논문 2009-46CI-3-4

유비쿼터스 센서 네트워크에서 스트림 데이터를 효율적으로 관리하는 저장 관리자 구현

(Implementation of Storage Manager to Maintain Efficiently Stream Data in Ubiquitous Sensor Networks)

이 수 안*, 김 진 호**, 신 성 현***, 남 시 병****

(Suan Lee, Jinho Kim, Sung-Hyun Shin, and Si-Byung Nam)

요 약

유비쿼터스 센서 네트워크를 통해 수집되는 데이터는 끊임없이 변화하는 스트림 데이터이다. 이 스트림 데이터는 기존의 데이터베이스와는 매우 다른 특성을 가지고 있어서, 이를 저장하고 분석 및 질의 처리하는 방법에 대한 새로운 기법이 필요하며, 이에 대한 연구가 최근에 많은 관심을 끌고 있다. 본 연구에서는 센서 네트워크로부터 끊임없이 들어오는 스트림 데이터를 수집하고 이를 효율적으로 데이터베이스에 저장하는 저장 관리자를 구현하였다. 이 저장 관리자는 무선 센서 환경에서 발생하는 오류에 대한 정제, 반복적으로 센싱되는 동일한 데이터에 대한 축소 기능, 장기간의 스트림 데이터를 경동 시간 구조로 유지하는 기능 등을 제공한다. 또 이 연구에서는, 구현된 저장 관리자를 건물의 온도, 습도, 조도 등을 수집하는 건물 화재 감시 센서 네트워크에 적용하여 그 성능을 측정하였다. 실험 결과, 이 저장 관리자는 스트림 데이터의 저장 공간을 현저히 줄이며, 건물 화재 감시를 위한 장기간의 스트림 데이터를 저장하는데 효과적임을 보였다.

Abstract

Stream data, gathered from ubiquitous sensor networks, change continuously over time. Because they have quite different characteristics from traditional databases, we need new techniques for storing and querying/analyzing these stream data, which are research issues recently emerging . In this research, we implemented a storage manager gathering stream data and storing them into databases, which are sampled continuously from sensor networks. The storage manager cleans faulty data occurred in mobile sensors and it also reduces the size of stream data by merging repeatedly-sampled values into one and by employing the tilted time frame which stores stream data with several different sampling rates. In this research, furthermore, we measured the performance of the storage manager in the context of a sensor network monitoring fires of a building. The experimental results reveal that the storage manager reduces significantly the size of storage spaces and it is effective to manage the data stream for real applications monitoring buildings and their fires.

Keywords: 유비쿼터스 센서 네트워크, 스트림 데이터, 저장 관리자, 경동 시간 구조, 데이터 축소

I. 서 론

무선 네트워크와 센서 기술의 발달로 유비쿼터스 센서 네트워크(Ubiquitous Sensor Network, USN)이 실용화되고 있다. 이 USN의 각 노드에 연결된 센서들을 통해 온도, 조도, 습도, 압력 등과 같은 주변 환경 정보를 실시간으로 수집하고, 이러한 정보를 토대로 환경

* 학생회원, ** 정회원, 강원대학교 IT특성화대학 컴퓨터과학과
(Dept. of Computer Science, Kangwon University)
*** 정회원, 한양대학교 BK21 사업단
(BK21, Hanyang University)
**** 정회원, 강원대학교 공학대학 전자공학과
(Dept. of Electornic Eng., Kangwon University)
※ 본 논문은 2007년도 강원대학교 캠퍼스간 공동연구비로 연구하였음.
접수일자: 2009년3월27일, 수정완료일: 2009년5월4일

감시, 재난 관리, 건강 관리, 물류 시스템 등의 다양한 유비쿼터스 응용들이 개발되고 있다[1~3].

센서 네트워크를 통해 수집되는 데이터는 연속적으로 끊임없이 발생하는 스트림 데이터이다. 이 스트림 데이터는 그 양이 방대하여 이를 저장하기가 매우 어려우며, 동시에 이를 검색하는데 많은 시간이 소요된다. 이렇게 전통적인 데이터베이스와는 매우 다른 성격을 갖는 스트림 데이터에 대한 연구가 많이 진행되고 있다[2~4].

스트림 데이터에 대한 연구는 주로 이를 직접 저장하는 것보다, (스트림 데이터를 한번 읽으면서) 이에 대한 특성 분석, 이상/특이 패턴 분석, 분류/클러스터링 등에 대한 연구가 많이 진행돼 왔다[2~3]. 전통적인 임의의 질의가 아니라 일정 주기로 반복적으로 실행하는 연속 질의를 스트림 데이터에 대해 효율적으로 처리하는 방법에 대한 연구들이 진행되었다[1, 5~7].

스트림 데이터가 무한히 생성되지만 이를 요약하여 저장하고 관리하는 연구들도 여러 개 수행되었는데, Stream[1], NiagaraCQ[5], TelegraphCQ[6], Aurota[7] 등이 그 예이다. 또한 최근에는 스트림 데이터에 대한 효율적인 다차원 분석을 위해 데이터 웨어하우스를 구축하고, OLAP 분석을 효율적으로 수행하는 방법에 대한 연구들이 진행되었다[4, 8~10]. 이 연구들은 스트림 데이터에 대한 다차원 큐브를 제한된 기억장소에 저장할 수 있도록, 시간 차원에 대한 경동 시간 구조(Tilted Time Frame) 및 제한된 관심 영역에 대한 큐보이드(Cuboid)만 저장하는 등의 방법을 제안하였다.

본 연구에서는 이러한 연구 결과들을 응용하여 유비쿼터스 센서 네트워크의 스트림 데이터를 수집하고, 이를 데이터베이스에 효율적으로 저장하는 저장 관리자를 구현하였다. 이 저장 관리자는 무선 센서 환경에서 발생할 수 있는 오류에 대한 정제 기능, 반복적으로 센싱된 동일한 데이터에 대한 압축 기능, 장기간에 수집된 스트림 데이터를 서로 다른 시간 간격(즉, 경동 시간 구조)으로 유지하는 기능을 제공한다. 이런 기능을 통해 사용자들이 응용의 목적에 맞도록 스트림 데이터를 압축/요약하고 저장할 수 있도록 한다. 본 연구에서는 이 저장 관리자를 건물의 온도, 습도, 조도 등을 수집하는 건물 화재 감시 센서 네트워크에 적용해 본 결과, 원천 스트림 데이터를 손실 없이 수백분의 일로 줄일 수 있으며, 경동 시간 구조를 사용할 경우 한 센서 당 수 MB 단위 내에서 전체 스트림 데이터를 저장할 수 있었다. 이 저장 관리자에 의해 압축/요약 저장된 스트림 데이터는 스트림 질의 처리, 데이터 마이닝 분석, 데이터 웨어하우스 구축 등의 원천 데이터로 활용될 수 있을 것이다.

Ⅱ. 관련 연구

센서 네트워크에서는 각각의 센서 노드들로부터 끊임없이 수집되는 스트림 데이터가 생성된다. 이 스트림 데이터를 제한된 크기의 데이터베이스에 모두 저장하는 것은 불가능할 뿐만 아니라, 이에 대한 질의를 처리하는 데는 많은 시간이 소요된다. 이러한 스트림 데이터를 효과적으로 관리하는 방법이 많이 연구되었다.

Chen 등[5]은 XML 스트림 데이터에서 연속적인 질의(continuous query)를 처리하기 위한 목적으로 NiagaraCQ를 개발하였다. NiagaraCQ는 많은 질의들이 유사한 구조를 포함하고 있는 사실을 기반으로 이들 질의를 점진적으로 그룹화하여 적은 연산만으로 많은 질의를 처리하였다. Chandrasekaran 등[6]은 대용량의 스트림 데이터를 다루는 질의어 StreaQuel을 개발하고, 연속 질의를 지원하는 TelegraphCQ를 개발하였다. 또한 Motwani 등[1]의 STREAM(STanford stREam datA Manager)과 Abadi 등[7]의 Aurora에서도 스트림 데이터를 관리하고 이에 대한 연속적인 질의를 처리하는 방법을 개발하였다. 이들 연구들은 주로 스트림 데이터에서 질의의 효율 향상이나 스트림 처리 시 발생하는 성능 개선에 초점을 두었을 뿐, 효과적으로 스트림 데이터 저장에 관한 문제 해결은 미흡하다.

다음으로, 스트림 데이터에 대한 OLAP 분석을 위해 스트림 데이터에 대한 다차원 큐브를 생성하는 방법에 대해 최근에 연구되었다. Han 등[8]은 시간 차원에 대한 경동 시간 구조와 각 차원에 대한 관심 계층(m-layer)와 관찰 계층(o-layer) 개념을 이용하여 관심 영역의 큐보이드만 저장하는 StreamCube를 제안하였다. Gonzalez 등[4]은 RFID(Radio Frequency Identification)의 데이터 집합에 대해 우선 순위에 따FMS 데이터를 압축(compression)과 경로 의존적인 집계(path-dependent aggregate)를 수행하는 방법을 개발하였다. 도기석 등[9]과 서대홍 등[10]은 StreamCube의 평균 응답 시간을 줄이고, 차원 속성 값들을 동적으로 그룹화하는 방식을 연구하였다. 이를 통해 스트림 큐브의 저장 공간과 질의 처리 시간을 감소시켰다.

Ⅲ. 스트림 데이터 저장 관리자

스트림 데이터를 저장하고 이에 대한 질의를 처리하는 스트림 데이터 관리 시스템의 일반적인 아키텍처는 아래 그림 1과 같은 구조로 이루어져 있다.

이 그림에서 보는 바와 같이, 스트림 데이터 관리 시스템은 스트림 데이터 수집기, 스트림 스케쥴러, 스트림 카탈로그 관리자, 스트림 전처리기, 스트림 저장 관리자, 질의 처리기, 스트림 데이터 모니터링 등의 컴포넌트로 구성된다. 이 중에서 스트림 저장 관리자는 스트림 데이터를 가공하여 데이터베이스에 저장하는 역할을 담당한다. 이 연구에서는 이들 컴포넌트들 중에서 스트림 저장 관리자를 구현하였다.

이 논문에서 구현한 스트림 저장 관리자는 아래 그림 2와 같은 구조를 가지고 있으며, 핵심 기능을 살펴보면 센서로부터 스트림 데이터를 수집하는 기능을 거쳐 스트림 데이터를 전처리 과정이 있으며, 데이터 축소 형태로 효율적으로 저장 장치에 저장하는 기능을 갖는다.

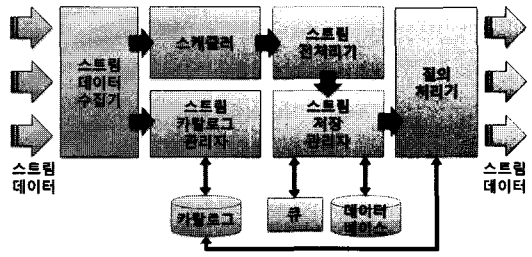


그림 1. 스트림 데이터 관리 시스템 구조
Fig. 1. Stream data management system architecture.

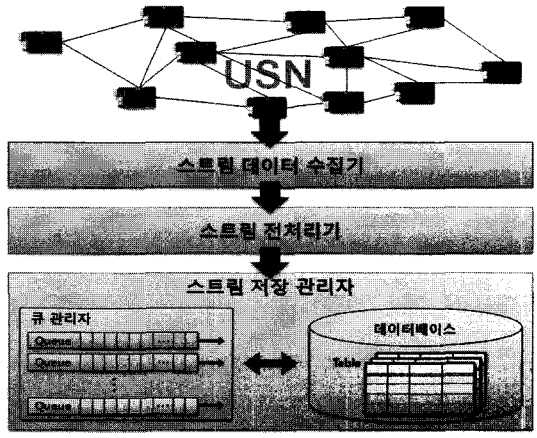


그림 2. 스트림 저장 관리자 구조
Fig. 2. Stream storage manager architecture.

1. 데이터 정제 (Data Cleaning)

일반적으로 데이터는 측정 장치의 한계나 오류, 데이터 수집 과정 결함 등으로 인해 데이터가 누락되거나 잘못된 데이터를 나타내는 현상을 보인다. 데이터 정제는 이러한 일관성이 없고 부정확한 데이터들을 삭제하거나 수정을 통해서 데이터 품질 문제를 해결하는 과정이다[11].

1.1 누락 값 및 비정상적인 값 처리

센서 스트림 데이터를 살펴보면 네트워크나 측정 장치의 문제로 인해 누락된 값들이 존재하거나 특정 센서에서 발생할 수 있는 값의 범위를 벗어나는 비정상적인 값이 발생한다.

본 논문에서는 그림 3과 같이 누락 값이나 비정상적인 값이 발생할 경우 바로 이전 값으로 대체하는 형태로 처리하였다. 이는 정확성은 고려하면서 누락 값과 비정상적인 값은 데이터 축소 방법에 의해 축소되도록 하기 위함이다.

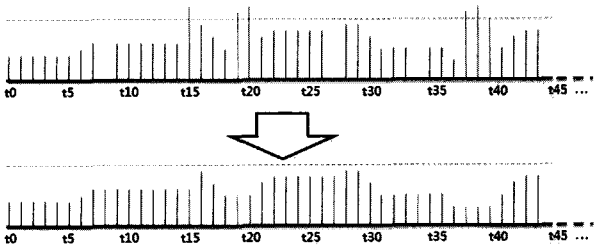


그림 3. 스트림의 누락 값 및 비정상적인 값 처리 예
Fig. 3. Example of handling missing values and faulty values of stream data.

1.2 잡음(Noise) 처리

잡음은 측정 오류의 한 형태로서 값의 왜곡이나 정확하지 않은 값들을 의미한다[11]. 센서 네트워크에서도 센서의 측정 범위를 벗어나는 비정상적인 값을 제외한 나머지 값들 가운데 잡음이 발생한다.

본 논문에서는 그림 4과 같이 이러한 잡음을 식(1)에 해당하는 Smooth 기법을 이용하여 감소시키는 방법을 이용하였다.

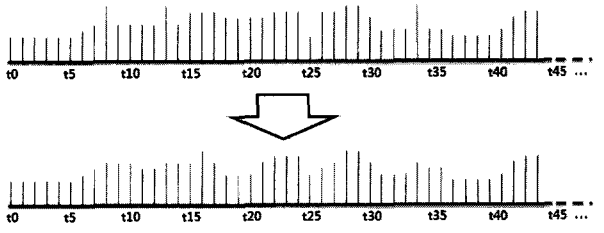


그림 4. 스트림의 잡음 처리 예
Fig. 4. Example of processing noise in stream data.

$$smooth = \frac{T^{now} + \left(\sum_{i}^{n} T^{i}/n\right) \times N - 1}{N} \quad (1)$$

T^{now}는 스트림 데이터의 현재 값을 의미하고, N은 사용자가 지정한 Smooth의 프레임 크기를 의미한다.

2. 데이터 축소 (Data Reduction)

센서 네트워크와 같은 연속적인 스트림 데이터에서는 대용량 데이터의 저장 비용과 데이터의 효율적인 분석을 목적으로 데이터 축소가 필요하다. 데이터 축소를 통해서 적은 데이터 공간과 처리시간을 사용하며, 효율적인 온라인 분석 처리(OLAP)이나 데이터 마이닝 알고리즘을 사용 할 수 있다.

2.1 구간(Interval) 저장 방식을 이용한 데이터 축소

센서 스트림 데이터에서 수집된 센서 값을 살펴보면 잦은 변화나 급격한 변화를 보이지 않는 특성을 보인다. 즉, 표 1과 같이 센서의 소스 스트림 데이터에서 데이터 값 v 가 어느 구간 t 에 대해서 동일 값을 나타내는 경우가 많다.

구간 t 에 대해서 동일 값을 나타내는 경우 그 구간에 대한 시간 간격(time interval)를 이용하여 축소가 가능하다. 예를 들어 총 5개의 mote에서 각각 100개의 스트림을 저장할 경우 500개의 레코드가 필요하지만, 표 2와 같이 구간 저장 방식을 이용할 경우 33개의 레코드로 저장이 가능하다.

표 1. 센서의 소스 스트림 데이터 예
Table 1. Example for source stream data of sensors.

mote id	no	value	time
m1	*n1*	*v*	*t1*
m2	*n1*	*v*	*t1*
m3	*n1*	*v*	*t1*
m4	*n1*	*v*	*t1*
m5	*n1*	*v*	*t1*
...	...	...	...
m1	*n8*	*v*	*t8*
m2	*n8*	*v*	*t8*
m3	*n8*	*v*	*t8*
...	...	...	...
m1	*n20*	*v*	*t20*
m2	*n20*	*v*	*t20*
...	...	...	...
m1	*n38*	*v*	*t38*
m2	*n38*	*v*	*t38*
...	...	...	...
m1	*n100*	*v*	*t100*
m2	*n100*	*v*	*t100*

표 2. 구간 저장 방식의 스트림 데이터 예
Table 2. Example of stream data in interval-based storage.

mote id	no	value	time in	time out
m1	*n1*	*v*	*t1*	*t8*
m2	*n1*	*v*	*t1*	*t20*
m3	*n1*	*v*	*t1*	*t100*
m4	*n1*	*v*	*t1*	*t100*
m5	*n1*	*v*	*t1*	*t20*
...	...	...	...	...
m1	*n13*	*v*	*t20*	*t100*
...	...	...	...	...
m2	*n19*	*v*	*t38*	*t100*
m5	*n19*	*v*	*t38*	*t100*

이와 같은 구간 저장 방식은 센서의 종류, 측정 범위, 감도, 주변 환경과 같은 특성에 영향을 받는다. 예를 들어 온도, 습도와 같이 긴 시간동안 값의 변화가 적은 경우 이 기법을 이용하여 효율적인 축소가 가능하다. 하지만 적외선이나 조도와 같이 측정 범위도 크고, 민감한 센서의 경우 이 기법의 효과가 적을 수 있다. 본 논문에서는 식(2)와 같이 임계값(threshold)을 이용하여 데이터 축소의 효율성을 증대시키는 방법을 제시한다.

$$interval = (T^{now} - T^{1} = T^{now} \pm threshold) \quad (2)$$

T^{now}는 스트림 데이터의 현재 값을 의미하고, threshold는 사용자로부터 입력 받은 임계값에 해당된다. 사용자가 임계값을 지정하면 해당 범위의 값들 또한 구간 저장 방식에 포함시키기 때문에 좀 더 축소율 향상에 도움이 된다.

2.2 경동 시간 구조(Tilted Time Frame)를 이용한 데이터 축소

경동 시간 구조는 데이터 분석시의 효율성을 고려하여 축소하는 방법이다. 사용자들은 최근 데이터 정보에 더 관심을 보이기 때문에 최근 데이터를 세밀히 유지하고, 어느 일정한 관심 영역 이후에는 프레임 간격을 넓혀서 저장한다[8].

예를 들어 그림 5(1)와 같이 센서의 수집 시간을 3초로 지정하고 경동 시간 구조를 이용하면 3초간 20번의 수집으로 60초 간격 프레임을 값으로 채우고, 60분 간격의 프레임은 60초 프레임을 60번 수집하여 값을 채우고, 24시간 간격의 프레임은 60분 프레임을 24번 수집하여 값을 채운다. 마찬가지로 1주 간격의 프레임은 24

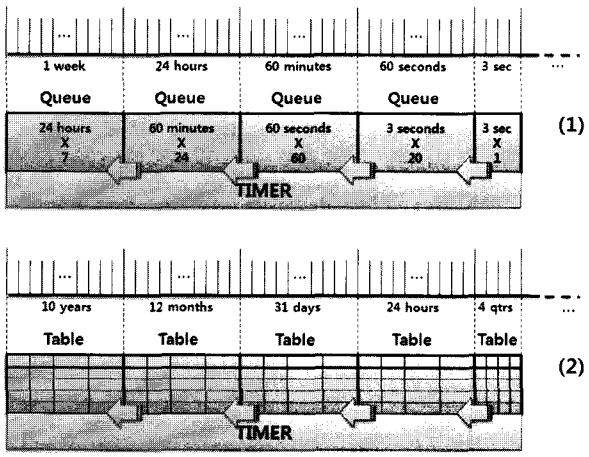


그림 5. 경동 시간 구조를 이용한 데이터 축소 예
Fig. 5. Example of data reduction using tilted time frame mechanism.

시간 프레임을 7번 수집하여 값을 채우게 된다.

아래 식 (3)과 (4)는 스트림 데이터를 일반적으로 저장한 경우(Normal)와 경동 시간 구조(TTF)를 이용했을 때의 저장 공간의 크기를 각각 나타낸다. 이때 경동 시간 구조는 T_1 기간 동안 t_1의 주기로, T_2 기간 동안 t_2의 주기로 등과 같이 저장한다고 가정한다.

$$normal = \frac{T_1 \times T_2 \times T_3 \times T_4 \times \cdots \times T_n}{t} \quad (3)$$

$$TTF = \frac{T_1}{t_1} + \frac{T_2}{t_2} + \frac{T_3}{t_3} + \frac{T_4}{t_4} + \cdots + \frac{T_n}{t_n} \quad (4)$$

예를 들어, 센서 수집 주기를 3초로 설정하고, 1주간의 데이터를 저장하면 식(3)과 같이 총 7 × 24 × 60 × 20 × 1 = 201,600 만큼의 데이터를 저장해야 한다. 하지만 경동 시간 구조를 이용할 경우 식(4)와 같이 총 7 + 24 + 60 + 20 + 1 = 112 만큼의 데이터만 저장하면 되므로 큰 차이로 센서 스트림 데이터의 양을 축소할 수 있다.

그림 5(2)는 데이터베이스의 테이블을 이용하여 경동 시간 구조로 데이터를 저장한 경우를 보여주며, 마찬가지로 각 테이블이 상위 경동 시간 구조의 테이블로 값을 전달할 때 사용자가 지정한 함수(CURRENT, COUNT, MIN, MAX, SUM, AVG) 값 형태로 전달된다.

3. 데이터베이스 스키마 구조

스트림 저장 관리자에서 센서 스트림 데이터를 데이터베이스에 저장하기 위한 스키마 구조는 그림 6과 같다. Location 엔터티는 위치 정보를 담고 있으며, Motes

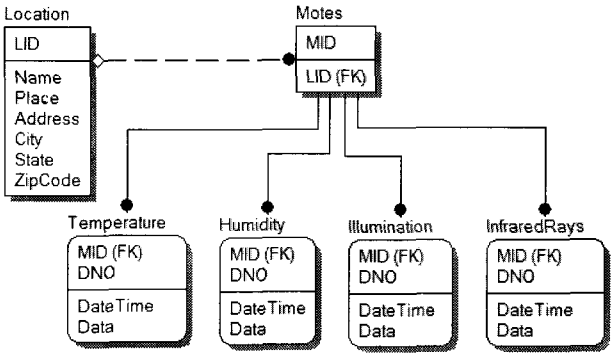


그림 6. 센서 스트림 데이터를 위한 스키마
Fig. 6. Schema for sensor stream data.

엔터티는 4개의 센서 엔터티를 저장하는 Temperature, Humidity, Illumination, InfraredRays 엔터티들과 1:N 관계로 형성된다.

센서 엔터티들은 Identity 형태의 DNO 애트리뷰트를 가지고 있고, 시공간(spatiotemporal)에 대한 질의를 처리하기 위해 Location 엔터티와 함께 각 센서들은 DateTime 애트리뷰트를 가지고 있다. 센서 장비에 센서가 추가 될 경우 센서 엔터티가 추가 되며, 각 센서 엔터티는 동일한 구조의 속성을 가지고 있다.

Ⅳ. 구 현

본 장에서는 센서 스트림 데이터의 저장 관리자의 구현 내용에 대해 기술한다. 센서 장치와 프로그램의 아키텍처, 각 구조 별 클래스 및 자료구조를 기술한다.

1. 센서 장치

센서 스트림 데이터의 수집과 분석을 위한 장치는 그림 7과 같은 한백 전자의 ZigbeX Mote를 사용하였다. ATmega128L CPU를 채택하고 있으며, 센서 운영체제로는 TinyOS를 사용한다. RF Chip은 Chipcon의 CC2420을 사용하며, 기본적으로 온도, 습도, 조도, 적외선 센서를 장착하고 있다. 센서 네트워크로는 Zigbee

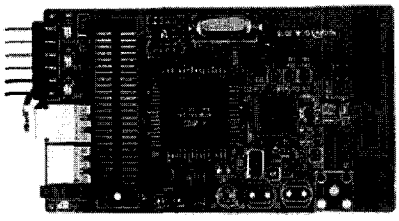


그림 7. 한백 전자 ZigbeX Mote
Fig. 7. SigbeX Mote of Hanback Electronic.

통신을 이용하며, USB 인터페이스를 통해 시리얼 통신하여 센서 데이터 스트림을 수집한다.

1.1 센서 데이터 프레임 구조

센서에서 센서 네트워크를 통해 통신하는 데이터 프레임 구조는 표 3과 같다. 센서 데이터 프레임 수집기는 각 센서 데이터 프레임 구조안의 시리얼 통신을 위한 주소와 패킷 타입, 센서들의 값 등을 스트림 형태로 수집하게 된다.

표 3. 센서 데이터 프레임 구조
Table 3. Frame structure for sensor data.

type	name	describe
uint8_t	add[2]	serial communication address
uint8_t	type	packet type
uint8_t	group	group id
uint8_t	length	data length
uint16_t	src	sending mote address
uint16_t	dst1	last destination address
uint16_t	dst2	multi-hop middle address
uint16_t	dst3	multi-hop middle address
uint32_t	seq	packet no
uint16_t	Temp	temporary value
uint16_t	Humi	humidity value
uint16_t	Photo	photo value
uint16_t	Ultrared	ultrared value
uint32_t	Totalduration	total duration time
uint32_t	TXduration	TX duration time
uint32_t	Sleepduration	sleep duration time
uint16_t	CRC	check sum

2. 저장 관리자 전체 구조

센서 관리 프로그램은 효율적인 수집 및 분석을 위해 서 그림 8과 같은 아키텍처로 구성되었고, .NET Framework 위에서 C# 언어를 이용해 개발하였다.

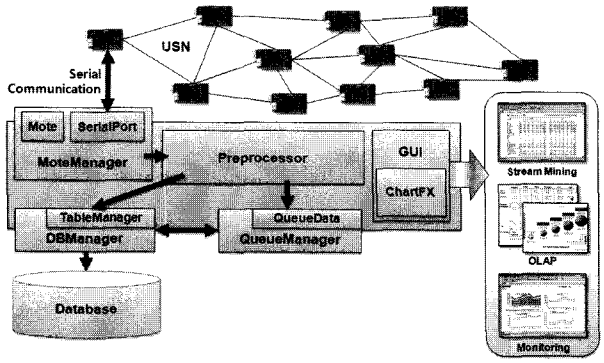


그림 8. 스트림 데이터 저장 관리자 구조
Fig. 8. Storage manager architecture for stream data.

2.1 센서 수집 관련 구조

수집된 스트림 데이터는 그림 9와 같이 각 Sensor 클래스를 가지고 있는 Mote 클래스 형태로 저장된다. MoteManager는 Hashtable 구조를 이용하여 Mote 클래스의 빠른 접근 및 관리가 가능하고, 동적 프로그래밍을 통해 Mote가 프로그램 동작 중에 켜져도 수집이 원활하게 이루어진다.

MoteManager에서 SerialPort는 실제 센서 스트림 데

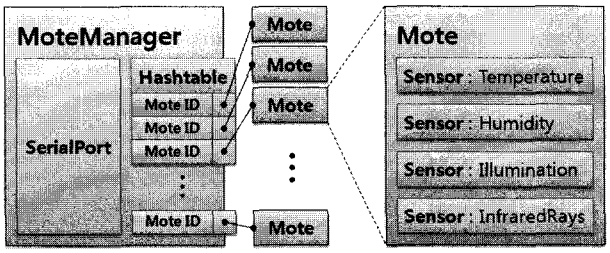


그림 9. Sensor, Mote, MoteManager 클래스 관계
Fig. 9. Relationship of Sensor, Mote and MoteManager classes.

표 4. 센서 스트림 데이터 수집 소스 코드
Table 4. Source code for sensor stream data acquisition.

```
while (true)
{
        for (int i = 0; i < _DataStream.Length-1; i++)
        {
            _DataStream[i] = (int)_MoteConnect.ReadByte();
            if (_DataStream[i] == 0x7e) i = 0;
        }

        if (!_HashMote.ContainsKey(_DataStream[8]))
        {
            mote = new Mote(_DataStream[8]);
            _HashMote.Add(_DataStream[8], mote);
        }
        else
            mote = (Mote)_HashMote[_DataStream[8]];

        MoteID = (_DataStream[9] << 8) + _DataStream[8];
        Temperature = (_DataStream[21] << 8) +
_DataStream[20];
        Humidity = (_DataStream[23] << 8) + _DataStream[22];
        Illumination = (_DataStream[25] << 8) +
_DataStream[24];
        InfraredRays = _DataStream[26];
}
```

이터를 시리얼 통신을 통해 수집하기 위해 사용하는 C# 에서 제공된 클래스이다. Thread를 이용해 시리얼 통신으로 받은 스트림 데이터를 각 Mote 클래스의 Sensor 클래스 자료구조에 저장한다. 표 4는 스트림 데이터 수집 부분에 대한 소스 코드이다.

2.2 센서 스트림 데이터 저장(Queue) 관련 구조

컴퓨터 메모리에 센서 스트림 데이터를 저장할 경우 그림 10과 같이 QueueManager 클래스의 각 센서별 Queue 자료구조를 이용하여 QueueData 클래스가 저장된다. 만약 경동 시간 구조를 이용하여 저장할 경우 여러 개의 QueueManager가 동적으로 생성되어 각 프레임에 맞게 저장되는 구조로 되어있다.

QueueData와 QueueManager의 자료 구조 관련 코드는 표 5와 같으며, 센서에 필요한 정보만 Queue 형태로 메모리에 저장하게 된다. 사용자로부터 옵션을 통해 일반적인 데이터 수집, 구간 저장 방식, 경동 시간 구조

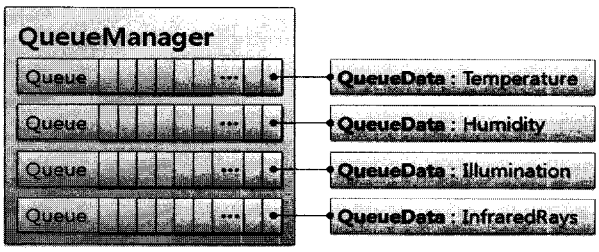


그림 10. QueueData, QueueManager 클래스 관계
Fig. 10. Relationships of QueueData and QueueManager classes.

표 5. 상위 경동 시간 구조의 튜플 삽입 질의
Table 5. Query inserting tuples into higher level of tilted time frame.

```
INSERT INTO frameTable
SELECT MID, MAX(DateTime), AVG(Data)
FROM sensorTable
GROUP BY MID
```

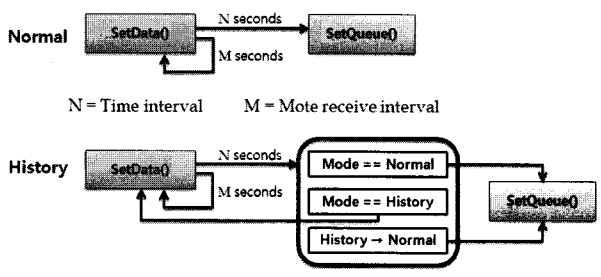


그림 11. 구간 저장 방식에 의한 메커니즘
Fig. 11. Mechanism for interval-based storage.

방식을 선택하면 그에 알맞게 QueueManager 클래스를 사용할 수 있다. 그림 11은 구간 저장 방식을 이용할 경우 동작하는 메커니즘이다. 모드의 식별을 통해 이전 값과 같을 경우 Queue에 저장하지 않고, 타임스탬프만 증가 시키고, 이전 값과 같지 않을 경우 Queue에 저장하는 절차로 되어있다.

2.3 센서 스트림 데이터 저장 관련 구조

데이터베이스에 센서 스트림 데이터를 저장할 경우 그림 12과 같이 TableManager 클래스의 각 센서별 SensorTable 클래스를 통해 센서 스트림 데이터를 저장한다.

사용자의 옵션에 따라서 DBManager 클래스는 실시간으로 데이터베이스에 저장하거나, 버퍼 형태로 유지하다가 일괄적으로 데이터베이스 저장한다. 데이터베이스에 저장할 경우도 마찬가지로 경동 시간 구조를 이용하여 저장할 경우 여러 개의 TableManager가 동적으로 생성되어 각 프레임에 맞게 저장되는 구조로 되어있다. 예를 들어 상위 경동 시간 구조의 테이블로 평균값에 대한 튜플을 삽입할 경우 표 5와 같은 질의를 이용한다.

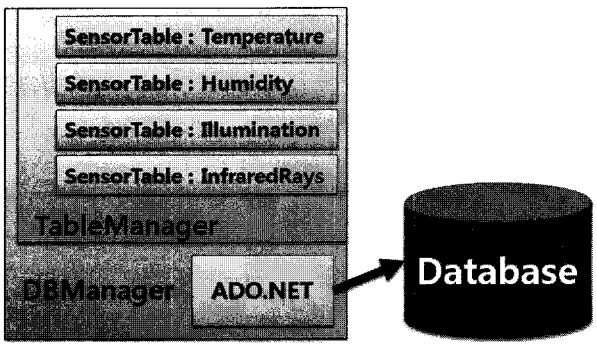


그림 12. DBManager, TableManager, SensorTable 클래스 관계
Fig. 12. Relationships of DBManager, TableManager, and SensorTable classes.

V. 실 험

이 장에서는 구현된 스트림 데이터 저장 관리자의 성능을 분석하고 실제 응용에 적용 가능한지 검증한다. 이를 위해 건물 화재 감시를 위한 센서 네트워크를 테스트베드로 구축하였으며, 여기서 수집된 온도, 습도, 조도, 자외선 등을 수집한 스트림 데이터를 저장하는데 적용하였으며 그 성능을 분석하였다.

1. 데이터 정제(Data Cleaning)

구현된 저장 관리자의 데이터 정제에 대한 효율성을 검증하기 위해 각 센서별로 약 97,000개의 데이터를 수집하였으며, 이때 발생하는 누락 또는 비정상적인 값을 개수를 측정하였다. 각 센서별로 스트림 데이터에서 정상치와 누락 또는 비 정상치의 발생 비율이 아래 표 6과 같이 나타났다.

센서 별로 차이를 보이지만, 약 0.5% 정도의 누락 값과 비 정상치가 나타나는 것을 볼 수 있다. 저장 관리자는 이 값들의 직전의 정상 값으로 대체하여 비정상적인 데이터를 제거하였다.

표 6. 소스 스트림에서 정상 값과 누락 값 및 비 정상치의 비율 (단위: 개)
Table 6. Ratio of normal values and missing/faulty values (Unit: one).

센서 종류	정상 값	누락 + 비 정상치
Temperature	96712 (99.42%)	565 (0.58%)
Humidity	96795 (99.43%)	553 (0.57%)
Illumination	96932 (99.55%)	441 (0.45%)
InfraredRays	96844 (99.45%)	535 (0.55%)

2. 데이터 축소(Data Reduction)

저장 관리자는 수집한 데이터를 매번 그대로 저장하지 않고, 동일한 값이 발생한 시간 구간에 대해 저장하여 데이터를 축소한다. 각 센서가 수집한 데이터는 일정한 임계값 이내로 변할 경우 이전 값과 동일한 값으로 간주하여 데이터 축소의 효율을 높였다. 이 임계값은 각 센서가 측정할 수 있는 범위의 1% 이내로 설정하였으며 그 값은 다음과 같다.

온도 : ±1 / 습도 : ±1 / 조도 : ±10 / 적외선 : ±3

센서들은 설치된 장소에 따라 서로 다른 특징의 스트림 데이터를 생성하므로, 각 센서 종류마다 실내와 실외에서 데이터를 수집하였다. 이렇게 수집한 소스 스트림의 저장 공간과 제안된 구간 저장 방식의 저장 공간의 크기를 서로 비교한 결과는 아래 표 7~표 10과 같다.

표 7에서부터 표 10까지의 센서 별 수집 결과를 살펴보면 온도와 습도의 경우 실내에서 좀 더 축소율이 높은 것을 볼 수 있고, 조도나 적외선의 경우 실내에서 점등과 같은 요소 때문에 좀 더 축소율이 높은 것을 볼 수 있다. 실외는 여러 환경적인 변수들이 존재하지만, 그래도 1% 이하로 축소되는 결과를 볼 수 있다. 전체적으로 데이터가 수십에서 수백분의 1로 축소됨을 확인할 수 있다. 이렇게 스트림 데이터가 축소되더라도 오랜 기간 동안 축적될 경우 여전히 엄청나게 큰 저장 공간을 차지할 수 있다. 이를 해결하기 위해, 경동 시간 구조를 활용하여 발생한 시점에 따라 서로 다른 시간 간격으로 데이터를 저장하였다. 경동 시간 구조는 설정하는 시간 간격에 따라 성능이 좌우된다. 건물 관리 및 화재 감시 응용의 특성에 맞춰 시간 간격을 설정하고 소

표 7. 온도 센서의 위치에 따른 축소 비율 비교
Table 7. Comparison to reduction ratio over the location of temperature sensors.

위치	소스 스트림 양	구간 저장 방식(±1)
실내	1,677,136 KB	800 KB (0.05%)
실외	1,249,184 KB	1,904 KB (0.15%)

표 8. 습도 센서의 위치에 따른 축소 비율 비교
Table 8. Comparison to reduction ratio over the location of humidity sensors.

위치	소스 스트림 양	구간 저장 방식(±1)
실내	1,678,080 KB	4,720 KB (0.28%)
실외	1,250,032 KB	13,536 KB (1.08%)

표 9. 조도 센서의 위치에 따른 축소 비율 비교
Table 9. Comparison to reduction ratio over the location of illumination sensors.

위치	소스 스트림 양	구간 저장 방식(±10)
실내	1,678,144 KB	25,536 KB (1.52%)
실외	1,250,208 KB	21,984 KB (1.76%)

표 10. 적외선 센서의 위치에 따른 축소 비율 비교
Table 10. Comparison to reduction ratio over the location of infrared ray sensors.

위치	소스 스트림 양	구간 저장 방식(±3)
실내	1,678,528 KB	7,936 KB (0.47%)
실외	1,250,528 KB	70,752 KB (5.66%)

표 11. 실내 온도 스트림 데이터에 대한 경동 시간 구조의 저장 공간 축소 비율 비교
Table 11. Comparison to storage space reduction ratio of tilted time frame for stream data of temperature sensors located inside rooms.

수집 기간	저장간격	소스 스트림	구간저장방식의 경동 시간 구조
1분	3초	320B	-
1시간	1분	20KB	-
1일	1시간	460KB	220B
1년	1시간	170MB	80KB
10년	1시간	1.7GB	800KB

스 스트림 데이터가 얼마나 축소되는지 측정한 결과가 아래 표 11에 나타나 있다. 아래 결과는 비교적 완만하게 값이 변화하는 실내에 설치된 온도 센서 한 개에 대한 스트림 데이터의 저장 공간을 측정한 것이다.

표 11에서 보는 바와 같이, 경동 시간 구조의 시간 간격 값을 설정하기에 따라, 한 온도 센서에서 수집되는 스트림 데이터를 건물의 생존기간 전체에 대해 저장하더라도 수 MB~수십 MB 단위로 현저히 축소할 수 있다. 건물에 수만~수십만 개의 센서를 설치하더라도 이들을 충분히 데이터베이스에 유지할 수 있다. 따라서 이런 방법을 사용할 경우 건물 관리 및 화재 감시 응용의 스트림 데이터를 데이터베이스 유지하는 것이 가능하며, 이러한 응용을 위한 스트림 데이터베이스를 구축하는 것이 실현 가능할 것이다.

또 위 표에서 측정된 경동 시간 구조의 저장 공간은 앞 절에 있는 식 (4)의 이론적인 크기의 약 55% 정도를 갖는다. 이는 경동 시간 구조로 데이터를 수집한 후에 구간 저장 방식에 의해 데이터가 더 축소됐기 때문이며, 개발된 시스템이 이론적인 저장 공간 크기 이내에서 스트림 데이터를 저장하고 있음을 알 수 있다.

VI. 결 론

본 논문에서는 유비쿼터스 센서 네트워크에서 발생하는 스트림 데이터를 데이터베이스에 저장하고 관리하는 스트림 저장 관리자를 개발하였다. 이 저장 관리자는 수집된 데이터 중에서 센서의 물리적 특성으로 인한 누락 값 및 비 정상치, 잡음 등을 데이터 정제를 통해 데이터를 개선하며, 연속적이며 지속적인 대량의 스트림 데이터를 효율적으로 그 크기를 축소하여 저장한다. 이 저장 관리자를 .NET Framework 위에서 C# 언어와 MS SQL Server 2005를 이용하여 구현하였다.

구현된 스트림 저장 관리자의 성능과 실효성을 검증하기 위해, 건물 관리 및 화재 감사 응용의 스트림 데이터를 대상으로 실험하였다. 실험 결과 소스 스트림 데이터를 손실없이 수백분의 일로 축소하였다. 또한 스트림의 저장 시간 간격을 다중화할 경우 각 센서의 전체 스트림을 유지하는 것도 가능하여, 화재 감시와 유사한 실제 응용에 이용 가능할 것이다.

저장 관리자를 이용하여 저장한 스트림 데이터는 사용자 질의 처리에 이용하거나, 스트림 데이터의 분석 또는 스트림 데이터 웨어하우스 구축 등에 이용될 수 있다. 따라서 향후 연구로 저장 관리자의 효율을 더욱 높일 수 있는 방법을 연구하고, 축소된 스트림 데이터를 이용한 연속 질의 최적화 연구와 다차원 큐브 생성 방법에 대한 연구도 진행하고자 한다.

참 고 문 헌

[1] R. Motwani, J. Widom, A. Arasu, B. Bobcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query Processing, Resource Management, and Approximation in a Data Stream Management System," In *Proc. of Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003

[2] L. Golab and M. T. Ozsu, "Issues in Data Stream Management," *SIGMOD Record*, vol. 32, no. 2, June 2003.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," In *Proc. of ACM SIGACT-SIGMOD-SIGART Sym. on Principles of Database Systems*, pp. 1-16, Wisconsin, USA, June 2002.

[4] H. Gonzalez, J. Han, X. Li, and D. Klabjan, "Warehousing and Analyzing Massive RFID Data Sets," In *Proc. of IEEE Data Engineering Conference*, pp. 1-10, 2006.

[5] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang, "NiagaraCQ: A Scalable Continuous Query System for Internet Databases," In *Proc. of ACM SIGMOD Conf. on Management of Data*, pp. 379-390, Dallas, Texa, USA, May 2000.

[6] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Wrishnamurthy, S. Madden, Y. Raman, F. Reiss, M. Shah, "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," In *Proc. of Conf. on Innovative Data Systems Research*, Asilomar, CA, USA, Jan. 2003.

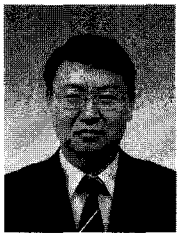
[7] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A New Model and Architecture for Data Stream Management," *VLDB Journal*, vol. 12, no. 2, pp. 120-139, 2003.

[8] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, and Y. D. Cai, "Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams," *Distributed and Parallel*

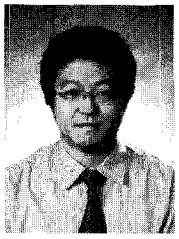
Databases, vol. 18, no. 2, pp. 173-197, 2005.
[9] 도기석, 박석, "스트림 데이터의 다차원 분석에서 평균 응답 시간을 줄이는 스트림 큐브," *한국 컴퓨터종합학술대회 2005 발표논문집*, vol. 32, no.1(B), pp. 55-57, 2005.
[10] 서대홍, 양우석, 이원석, "데이터 스트림에서 동적 데이터 큐브," *한국정보과학회 논문지: 데이터베이스*, 제35권 제4호, pp. 319-332, 2008년 8월.
[11] J. Han and M. Kamber, *Data Mining: concepts and techniques*, Morgan Kaufmann Publishers, 2006.

저 자 소 개

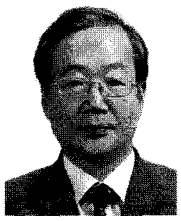
이 수 안(학생회원)
2008년 강원대학교 컴퓨터과학과 학사 졸업
2009년 강원대학교 컴퓨터과학과 석사과정 재학중
<주관심분야 : 데이터웨어하우징, OLAP, 스트림 데이터, 유비쿼터스 센서 네트워크, 데이터 마이닝>



김 진 호(정회원)
1982년 경북대학교 전자공학과 전산전공 학사 졸업.
1985년 KAIST 전산학과 석사 졸업.
1990년 KAIST 전산학과 박사 졸업.
1990년~현재 강원대학교 컴퓨터과학과 교수
<주관심분야 : OLAP, 데이터웨어하우스, 데이터 마이닝, 임베디드/실시간 데이터베이스, 정보검색, 데이터 모델링>



신 성 현(정회원)
2000년 관동대학교 컴퓨터공학과 학사 졸업.
2002년 강원대학교 컴퓨터과학과 석사 졸업.
2007년 강원대학교 컴퓨터과학과 박사 졸업.
2007년~현재 한양대학교 BK21 사업단 정보기술분야 Post-Doc.
<주관심분야 : 데이터 웨어하우스, OLAP, 데이터 마이닝, 이동 객체 데이터베이스>



남 시 병(정회원)
1979년 단국대학교 전자공학과 학사 졸업.
1982년 단국대학교 전자공학과 석사 졸업.
1994년 단국대학교 전자공학과 박사 졸업.
1986년~현재 강원대학교 전자공학과 교수
<주관심분야 : 임베디드 시스템, 센서 네트워크, 인터페이스, 패턴인식>