센서 데이터의 시간 정보를 이용한 이력 정보 관리

(Historical Sensor Data Management Using Temporal Information)

이 양 구* 류 근 호**
(Yang Koo Lee) (Keun Ho Ryu)

요 약 무선 센서 네트워크는 공간상에 분포되어 거의 무제한적이고, 연속적으로 데이터를 발생시키며, 주로 샘플링이나 시놉시스 등을 이용하여 수집되는 데이터를 저장한다. 그러나 이러한 방법은 원시 데이터의 정확한 저장 및 장기간 수집된 이력 데이터로부터의 분석을 요구하는 응용 환경에는 적합하지 않다. 따라서 이 논문에서는 센서로부터 수집된 데이터를 손실 없이 저장하는 동시에 시간에 따라 누적되는 센서 데이터의 이력을 효율적으로 관리할 수 있는 저장 기법을 제안한다. 제안된 방법에서는 센서의 측정값이 변경된 시점을 기준으로 동일한 값이 지속된 기간을 Time-segment와 Time- point로 저장한다. 이러한 방법은 센서의 모든 측정값을 정확하게 저장할 수 있고, 특정 시간에 걸쳐 중복되는 데이터들을 하나의 튜플에 저장하기 때문에 장기간 동안 수집된 데이터를 메모리에서 유지할 수 있다.

키워드 : 센서 네트워크, 데이터 스트림, 데이터 스트림 관리 시스템, 이력 센서 데이터

Abstract A wireless sensor network consists of many sensors that collect and transmit physical or environmental conditions at different locations to a server continuously. Many researches mainly focus on processing continuous queries on real-time data stream. However, they do not concern the problem of storing the historical data, which is mandatory to the historical queries. In this paper, we propose two time-based storage methods to store the sensor data stream and reduce the managed tuples without any loss of information, which lead to the improvement of the accuracy of query results.

Key words : Sensor Network, Data Stream, Data Stream Management System, Historical Sensor Data

† 본 연구는 교육과학기술부와 한국산업기술재단의 지역혁신인력양성사업 및 2008년 교육과학기술부로부터 지원받아 수행된 연구임(지역거점연구단육성사업/충북BIT연구중심대학육성사업단).

* 충북대학교 전자계산학과 박사과정, leeyangkoo@dblab.chungbuk.ac.kr
** 충북대학교 전기전자컴퓨터공학부 교수, khryu@chungbuk.ac.kr
(교신저자)

논문접수 : 2008.11.17
수 정 일 : 2008.11.18
심사완료 : 2008.11.19

1. 서론

무선 센서 네트워크는 물리적 공간에서 연속적으로 통신하면서 데이터를 수집하는 수많은 센서들로 구성된다. 공간상에 분포된 센서들은 주로 온도, 조도, 소리, 압력 등과 같은 주변 환경 정보를 실시간으로 측정하여 서버로 전송하며, 전송된 정보들은 응용에 따라 지구 환경 모니터링, 의료 환자 모니터링, 홈 네트워크 등에 다양하게 활용되고 있다[1, 2].

센서 데이터는 시간의 흐름에 따라 거의 무제한적이고, 연속적으로 데이터를 발생시키기 때문에 모든 데이터를 데이터베이스에 저장하지 않고, 샘플링이나 시놉시스와 같은 방법으로 요약 데이터를 저장하고 근사 값을 계산하여 사용자의 연속 질의를 처리한다. 그러나 이러한 방법은 질의 처리 효율은 높일 수 있는 반면에, 수집된 원시 데이터의 손실로 인해 데이터의 정확성을 보장하기 어려운 단점이 있다. 특히, 생태계 모니터링, 기후 모니터링, 대기 오염 모니터링 등과 같이 수집된 데이터의 정확한 저장을 요구하는 응용 환경에서 기존의 시스템을 그대로 적용하기에는 한계가 있다[3, 4, 5]. 또한, 기간별 분석이나 통계와 같은 범위 질의를 처리하기 위해서는 장기간 수집된 데이터의 이력을 저장하여야 하는데, 실시간으로 발생되는 모든 센서 데이터를 메모리에 저장하는 것은 불가능하며, 수집된 데이터를 디스크에 저장하기에는 디스크의 처리 속도가 입력 스트림의 부하를 감당하지 못하는 문제가 있다.

따라서 이 논문에서는 센서로부터 수집된 데이터를 손실 없이 저장하는 동시에 시간에 따라 누적되는 센서 데이터의 이력을 효율적으로 관리할 수 있는 저장 기법을 제안한다. 제안된 기법은 센서의 측정값이 변경된 시점을 기준으로 동일한 값이 지속된 기간을 Time-segment와 Time-point로 저장한다. 이러한 방법은 센서의 모든 측정값을 정확하게 저장할 수 있고, 특정 시간에 걸쳐 중복되는 데이터들을 하나의 튜플에 저장하기 때문에 장기간 동안 수집된 데이터를 메모리에서 유지할 수 있다.

2. 관련 연구

DSMS(Data Stream Management System)에서의 데이터 스트림 처리 과정을 살펴보면, 먼저 사용자 또는 응용 시스템은 데이터 스트림으로부터 원하는 결과를 얻기 위하여 연속 질의를 시스템에 등록하고, 질의 프로세서는 연속적으로 입력되는 데이터 스트림에 대하여 등록된 질의의 수행 결과를 지속적으로 사용자 또는 응용 시스템에 반환한다. 이와 같은 처리를 위해서 DSMS는 최근에 입력된 데이터 스트림과 연속 질의 수행에 필요한 부가적인 정보들을 유지하기 위한 저장 공간을 필요로 한다[6].

데이터 스트림을 처리하기 위한 대표적인 프로젝트로써 AURORA[7]는 입력 스트림에 대한 한정된 영역에 대해 동작할 수 있는 윈도우 기반 연산자 집합을 제공하

고, QoS를 위한 온라인 스케줄링과 부하 분산 정책 등을 제안하였다. NiagaraCQ[8]는 분산 환경에서 XML 데이터 스트림에 대한 질의 언어인 XML-QL을 제안하고, 유사한 구조를 갖는 연속 질의를 그룹화 함으로써 연산 비용을 감소시킨다. STREAM[9]은 데이터 스트림에 대한 연속 질의를 정의할 수 있도록 표준 SQL을 확장한 CQL(Continuous Query Language)을 정의하고 있으며, 다중 데이터 스트림에 대한 비용 기반 질의 최적화 및 분산 데이터 스트림 처리 등의 방법을 제안하였다. TelegraphCQ[10]는 연속 데이터 스트림에 대한 질의 처리를 위하여 입력 데이터 스트림에 대한 라우팅과 연산자 스케줄링을 제공하며, 연산자간의 통신을 제공하고 있다. COUGAR[11]는 기본적으로 센서 노드들로 구성된 네트워크 환경을 데이터베이스라고 가정하고, 에너지 효율적인 데이터 분배 방법, 센서 네트워크에서의 In-Network 질의 처리, 질의 최적화 등의 방법들을 제안하였다.

이 논문에서 제안하는 시스템은 COUGAR와 같은 분산 네트워크 환경을 기반으로 센서 데이터 스트림의 시간 및 공간 특성을 고려하여 확장된 구조를 갖는다.

3. 센서 데이터 이력 관리 기법

3.1 Time-Segment Insertion 기법

센서로부터 수집되는 데이터 스트림은 <*S, V, T*>로 구성되며, 저장을 위한 테이블은 <*S, V, T_s, T_e*>로 구성된다고 가정한다. 여기서 *S*는 sensorID, *V*는 measuredValue, *T*는 Time-point, T_s와 T_e는 각각 Start-time과 End-time을 의미한다.

Time-Segment Insertion(TSI) 방법에서는 새로운 아이템이 도착할 때, 입력 스트림 저장하기 전에 이전의 측정값과 비교하여 측정값이 변경되었는지 확인한다. 만약, 측정값이 변경되었으면 변경된 시점을 T_e로 하는 Time-interval을 구성하여 이전의 튜플을 갱신하고 변경 시점을 T_s, T_e를 "*now*"로 하는 새로운 튜플을 삽입한다. 그리고 새로운 입력 스트림 값과 이전의 측정값이 같으면 입력 스트림은 삽입되지 않고 제거되며, 다음 입력 값이 도착할 때까지 Time-interval는 이전의 "*now*" 상태를 그대로 유지한다.

그림 1은 입력 스트림이 Time-interval로 저장되는 과정을 나타낸다. 그림 1(a)는 실시간으로 입력되는 데이터 스트림을 나타내고, 그림 1(b)는 그림 1(a)의 입력 스트림이 저장된 이후의 튜플을 나타낸다. 그림에서 보는바와 같이 처음에 t_1 시점에서 s_1, s_2, s_3이 입력되면, 각각의 아이템에서 T_s는 t_1, T_e는 *now*로 저장된다. t_2 시점에서 s_1은 이전 값인 25와 같은 값이 측정되었으므로 T_e는 "*now*"를 계속 유지한다. s_2는 측정값이 25에서 27로 변경되었기 때문에 t_1 시점의 튜플에서 T_e는 "*now*"에서 t_2로 갱신되고, 변경된 27을 저장하기 위해 t_2를 T_s로 하는 새로운 튜플이 추가된다. 마찬가지로 t_3 시점에서는 s_1의 튜플이 같은 방법으로 갱신된다. 최종적으로 그림 1(a)의 입력 스트림이 모두 처리되었을 때, 튜플은 그림 1(b)와 같은 형태로 저장된다. s_3의 경우는 최초의 입력부터 t_{now}까지 값의 변동이 없으므로 오직 하나의 튜플만이 생성되는 것을 볼 수 있다. 그림 1(c)는 전체 입력 스트림에 따라 T_s와 T_e의 변경이 진행되는 것을 Time-line으로 표현한 것이다. 이렇게 형성된 Time-segment를 통해 이력 질의를 효율적으로 수행할 수 있으며, 질의는 $T_s <= T < T_e$의 범위를 갖는다.

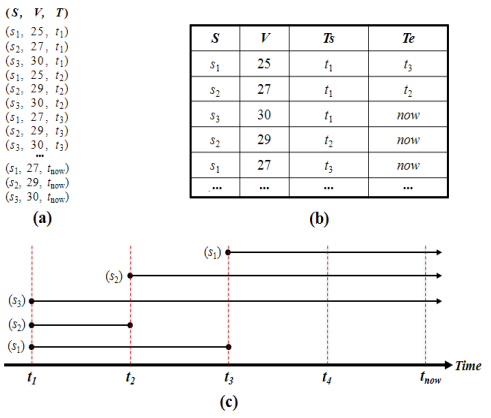

그림 1. TSI의 센서 데이터 처리 과정

그림 2는 TSI 알고리즘을 나타낸다. TSI는 입력 스트림의 측정값이 이전 값과 다를 경우에 이전 튜플에 대한 Time-interval을 갱신하여야 하기 때문에 갱신 연산으로 인한 부하가 추가적으로 발생하는 문제가 있다. 반면에 센서가 동일한 값을 센싱하는 빈도가 높은 경우에는 저장을 위한 연산이 발생하지 않기 때문에 갱신 연산으로 인한 부하는 어느 정도 상쇄될 수 있다. 또한 원시 데이터를 손실 없이 정확하게 저장할 수 있고, 가능한 한 많은 양의 입력 스트림을 하나의 튜플로 저장할 수 있는 장점이 있다.

Algorithm : Time-segment Insertion Method
Input : *k* input stream
Begin
if(new tuple *k* arrives from sensor) **then**
  n = find a tuple the same with *k.S* and T_e = *now*)
  if(*n* = *null*) **then**
    insert new tuple *k* with T_s = *k.T* and T_e = *now*
  else if(*n.V* ≠ *k.V*) **then**
    update tuple *n* with *n.*T_e = *k.T*
    insert new tuple with T_s = *k.T* and T_e = *now*
  end if
end if
End

그림 2. Time-Segment Insertion 알고리즘

3.2 Time-Point Insertion 기법

일반적으로 갱신 연산은 조건에 만족하는 튜플을 찾는 과정과 해당하는 튜플에 변경을 적용하는 단계로 수행되기 때문에 삽입 연산에 비해 처리 비용이 높다. TSI의 경우에 만약 입력 스트림 값이 동적으로 변화하는 빈도가 높을 경우에는 제거되는 튜플로 인해 감소되는 삽입 비용보다 추가로 발생되는 갱신 비용이 증가하는 문제가 발생한다. Time-Point Insertion(TPI) 기법에서는 입력 스트림을 Time-point로 저장하여 TSI에서 Time-interval을 갱신 하는 문제를 해결한다.

그림 3은 입력 스트림을 Time-point로 처리하는 과정을 나타낸다. TSI와 마찬가지로 TPI는 t_1 시점에서 s_1, s_2, s_3이 입력되면, 각각의 Time-point를 저장하게 되고, t_{now} 시점까지 새로운 입력 스트림에 대해 이전 값과 비교하여 측정값이 변경되었을 경우에만 데이터를 저장하고, 변경되지 않은 입력 스트림은 제거한다.

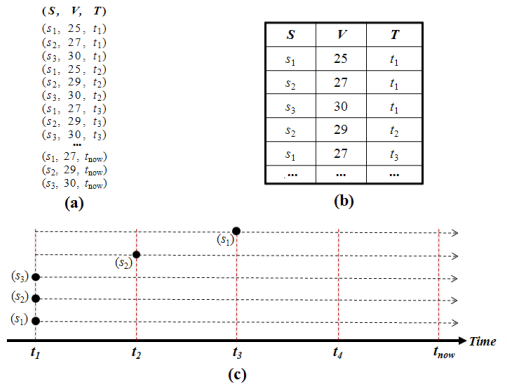

그림 3. TPI의 센서 데이터 처리 과정

그림 4는 TPI 기법의 자세한 알고리즘을 나타낸다. TPI 알고리즘에서 이전 값과 입력 스트림을 비교하는 과정은 TSI 알고리즘과 같다. 그러나 TPI는 변경된 입력 스트림을 Time-point와 함께 저장하는 과정만 필요하기 때문에 추가적인 갱신 연산은 발생하지 않으며, 이에 따라 입력 스트림에 대한 전체 저장 비용을 절감하는 장점이 있다.

Algorithm : Time-Point Insertion Method
Input : k input stream
Begin
if(new tuple k arrives from sensor) **then**
  n = find a tuple the same with $k.S$ and including maximum T
  if(n = *null* or $v.V \neq k.V$) **then**
    insert new tuple k
end if
End

그림 4. Time-Point Insertion 알고리즘

4. 평가 및 분석

실험에 사용된 데이터는 Intel Berkeley Research Lab[12]에서 제공된 데이터 셋을 사용하였고, 데이터 셋은 54개의 Mica2Dot 센서로부터 수집된 온도, 습도, 조도, 전압 데이터로 구성되어 있으며, 2004년 2월 28일부터 2004년 4월 5일까지 수집된 약 230만개의 데이터를 포함하고 있다. 성능 평가는 제안된 TSI, TPI, 그리고 시간 속성을 고려하지 않은 Non-Temporal Insertion (NTI)과의 실험을 통해 평가하였다.

그림 5는 입력 스트림의 증가에 따라 삽입된 튜플의 개수를 비교 평가한 것이다. 그림에서 알 수 있듯이 NTI는 모든 입력 스트림을 그대로 삽입하기 때문에 실제 저장되는 튜플과 입력스트림은 비례하여 증가한다. 반면에 TSI와 TPI는 입력 스트림이 이전의 입력 스트림과 같은 값을 갖는지 비교하여 제거하기 때문에 전체적으로 저장되는 튜플은 약 30% 정도의 감소율을 보이며, 저장되는 튜플의 개수는 항상 동일함을 알 수 있다.

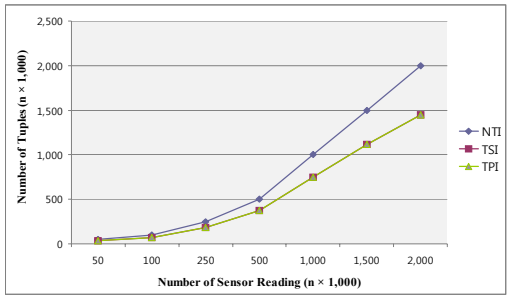

그림 5 삽입된 튜플의 개수

그림 6은 입력 스트림의 증가에 따른 평균 삽입 시간을 비교 평가한 것이다. 전체적으로 삽입 성능은 NTI가 가장 좋은 성능을 보인다. NTI는 입력 스트림에 대해 삽입 연산만 수행하기 때문에 추가적인 삽입 비용이 발생하지 않는다. 그러나 TSI와 TPI의 경우는 입력 스트림을 저장할 것인지 제거할 것인지를 결정하기 위하여 이전에 저장된 데이터와 비교를 수행해야 하므로 삽입 비용은 NTI에 비해 증가하게 된다.

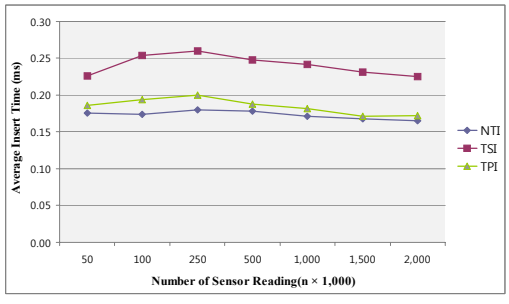

그림 6. 평균 삽입 시간

특히 TSI의 경우에는 삽입 연산과 함께 이전에 저장된 튜플의 Time-interval을 갱신해야 하기 때문에 NTI보다 약 25% 이상 낮은 성능을 보였다. 반면에 TPI는 입력 스트림을 Time-point로 저장하므로 별도의 갱신 연산이 발생하지 않고, 삽입을 결정하기 위한 계산 과정만 요구되므로 NTI와의 성능 차이는 크지 않다.

그림 7은 전체 입력 스트림에 대해 과거 1시간 동안의 시간 질의를 수행하고, 이에 대한 평균 질의 수행 시간을 비교한 것이다. 그림에서 알 수 있듯이 TPI가 가장 좋은 성능을 나타낸다. TPI의 경우, 질의 조건의 Time-interval에 포함된 Time-point만을 비교하므로 복잡한 연산이 필요하지 않고, NTI보다 적은 튜플을 대상으로 질의를 수행하기 때문에 연산 횟수가 감소되는 장점이 있다. TSI의 경우는 TPI에 비해 낮은 성능을 보이는데, TSI는 입력 스트림을 Time-segment로 저장하기 때문에, 범위 질의 수행 시 주어진 Time-interval에 대해 Time-segment의 시작 시점과 종료 시점을 모두 비교하여야 하고, 이로 인해 질의 연산이 복잡해지는 단점이 있다. NTI는 가장 낮은 성능을 보이는데, 많은 대상 튜플로 인한 질의 연산 횟수의 증가가 원인임을 알 수 있다.

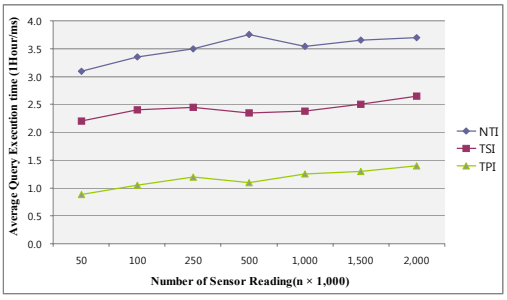

그림 7. 평균 시간 질의 실행 시간 (1시간)

그림 8은 전체 입력 스트림에 대해 과거 24시간 동안의 시간 질의를 수행하고, 평균 질의 수행 시간을 비교한 것이다. 그림 7과 마찬가지로 TPI가 가장 좋은 성능을 나타내고, NTI가 가장 낮은 성능을 나타낸다.

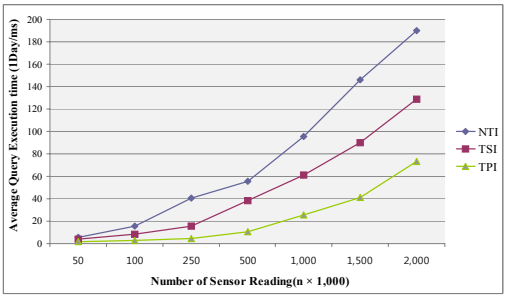

그림 8. 평균 시간 질의 실행 시간 (1일)

그림 8에서 보는바와 같이 전체적으로 평균 질의 수행 시간은 선형적으로 증가하고 있음을 알 수 있다. 질의 범위가 큰 이력 질의의 경우에는 메모리에서 만료된 데이터를 검색하기 위하여 디스크를 접근해야 하기 때문에 질의 처리 비용은 급격히 증가하게 된다. 또한 입력 스트림이 증가할수록 디스크에 저장되는 데이터도 증가하게 되고, 이에 따라 질의 수행을 위한 디스크 접근 횟수도 증가하게 된다.

5. 결론

이 논문에서는 센서 데이터 스트림을 손실없이 저장하고, 효율적인 이력 관리를 위해 시간 기반 센서 데이터 저장 기법을 제안하였다. 제안된 기법에서는 입력 스트림을 측정값이 변경된 시점을 기준으로 동일한 값이 지속된 기간을 Time-segment와 Time-point로 저장하였고, 이를 통해 일정 시간에 걸쳐 중복되는 값을 하나의 튜플에 저장함과 동시에 원시 데이터를 정확하게 저장할 수 있었다. 다양한 실험을 통해 제안된 시스템의 성능을 평가한 결과, 데이터 저장 및 질의 처리 비용이 크게 감소하였으며, 약 30% 이상의 저장 공간 감소율을 보였다.

참 고 문 헌

[1] Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A. S., Ryvkina, E., Stonebraker, M., Tibbets, R., “Linear Road: A Stream Data Management Benchmark,: In Proc. 30th conf. VLDB, 2004, pp. 480-491.

[2] Lee, Y. K., Jung, Y. J., Ryu, K. H., “Design and Implementation of a System for Environmental Monitoring Sensor Network,” In Proc. Conf. APWeb/WAIM Workshop on DataBase Management and Application over Networks, 2007, pp. 223-228.

[3] Nehme, R. V., Rundensteiner, E. A., “Cluster-Sheddy : Load Shedding Using Moving Clusters over Spatio-temporal Data Streams,” In Proceeding of DASFAA, 2007, pp.637-651.

[4] 이양구, 김원태, 정영진, 김광득, 류근호, “날씨 마케팅 적용을 위한 기후 데이터의 군집 분석,” 한국공간정보시스템학회 논문지, 제7권 제3호, 2005, pp. 33-44.

[5] 김은희, 지정희, 손호선, 류근호, 이충호, “클러스터링 기법을 이용한 산불 데이터 상관관계 분석,” 한국공간정보시스템학회 추계학술대회, 2005, pp. 81-86.

[6] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R., “Query Processing, Resource Management, and Approximation in a Data Stream Management System,” In Proc. of CIDR, 2003, pp. 245-256.

[7] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S. B., "AURORA: A New Model and Architecture for Data Stream Management," VLDB Journal, Vol.12 No.2, 2003, pp. 120-139.

[8] Chen, J., DeWitt, D. J., Tian, F., Wang, Y., "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," SIGMOD Conference, 2000, pp. 379-390.

[9] Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J., "STREAM: The Stanford Stream Data Manager," IEEE Data Eng. Bull., Vol.26 No.1, 2003, pp. 19-26.

[10] Reiss, F., Hellerstein, J. M., "Data Triage: An Adaptive Architecture for Load Shedding in TelegraphCQ," ICDE, 2005, pp. 155-156.

[11] Yao, Y., Gehrke, J., "The COUGAR Approach to In-network Query Processing in Sensor Networks," in SIGMOD Record, Vol.31 No.3, 2002, pp. 9-18.

[12] http://berkely.intel-research.net/labdata/