State Thread 기반 실시간 데이터 스트림 관리 시스템

박원빈*, 송창근**, 고영웅*
*한림대학교 컴퓨터공학과
**한림대학교 유비쿼터스 컴퓨팅공학과
e-mail : {wonvien, cgsong, yuko}@hallym.ac.kr

Real-Time Data Stream Management System Using State Thread

Won-Vien Park*, Chang-Geun Song**,Young-Woong Ko*
*Dept of Computer Engineering, Hallym University
**Dept of Ubiquitous Computing, Hallym University

요 약

RFID를 기반으로 유비쿼터스 환경의 응용 서비스를 지원하는 미들웨어는 지속적으로 끊임없이 입력되는 데이터 스트림을 실시간으로 처리하고 응용 서비스에서 요구하는 결과를 획득하여 전달해야 한다. 이와 같은 요구사항을 만족하기 위해 데이터 스트림 관리 시스템(DSMS)이 제안되었으며 다양한 연구가 시도되고 있다. 본 논문에서는 대량의 이벤트가 입력되는 환경에서 우선순위가 높은 질의를 실시간으로 처리하기 위한 DSMS를 제안하고 있다. 본 연구는 스탠포드의 STREAM 프로젝트를 활용하여 설계 및 구현하였으며, 각 쿼리를 State Thread로 동작시키는 방법을 이용하였다. 쓰레드 라이브러리의 스케줄러 부분을 실시간 스케줄러로 개선하는 작업을 진행하였으며, 실험을 통하여 쓰레드 스케줄러가 질의에 대해서 실시간으로 스케줄링을 할 수 있음을 보이고 있다.

1. 서론

최근 데이터 스트림(data stream)이 끊임없이 발생되고 처리되는 응용 분야가 다수 등장하고 있다. 특히 물류 및 유통 시장이 급격히 발달하면서 RFID 태그에서 발생되는 이벤트를 인식하고 처리하는 RFID 미들웨어 분야에 중요하게 다루어진다. 여기서 데이터 스트림은 연속적으로 발생되는 데이터의 흐름을 의미하며, 일반적으로 대용량의 데이터에 해당된다. 즉. 데이터 스트림 전체를 데이터베이스에 저장하고 관리하는 것은 매우 큰 저장 공간을 요구하며 특정한 데이터를 찾아내기 위해서는 높은 CPU 사용률과 입출력을 수반하게 된다[1,2].

RFID 미들웨어에서는 RFID 리더에서 감지된 이벤트를 사용자 또는 응용 등의 질의의 결과에 해당하는 값을 전송하는 형태를 취하고 있다. 하지만 RFID 미들웨어에 한꺼번에 많은 양의 데이터가 입력되거나 RFID 미들웨어가 동작되는 서버에서 다른 연산으로 인하여 입력된 스트림을 처리할 만한 자원이 부족할 때, 데이터의 정체 현상으로 인한 메모리 오버 현상이나 질의에 대한 올바른 결과를 전송하지 못하는 현상이 발생한다. 현재 사용되고 있는 대부분의 RFID 미들웨어는 데이터베이스 관리 시스템(DBMS : Database Management System)을 이용하여 RFID 이벤트 데이터를 관리하고 있다. DBMS는 RFID 스트림 데이터의 요구사항을 만족하기 위하여 데이터 스트림을 일반 테이블에 대한 삽입으로 처리하고, 연속질의는 트리거(trigger) 또는 객체화 된 뷰(materialized view)로 처리할 수 있다. 하지만, 삽입 연산 부하가 크고, 지원하는 트리거의 수가 한계가 있으며, 트리거로 표현할 수 있는 조건이 제한적이라는 문제점을 갖는다. 따라서 대용량의 RFID 이벤트를 효율적으로 처리할 수 있는 소프트웨어가 필요하며, 이와 같은 요구에 적합하게 사용할 수 있는 기술로 DSMS(Data Stream Management System)이 연구되고 있다[3,4,5,6,7]. DSMS는 지속적으로 입력되는 대량의 데이터 스트림을 처리하기 위해서 순서와 시간을 기반으로 하는 데이터 모델과 질의가 허용된다. 또한 질의에 대한 결과를 얻기 위하여 입력되는 모든 데이터 스트림을 대상으로 하는 블로킹(blocking) 연산을 사용하지 않는 특성을 가지며, 실시간으로 데이터 스트림을 모니터링하여 부적절한 데이터에 대하여 빠르게 대응할 수 있는 장점을 지닌다.

본 연구에서는 데이터 스트림에 대해서 실시간 쿼리가 수행될 수 있는 DSMS 시스템을 제안한다. 주요 아이디어는 사용자가 등록하는 쿼리를 쓰레드로 동작이 되도록 하였으며, 쓰레드 스케줄러를 실시간으로 동작하도록 개선하는 작업을 수행하였다. 또한 DSMS 시스템을 새롭게 구현하지 않고 기존에 널리 사용되고 있는 스탠포드 대학의 STREAM 시스템에 실시간적인 처리를 지원하는 방향으

본 연구는 교육과학기술부와 한국연구재단의 지역혁신인력양성사업으로 수행된 연구결과임

로 접근하였다. 본 연구에서는 기존의 쿼리 요청 부분을 개선하여 각 쿼리가 주기적/비주기적으로 동작이 될 수 있도록 등록하는 클라이언트 모듈을 새롭게 구현하였고, 각 쿼리의 파라미터로 수행 주기(period) 및 수행시간(execution time) 값, 쿼리의 우선 순위를 사용자가 입력할 수 있게 하였다.

2. 실시간 쿼리 스케줄링

2.1 기존 실시간 쿼리 스케줄링 모델의 제한점

대표적인 DSMS인 Aurora[5] 연구에서는 어떤 오퍼레이터를 어떤 순서대로 처리할 것이며 얼마나 많은 튜플들을 한꺼번에 수행할 수 있는지에 연구 초점이 맞춰져 있었고, 질의를 실시간으로 스케줄링 하는 연구는 이뤄지지 않았다. QStream[10] 등의 연구에서는 실시간 오퍼레이터 스케줄링에 대한 연구가 제시되었다. 하지만, 실시간 처리가 가능한 RTAI(Real-time Application Interface) 플랫폼에서 각 오퍼레이터를 실시간 프로세스로 매핑하여 스케줄링하고 있으며, 오퍼레이터간의 데이터 공유는 RT-FIFO와 같은 실시간 IPC 모듈을 이용한다는 특징이 있다. 이와 같은 방식은 각 오퍼레이터의 실시간 처리가 가능하지만 프로세스 생성 및 IPC 설비 이용에 오버헤드가 매우 높은 방식이다. 따라서 엄격한 경성 실시간 작업이 필요한 상황을 제외하고는 적용하기 어렵다는 단점을 가진다.

이외에도 RTSTREAM[7] 등의 연구에서 QoS(Quality of Service)에 기반한 실시간 오퍼레이터 스케줄링을 지원하고 있지만, 과부하 상황이 발생하는 경우에 전반적으로 부정확한 쿼리 결과를 보이는 문제점이 있다. 즉, 대부분의 DSMS 시스템에서는 과부하 문제를 부하 줄임(load shedding)으로 해결함으로써 실시간 쿼리가 제한 시간 내에 수행될 수 있도록 접근하고 있다. 이와 같은 접근 방식의 배경에는 부분적인 결과가 제한 시간을 넘기고 도착한 데이터보다 더 바람직하다는 전제사항이 있다. 시스템이 과부하 상황에 도달이 되는 경우에 입력되는 이벤트 데이터를 줄이는 것은 과부하 제어에 효과적일 수 있지만, DSMS 시스템이 부정확한 데이터를 가지고 쿼리 작업을 수행해야 한다는 단점을 가진다.

2.2 제안하는 쿼리 스케줄링 모델

본 연구에서는 사용자가 요청하는 쿼리를 사용자 수준의 쓰레드로 구성하고 각 쓰레드가 실시간적인 처리가 될 수 있도록 우선순위를 부여하여 제한된 시간 내에 쿼리가 동작이 될 수 있도록 하는 방안을 제시한다. 이를 위하여 다음과 같은 두 가지 방식에서 접근하였다.

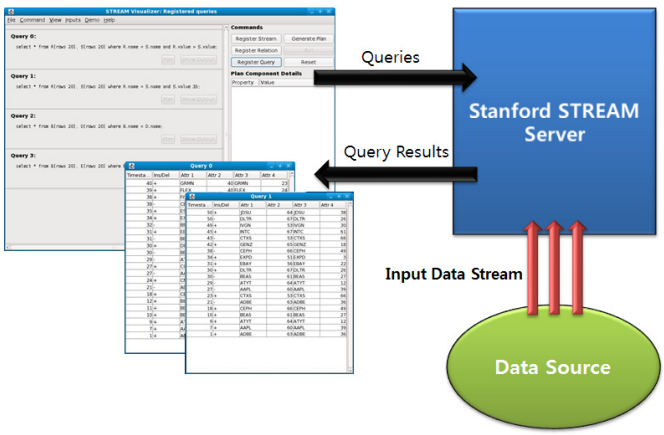
첫 번째, 쓰레드 라이브러리의 스케줄러를 개선하여 우선순위를 기반으로 동작이 가능하도록 개선하였다. 본 연구에서는 다양한 쓰레드 라이브러리 중에서 성능이 우수한 것으로 알려진 State Thread를 이용하여 시스템을 구성하였다. State Thread는 빠르고 매우 폭넓게 확장 가능한 네트워크 어플리케이션에 적합한 소규모의 어플리케이션 라이브러리이다. POSIX 규약을 따르고, 일반적인 용도로 쓰이는 pthared와는 다르게State Thread는 전역 데이타는 상호 베타적 잠금(뮤텍스)로 보호될 필요가 없고, 성능 향상과 동시에 프로그래밍과 디버깅을 단순하게 만들어 준다. State Thread의 API와 내부는 특정 어플리케이션 영역의 필요위주로 디자인된 특징이 있다. 쓰레드 생성 및 문맥 전환을 위한 시스템 호출 수는 가능한 한 최소화하도록 되어있고, 쓰레드의 선점적 혹은 우선순위 기반의 스케줄링은 낮은 쓰레드 관리상의 과부하를 고려하여 생략되어 있다. 따라서 본 논문에서는 State Thread를 이용하여 쓰레드를 생성하고 이 State Thread를 제어할 수 있는 스케줄러를 추가하였다.

두 번째, 본 연구에서는 실시간 쿼리 스케줄링에 초점을 맞추고 있기 때문에 전체 시스템을 새롭게 설계 및 구현하는 것보다는 기존에 사용되는 시스템을 개선하는 것으로 접근하였다. 이를 위하여 공개적으로 소스가 공개된 스탠포드의 STREAM[3] DSMS를 기반으로 실시간 쿼리 스케줄링을 추가하는 작업을 수행하였다. STREAM 프로젝트는 거대하고 끊임없이 생성되는 데이터 스트림을 효과적으로 처리하고, 연속적으로 입력되는 데이터 스트림에 대한 연속 질의(CQ: Continuous Query)를 지원하기 위하여 스탠포드 대학에서 개발하였다. STREAM 프로젝트는 기존의 표준 SQL을 확장한 CQL(Continuous Query Language)를 사용하였고, 제약조건 탐색, 연산자 스케줄링 등을 통하여 자원 사용률을 줄였으며, 부하 감소(load shedding)을 이용하여 근사값 계산을 허용하고 있다.

3. 시스템 설계 및 구현

3.1 STREAM 동작 모델

그림 1은 스탠포드의 STREAM 시스템에 대한 대략적인 실행순서를 나타내고 있다.



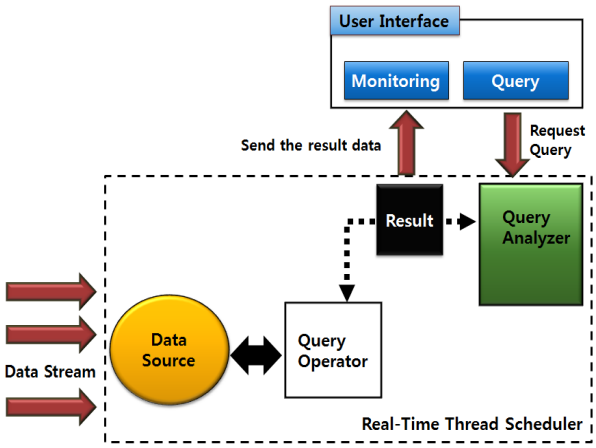
[그림 1] STREAM 동작 모델

서버가 실행되면 메모리로부터 데이터 스트림들이 서버로 유입된다. 클라이언트에서는 사용자가 필요한 쿼리들을 작

성하여 서버에 전송하게 되는데, 이 때 서버는 클라이언트로부터 받은 쿼리들을 분석하여 그에 맞는 결과 값을 데이터에서 찾아 클라이언트에 전송한다. STREAM은 기본적으로 라운드로빈 스케줄러 방식을 사용해서 실행되고, 서버는 클라이언트에서 쿼리가 들어올 때까지 대기한다. 클라이언트에서 사용자가 원하는 쿼리를 작성하여 서버에 보내는 작업은 한 번만 보내지게 되며, 서버에서 쿼리들의 결과를 처리한 후에 클라이언트에게 결과 값을 전송하면 클라이언트에서는 그 결과 값을 받아 화면에 출력하고 끝나게 된다. 만약 사용자가 쿼리에 대한 결과 값을 다시 보고 싶거나 새로운 쿼리를 작성하려고 한다면 이전에 실행되었던 결과 값들과 쿼리문들을 지우고 새로 작성해야 한다는 번거로움이 있다.

3.2 제안하는 시스템의 동작 모델

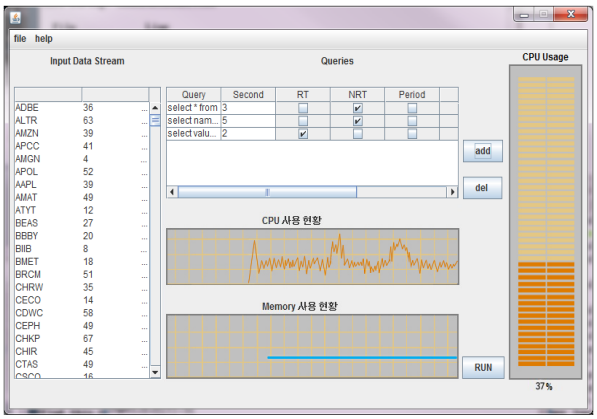
본 연구에서 제안하는 시스템은 그림 2와 같이 사용자 인터페이스(Query Interface와 Monitoring Interface를 포함한다.), Query Analyzer, Real-time Thread Scheduler, Query Operator, Storage 등으로 구성된다.



[그림 2] 제안하는 DSMS 모델

대부분의 컴포넌트는 STREAM의 모듈을 그대로 사용하고 있으며, 쿼리 오퍼레이터를 스케줄링하는 부분에 있어서 State Thread로 동작이 되도록 수정하였다. 또한 사용자 인터페이스에서 요청되는 쿼리들이 서버에서 등록되어 주기적으로 수행이 될 수 있도록 하였다. 기존의 STREAM 서버가 매번 쿼리를 요청하고 결과를 받는 방식에서 한번 쿼리를 등록하면 주기적으로 지속적으로 수행되는 모델로 변경한 것이다.

User Interface 사용자는 그래픽 유저 인터페이스로 구현된 인터페이스로 다양한 질의를 작성한다. 수행될 내용을 인터페이스에 입력하고 각 쿼리들이 어떤 주기로 동작할 것인지 정한다. 그리고 실시간 또는 비실시간을 선택한다. 다음 그림 3은 사용자 인터페이스를 보이고 있다. 사용자는 각 쿼리에 대해서 실시간/비실시간을 선택하고, 주기와 수행시간을 입력하게 된다. 서버의 동작에 대해서는 CPU와 메모리 사용량이 표시되어 현재 동작되는 시스템의 현황을 알 수 있다.



[그림 3] 사용자 인터페이스

Query Analyzer 인터페이스로부터 받은 쿼리를 분석하는 작업을 수행하고, 쿼리를 구성하는 오퍼레이터를 쓰레드로 매핑하는 작업을 한다. 본 모듈은 STREAM에서 제공하는 것을 그대로 사용하였다. 이때 쿼리 스케줄링이 처리됨에 있어서 STREAM에서 라운드 로빈 방식으로 쿼리를 처리하는 방식에서 주기와 수행시간을 고려한 실시간 스케줄링 방식으로 처리하도록 수정하였다. State Thread는 라운드 로빈에 기반한 쓰레드 스케줄링 알고리즘을 사용하고 있으며, 본 연구에서는 쓰레드 스케줄링 알고리즘이 실시간 쓰레드를 지원할 수 있도록 수정하였다. 스케줄링에 사용되는 큐는 크게 실시간 큐와 비실시간 큐로 나뉜다. 실시간 큐에는 주기적으로 수행되는 실시간 쿼리를 구성하는 쓰레드들이 대기하게 된다.

Monitoring Interface 서버에서 쿼리를 분석하여 그에 맞는 데이터 스트림을 실시간으로 결과 값을 보낸다. 이때, User Interface에서는 Monitoring Interface을 이용하여 사용자가 각 쿼리에 맞는 결과 값을 확인 할 수 있도록 한다.

4. 실험 및 결과

4.1 실험 환경 설정

제안하는 DSMS 모델의 성능을 측정하기 위하여 다음과 같은 환경에서 실험을 진행하였다. 우선, 물품 등에 부착되어있는 태그들로부터 대량의 데이터를 RFID 리더기로 받아오는 등의 구성을 하기에 어려움이 있기 때문에 데이터 스트림들을 디스크에 저장하고 일정한 비율로 스트림 데이터를 전송하도록 구성하였다.

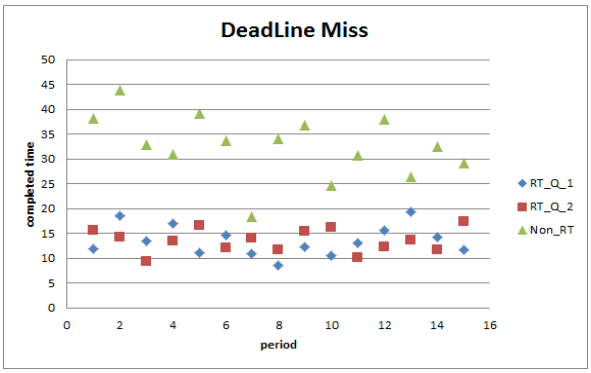
[표 1] 실험 환경 설정

하드웨어 사양	
CPU	Intel Xeon E5520 2.27Ghz Quad Cord
HDD	Seagate 3TB
RAM	6GB
소프트웨어 사양	
OS	Fedora 11
Language	C++, Java

실시간 쿼리가 제한시간 내에 수행되는지 확인하였으며, 쿼리들의 수를 늘려 CPU 과부하를 발생시켰을 경우에도 실시간 쿼리가 데드라인을 만족하는지 측정하였다. 각각의 쿼리를 실시간/비실시간으로 선택한 후 실시간 스케줄링 방식으로 쓰레드 스케줄링을 수행할 때, 비실시간 쿼리들보다 실시간으로 선택되어있는 쿼리들이 먼저 수행되는지를 측정하였다.

4.2 실험 결과

본 연구 내용의 유용성을 보이기 위하여 다수의 쿼리를 등록하여 주기적으로 작업을 수행하게 하였으며, 시스템을 비실시간적인 쿼리를 동작시켜서 과부하게 발생시키는 방식을 적용하여 실험을 진행하였다. 이때 주기와 데드라인은 동일한 값을 사용하였다. (다음 그래프에서 단위는 10ms임)



[그림 4] deadline=20 일 때 결과 그래프

그림 4는 실시간 쿼리의 데드라인을 20 (0.2초)으로 했을 때 나온 결과 값을 보여준다. 실시간 쿼리들은 모두 데드라인 내에 실행하였다. 비실시간 쿼리들 중에서 데드라인 내에 수행되어 마친 쿼리도 볼 수 있었으나, 실시간 쿼리의 수행이 끝난 후에 처리되었으므로 비실시간 쿼리들은 모두 실시간 쿼리 실행 후에 실행되었음을 확인할 수 있다.

5. 결론

본 연구에서는 데이터 스트림에 대해서 실시간 쿼리가 수행될 수 있는 DSMS 시스템을 제안하고 시스템의 설계 및 구현에 대해서 기술하였다. 주요 기여사항은 사용자가 등록하는 쿼리들을 쓰레드에 매핑시켜서 쓰레드 스케줄러가 실시간적으로 스케줄링이 될 수 있도록 하는 부분에 있다. 이를 위하여 DSMS 시스템을 새롭게 구현하지 않고 기존에 널리 사용되고 있는 스탠포드 대학의 STREAM 시스템에 실시간적인 처리를 지원하는 방향으로 접근하였다.

본 연구에서는 기존의 쿼리 요청 부분을 개선하여 각 쿼리가 주기적/비주기적으로 동작이 될 수 있도록 등록하는 클라이언트 모듈을 새롭게 구현하였고, 각 쿼리의 파라미터로 수행 주기(period) 및 수행시간(execution time) 값, 쿼리의 우선 순위를 사용자가 입력할 수 있게 하였다. 실험 결과 제안하는 방식이 실시간적으로 쿼리를 수행시킬 수 있음을 보이고 있다. 행후, 스케줄링 정책과 실험의 카테고리를 다양화하여 결과를 제시할 계획이다.

참고문헌

[1] F. Wang, S. Liu, P. Liu and Y. Bai, "Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams", 10th International Conference on Extending Database Technology (EDBT'2006), Munich, Germany, March 2006.

[2] Kaushik Dutta, Krithi Ramamritham, Kamlesh Laddhad, Karthik B, "Real-Time Event Handling in an RFID Middleware System", Workshop on Databases in Networked Information Systems(DNIS) 2007.

[3] Motwani et al. (Stanford group), "Query Processing, Resource Management, and Approximation in a Data Stream Management System", CIDR 2003.

[4] Lukasz Golab and M. Tamer Ozsu, "Issue in Data Stream Management", SIGMOD Record, Vol.32, No.2, June 2003.

[5] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik. "Aurora: A New Model and Architecture for Data Stream Management", In VLDB Journal, 2003.

[6] A. Arasu, B. Babcock et al. (Stanford group), "STREAM : The Stanford Stream Data Manager", IEEE Data Engineering Bulletin, March 2003.

[7] Y. Wei, S. H. Son, and J. Stankovic, "RTStream: Real-Time Query Processing for Data Streams," IEEE Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'06), Gyeongju, Korea, April 2006.

[8] Sirish Chandrasekaran, etal. "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," CIDR 2003.

[10] Sven Schmidt, Thomas Legler, Sebastian Schär, Wolfgang Lehner, "Robust Real-time Query Processing with QStream." VLDB 2005: 1299-1302