타를 과거 세그먼트, 현재 세그먼트, 그리고 미래 세그먼트로 분리한 저장 구조를 기반으로 하고 시간지원 데이타의 평균 경계값 특성을 이용한 데이타 이동 방법을 제안하였다.

기존의 LST-GET 이동 방법은 LLT가 있는 경우에 현재 세그먼트가 커지므로 검색 속도가 급속히 저하되는데, 이 문제점을 해결하기 위해서 AST-AET 이동 방법을 제안하였다. AST-AET 이동 방법에서 *AST*와 *AET*를 정의하고 *AST*와 *AET*를 경계값으로 각 세그먼트에 저장되는 개체 버전을 정의하였다. *AST*와 *AET*를 구하는 방법을 보이고, 이동 대상이 되는 개체 버전을 검색하고 이동하는 과정을 보였다. 또한, 본 논문에서 제안하는 AST-AET 이동 방법과 기존의 LST-GET 이동 방법에 대해서 사용자 질의에 대한 성능을 평가하였다.

사용자 질의에 대한 성능 평가에서LLT가 없는 경우에는 현재 세그먼트의 크기가 비슷하기 때문에 성능의 차이가 크게 나타나지 않았지만, LLT가 있는 경우에는 현재 세그먼트의 크기가 커지는 LST-GET 이동 방법의 성능이 저하되었다. AST-AET 이동 방법은 시간지원 질의 평균 도착 시간의 변화와 LLT 비율의 변화를 통한 실험에서 LST-GET 이동 방법보다 질의에 대한 성능이 우수함을 보였다.

본 논문에서 제안한 이동 방법은 시간이 흐름에 따라 데이타의 양이 지속적으로 증가하는 시간지원 데이타베이스에서 효율적인 데이타 이동 방법으로 활용할 수 있을 것이다. 제안한 이동 방법은 시점 질의에 대한 검색 속도를 향상시키기 위해서 두 개 세그먼트 또는 세 개 세그먼트에 중복해서 저장되는데, 향후에는 중복의 최소화와 중복된 개체 버전의 일관성을 유지하는 방법에 대한 연구가 필요하다.

참 고 문 헌

[1] R.T. Snodgrass, "Temporal Databases: Status and Research Directions," ACM SIGMOD Record, vol. 19, no. 4, pp. 83-89, December 1990.

[2] R.T. Snodgrass(ed.), *Proceeding of the International Workshop on an Infrastructure for Temporal Databases, Arlington*, TX, June 1993.

[3] C.S. Jensen, M.D. Soo, and R.T. Snodgrass, "Unifying Temporal Models via a Conceptual Model," Information Systems, vol. 19, no. 7, pp. 513-547, December 1994.

[4] C.S. Jensen and R.T. "Snodgrass. Semantics of Time-Varying Information, Information Systems," vol. 21, no. 4, pp. 311-352, 1996.

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

[6] H. Gregersen and C. S. Jensèn, "Temporal Entity-Relationship Models-a Survey," IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 3, pp.464-497, May/June 1999,

[7] R. Elmasri, M. Jaseemuddin, and V. Kouramajian, "Partitioning of Time Index for Optical Disks," Proceedings of the International Conference on Data Engineering, Phoenix, AZ, pp. 574-583, February 1992.

[8] V. Kouramajian, *Temporal Databases: Access Structures, Search Methods, Migration Strategies, and Declustering Techniques*, Ph. D. Dissertation, pp. 101-118, The University of Texas at Arlington, 1994.

[9] M.A. Nascimento, M.H. Dunham, and R. Elmasri. "M-IVTT : A Practical Index for Bitemporal Databases," Proceedings. of the 7th International Conference on Database and Expert Systems Applications(DEXA'96), September 1996.

[10] B. Salzberg and V. J. Tsotras, "Comparison of access methods for time-evolving data," ACM Computing Surveys , Vol. 31, No. 2, pp. 158-221, 1999.

[11] M. Bhlen, C. S. Jensen, and B. Skjellaug, "Spatio-Temporal Database Support for Legacy Applications," in Proceedings of the 1998 ACM Symposium on Applied Computing, pp.226-234., Atlanta, Georgia, February 27-March 1, 1998.

[12] N. Tryfona and C. S. Jensen, "Conceptual Modeling for Spatiotemporal Applications," Geoinformatica, Vol. 3, No. 3, October 1999, pp.245-268.

[13] N. Tryfona, S. Andersen, S. R. Mogensen, and C. S. Jensen, "A Methodology and a Tool for Spatiotemporal Database Design," in Proceedings of the Seventh Hellenic Conference on Informatics, pp. 53-60, Ioannina, Greece, August 26-29, 1999.

[14] J. Skyt and C. S. Jensen, "Vacuuming Temporal Databases," TimeCenter Technical Report TR-32, 20 pages, September 1998.

[15] 윤홍원, 김경석, "시간지원 데이타베이스에서 자료의 이동을 고려한 저장 방법의 성능 평가", 정보과학회논문지(B) 제25권 제5호, pp. 756-767, 1998년 5월.

[16] S. Kim and S. Charkravarthy, "Temporal Databases with Two-Dimensional Time: Modeling and Implementation of Multihistory," Information Sciences, vol.80, nos.1-2, pp.43-89, September 1994.

[17] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass, "On the Semantics of 'Now' in Databases," ACM Transactions on Database Systems. Vol. 22, No. 2, pp.171-214, June 1997.

[18] 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, CA, January 1998.

[19] 윤홍원, 김경석, "시간지원 데이타를 위한 분리 저장 구조와 데이타 이동 방법", 한국정보처리학회 논문지 제6권 제4호., pp. 851-867, 1999년 4월.

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

[21] 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.

[22] http://www.cs.auc.dk/TimeCenter

윤 홍 원

1986년 부산대학교 계산통계학과 학사. 1990년 한국외국어대학교 경영정보대학원 전자계산학과 석사. 1998년 부산대학교 전자계산학과 박사. 1996년 ~ 현재 신라대학교 컴퓨터정보공학부 조교수. 관심분야는 데이타베이스 시스템, 멀티미디어 시스템

김 경 석

1977년 서울대학교 무역학과 학사. 1979년 서울대학교 계산통계학과 석사. 1988년 미국 일리노이 주립대학(어바나 샴페인) 전산학 박사. 1988년 ~ 1992년 미국 노스다코타 주립대학교 전산학과 조교수. 1992년 ~ 현재 부산대학교 정보컴퓨터공학부 교수. 관심분야는 데이타베이스 시스템, 멀티미디어 시스템, 한글정보처리