RFID 입력 데이터 스트림에 대한 다중 버퍼 기반의 고속 데이터 처리 알고리즘

한 수○ 신승호
시립인천대학교 컴퓨터공학과
pucktan○@incheon.ac.kr, shin0354@incheon.ac.kr

A High-Speed Data Processing Algorithm for RFID Input Data Stream Using Multi-Buffer

Soo Han○ SeungHo Shin
Dept. of Computer Engineering, University of Incheon

요 약

RFID를 기반으로 유비쿼터스 환경의 응용 서비스를 지원하는 미들웨어는 지속적으로 끊임없이 입력되는 데이터를 정확하게 실시간으로 처리하고 응용 서비스에서 질의하는 결과를 획득해서 전달하여야 한다. 이와 같은 지속적으로 입력되는 대량의 데이터 스트림을 처리하기 위해서 데이터 스트림 관리 시스템(Data Stream Management System: DSMS)을 개발하기 위한 연구가 진행되고 있다. 기존에 연구 되는 데이터 스트림에 대한 알고리즘은 대부분 연속 질의 결과들 사이의 평균 오차를 줄이고, 부하 발생 시 데이터의 우선순위에 따라 버리는 것에 초점이 맞추어져 있다.

본 논문에서는 RFID EPC 라는 데이터 특성에 맞추어 다중버퍼를 이용함으로써 고속의 데이터 처리 능력을 얻고, 각 버퍼마다 일정한 규칙을 통해 질의에 있어서도 빠른 대응을 할 수 있는 알고리즘을 제안한다. 본 논문은 현재 DSMS의 관련 연구와 고속 데이터 처리의 필요성을 말하고, 제안하는 알고리즘 설명과 시뮬레이션을 통해 단일버퍼와 다중버퍼일 경우 데이터 처리 속도 성능 평가와 제안한 알고리즘에 맞도록 버퍼가 생성 되는지 테스트하는 것으로 구성된다.

1. 서 론

유비쿼터스 컴퓨팅 환경 실현을 위한 연구 개발이 활발이 이루어 지고 있다. 유비쿼터스 컴퓨팅의 차세대 핵심 기술로 RFID 기술이 주목 받고 있으며, RFID를 기반으로 하는 유비쿼터스 서비스 환경 구축을 위한 연구 개발이 진행되고 있다. 사용자의 다양한 요구에 대하여 시간과 장소에 상관없이 응답 가능토록 하는 유비쿼터스 서비스 환경을 위해서는 다양한 컴퓨팅 디바이스가 생성하는 다양한 대량의 데이터를 가공하여 사용자 또는 응용 서비스에게 전달해주는 미들웨어 기술이 요구 된다. 특히, RFID 시스템의 경우 데이터 스트림이 지속적으로 대량이 발생되고, 실시간적이여야 한다는 특성이 있다. 현재 RFID 관련 하드웨어 장비 기술의 발전으로 한 개의 리더는 여러 개의 Tag 데이터를 고속으로 수집하고 미들웨어에 전송한다[1]. 이 때 미들웨어에서 수집되는 데이터를 버퍼에 저장하고, 사용자 또는 응용 서비스의 질의에 대한 결과 값을 상위 모듈에 전송한다. 그런데 미들웨어가 되는 하드웨어의 한정 된 자원과 연산 속도로 인하여 데이터의 수집과 질의 연산을 제대로 수행을 하지 못하고 데이터의 정체 현상으로 인한 메모리 오버 현상이나, 질의에 대한 올바른 결과를 전송하지 못하는 현상이 발생 한다. 그래서 RFID 미들웨어 시스템에는 데이터 스트림 관리 모듈이 내재 된다[2].

데이터 스트림 관리 시스템(Data Stream Management System: DSMS)은 데이터 스트림과 저장된 릴레이션에 대해 다수의 연속 질의를 처리하는 시스템을 의미한다[3,4,5]. 현재 DSMS를 연구하는 주요 프로젝트에는 Aurora[6], NiagaraCQ[7], STREAM[8]등이 있다.

기존 연구에서는 일반적인 데이터에 초점을 맞추어져 있고 연구 주제 또한 연속 질의에 대한 결과의 올바른 응답 오차율을 줄이는 것과, 부하 발생 시 입력 속도를 기반으로 데이터를 제한하는 무작위 부하 제한에 대한 것이다.

본 논문에서는 RFID 시스템에서 발생되는 데이터, 즉 EPC 데이터 수집 시 미들웨어 자체에서 데이터가 고속으로 처리 될 수 있으며, 질의를 받을 시 질의 결과를 고속으로 연산해 내는 알고리즘을 제안한다. 본 논문에서는 다음과 같이 두 가지 사항을 목표로 한다.

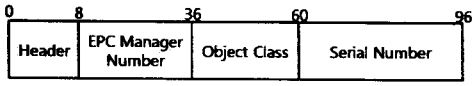
- 다중 버퍼를 이용하여 데이터가 상위 모듈로 빠르게 전송 될 수 있다. 한 개의 버퍼를 통해서 FIFO (First Input First Out) 방식으로 데이터 전송 할 경우보다 다중 버퍼를 통해서 데이터를 상위 모듈로 전송할 때 데이터 처리의 효율을 높일 수 있다.
- 다중 버퍼 이용 시 각 버퍼에 특성을 부여하여 해당하는 버퍼에 데이터를 입력함으로서 질의를 받을 시 특성에 맞는 버퍼에서 데이터를 찾음으로서 빠른 질의 결과를 획득 할수 있다.

본 논문의 구성은 다음과 같다. 먼저 2절에서는 관련연구들을 살펴보고, 3절에서는 본 논문에서 제안하는 다중 버퍼를 이용한 데이터 처리 알고리즘을 설명한다. 그리고 4절에서는 제안한 알고리즘의 성능평과를 하고, 5절에서 결론 및 향후 연구 방향에 대해 논의 한다.

2. 관련 연구

2.1 RFID 데이터 [9]

RFID 데이터의 정의는 일반적으로 태그 값으로 알고 있는 코드 값을 말한다. 코드 체계의 종류에는 여러 가지가 있지만 세계적으로 표준으로서 사용이 확산되고 있는 전자코드(Electronic Product Code: EPC)를 대표적으로 말할 수 있다. EPCglobal에서 보급하는 RFID 태그 입력용 코드 체계로서, 총 96비트의 정형화 된 데이터를 입력할 수 있는다. 본 논문에서 사용하는 데이터는 EPC 코드체계에 맞는 데이터를 사용한다. EPC 데이터의 구조는 [그림1]과 같다.



[그림 1] EPC 코드 구조

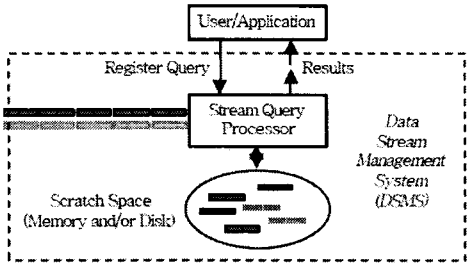
- 헤더(Header-8비트) : 데이터 유형 및 길이를 정의
- EPC 관리자(EPC Manager Number-28비트) : EAN 바코드의 업체코드에 해당하며 각국 EAN 회원기관이 할당함
- 객체 분류 번호(Object Class-24비트) : 바코드의 상품 품목 코드에 해당하며 사업 업체가 할당함
- 일련번호(Serial Number-36비트) : 동일 상품에 부여되는 고유한 식별번호로서 사업 업체가 할당함

본 논문에서는 EPC 데이터의 고유 식별자를 이용해 각 버퍼에 입력 할 때의 특성 부여 기준을 생성한다.

2.2 데이터 스트림 관리 시스템(Data Stream Management System: DSMS) [3,4,5]

DSMS에서 사용자 또는 어플리케이션은 입력 되는 데이터 스트림으로부터 원하는 결과를 얻기 위한 질의를 등록해 두고, 지속적으로 입력이 되는 데이터 스트림을 처리하는 스트림 질의 프로세서(stream query processor)는 등록된 질의에 대한 결과를 지속적으로 사용자 또는 어플리케이션으로 반환한다. 이와 같이 특정 저장소에 저장되어 있는 한정된 데이터 보다는 끊임없이 계속 만들어지는 데이터에 대한 연속 질의(continuous query)에 대한 처리를 수행하여야 하며, 연속 질의의 결과를 얻기 위해 요구되는 데이터를 임시로 유지하기 위한 임시 저장 공간(scratch space)을 관리하여야 한다. DSMS는 [그림 2]와 같은 구조를 가진다.

DSMS 연구 프로젝트로는 Aurora 프로젝트, STREAM 프로젝트, NiagaraCQ 프로젝트 등이 있다.



[그림 2] 데이터 스티림 관리 시스템(DSMS)

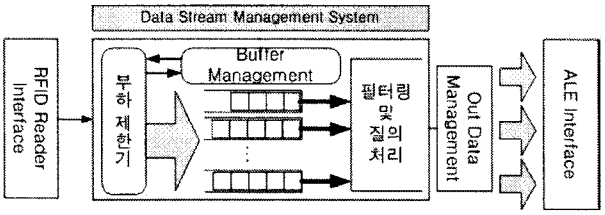
하지만 기존 연구에서는 데이터 유입 시 단일 버퍼를 통해 데이터 저장소로 활용하고 있으며, 데이터 스트림에 의한 부하 발생 시 부하 처리기에서 데이터를 효율적으로 버리는 데에 연구 초점이 맞추어져 있다. 이외에 질의 처리기 성능을 높이는 데 중점을 두고 있다.

본 논문에서는 연속 질의의 결과를 얻기 위해 요구되는 데이터를 임시로 저장하는 저장 공간 관리를 다중 버퍼를 이용하여 관리 한다. 특성이 부여 된 다중 버퍼를 사용함으로써 데이터 스트림의 그룹을 분리하여 질의에 대한 결과 값을 추론할 때 고속으로 결과 값을 얻어낼 수 있도록 하고, 데이터 유입 능력을 극대화 하도록 한다.

3. 고속 데이터 처리 알고리즘

3.1 제안 모델

[그림 3]은 본 논문에서 제안하는 알고리즘이 들어가는 데이터 스트림 관리 모델을 나타낸다.



[그림 3] 제안 모델

데이터 스트림 S가 RFID Reader Interface를 통해서 계속적으로 부하 제한기로 유입 된다. 부하 제한기는 Buffer Management를 통해 관리 되며, Buffer Management는 버퍼 한계 메모리를 초과하는 부하 발생 시 부하 발생 이벤트를 부하 제한기에 보내며, 이벤트를 받은 부하 제한기는 데이터 유입의 조절을 시도한다.

데이터가 입력 되면 버퍼 관리자에서는 부하 제한기에서 받은 현재 데이터 유입 속도에 따라 버퍼의 개수를 결정한다. 즉, 데이터가 많이 입력될 경우 버퍼의 개수를 늘리고, 데이터가 적게 입력될 경우 배퍼의 개수를 줄인다.

동적으로 버퍼의 개수가 결정되면, 입력 된 데이터는 질의 하는 패턴에 의하여 버퍼의 특성이 결정된다. 예를 들어 EPC의 OCcode(Object Class) 가 a< OCcode <= b 를 만족하는 데이터만을 원하는 질의 q가 질의 처리기에 입력 되어 있을 경우 버퍼의 개수에 따라 OCcode의 범위를 나누어 각 버퍼에 Object Class Code의 범위별로 입력 한다. 질의 결과 검출 시 해당 인덱스의 버퍼에서만 데이터를 찾으면 되므로, 빠른 질의 응답을 만들어 낼 수 있다.

본 논문에서는 버퍼 특성 기준에 대하여 EPC를 이진 데이터화 시켜 버퍼의 개수로 나누어 버퍼 특성 기준 값을 도출 하였다. 각 버퍼는 기준 값 범위를 기반으로 특성이 부여 되고 검출 하고자 하는 질의를 넣었을 때, 원하는 데이터를 통해 인덱스를 도출하고 해당하는 버퍼에서만 검색 하여 찾아낼 수 있도록 하였다.

3.2 동적 다중 버퍼 생성 및 삭제

자원의 낭비를 막고 버퍼 사용의 효율성을 얻기 위해 버퍼를 동적으로 운영한다. 버퍼 개수의 Default 는 1개로 시작한다. 동적 다중 버퍼를 관리 하기 위해서 데이터 유입 속도를 알아야 한다. 부하 제한기에서 데이터 유입 속도를 모니터링 하는데, 데이터 유입 속도 S_i 를 계산하는 알고리즘은 식(1) 과 같다.

$$S_i = \frac{\sum_{n=1}^{Sn} S}{t} \quad (1)$$

S_i(Input Data Stream) : 데이터 유입 속도
t(Time) : 일정 시간
S(Data Stream) : 데이터 스트림
S_n(Stream Number) : 데이터 유입 개수

즉, 데이터 유입 속도 S_i는 일정 시간 당 유입되는 데이터 개수를 의미 한다. 얻어진 데이터 유입 속도를 버퍼 증가·감소 이벤트로 하여 일정한 조건에 따라 버퍼를 증가 또는 감소 시킨다. 일정한 조건 상태 값을 구하기 위해서는 데이터 처리 능력을 구해야 하는데 데이터 처리 능력은 식(2)와 같이 계산할 수 있다.

$$D_p = \frac{t}{P_e - P_s} \quad (2)$$

D_p(Data Processing Ability) : 데이터 처리 능력
t(Time) : 일정 시간
P_s(Processing Start) : 프로세스 시작 시간
P_e(Processing End) : 프로세스 종료 시간

식 (2)을 통해서 일정 시간동안 몇 개의 데이터가 처리 되었는지 알 수 있다. 데이터 처리 시간을 산출해서 일정 시간동안 몇 개의 데이터가 처리 되었는지 D_p 를 구하면 버퍼의 증감을 결정하는 조건 값들이 완벽 해진다. 버퍼 증감을 결정하는 조건은 식(3)과 같다.

$$\left.\begin{array}{l} S_i > D_p \ OR \ B_c * B_n < S_i \rightarrow B_n + 1 \\ S_i < D_p \ OR \ B_c * (B_n - 1) > S_i \rightarrow B_n - 1 \end{array}\right\} \quad (3)$$

B_c(Buffer Capacity) : 버퍼 용량
B_n(Buffer Number) : 현재 버퍼 개수

식 (3)에서 보면 버퍼가 증가 되는 조건은 데이터 처리 능력보다 데이터 유입량이 많거나 데이터 유입량이 버퍼 총 용량보다 클 경우이다. 반면에 버퍼가 감소하는 조건은 데이터 처리 능력이 데이터 유입량보다 높거나 버퍼 개수가 한 개 감소 했을 경우 버퍼 총 용량이 데이터 유입량보다 클 경우이다.

동적인 다중 버퍼의 생성과 삭제는 위와 같이 이루어지고 버퍼의 생성과 삭제 시 버퍼의 특성이 변경 되어 적용 되어야 한다. 이때 데이터의 집단을 나누는 기준에 따라 특성이 변화된다.

3.3 데이터 집단의 다중 버퍼 입력

데이터 집단을 나누어 버퍼에 입력할 때 데이터 집단을 나누는 기준은 표 1 과 같은 알고리즘을 사용한다.

[표 1] 데이터 집단 구분 기준 값 설정 알고리즘

1. EPC 코드에서 Target이 되는 코드 (Target Code : TC) 를 선택 한다.
   (1) EPC Management Number
   (2) Object Class
   (3) Serial Number
2. 선택한 Target Number가 가질수 있는 Max값을 버퍼의 현재 개수로 나눈다.
   Basic Value = MAX(TC) / Buffer Count
3. 구해진 기준 값을 범위로 버퍼의 처음부터 유입될 수 있는 버퍼의 특성을 정한다.

기준 값을 생성하여 각 버퍼의 특성을 부여하고 데이터 유입시 해당 특성에 맞추어 입력 된다. 버퍼 관리자

는 해당 버퍼의 특성에 맞추어 인덱스 관리를 하며 질의가 들어왔을 때, 범위를 찾아 해당 인덱스의 버퍼에서만 결과를 찾음으로써, 시간 낭비를 줄일 수 있다. 하지만 만약 집중적으로 한쪽 버퍼에만 입력되어서 자원 낭비가 생기게 될 경우가 있는데, 이때는 현재 생성되어 있는 다른 버퍼의 상태를 버퍼 관리자가 모니터링 하면서 버퍼 수용능력을 체크 후 버퍼 자원이 남아 있는 공간이 많은 데 비하여 한 쪽 버퍼에만 몰려 있는 것을 확인하면 기준 값 범위를 조정을 함으로써 버퍼의 특성 부여를 새로 한다. 그럼으로써 자원 낭비를 줄일 수 있다. 예를 들면, 버퍼의 개수가 현재 5개에 최대 데이터 입력 값 MAX(TC)가 10이라고 했을 때, 기준 값 범위는 초반에 2로 검출이 된다. 버퍼의 용량은 [표 2]와 같이 되고, 이 상태에서 5와 6 값의 데이터만 들어오게 되어서 버퍼의 부하가 생길 경우 버퍼 관리자에서는 기준 값을 동적

[표 2] 각 버퍼 입력 데이터 허용치

버퍼	버퍼1	버퍼2	버퍼3	버퍼4	버퍼5
허용 값 범위	1~2	3~4	5~6	7~8	9~10

으로 변경 하게 된다. 5와 6에 대한 데이터를 5~6 의 간격을 버퍼 카운터로 나누게 된다. 결과 값 Base Value 는 BaseValue>1의 조건을 만족해야 한다. 즉, 2/5=0.4, 이때, 결과 값이 1보다 작으므로 1보다 커야 한다는 조건을 만족하지 못한다. 결국 2/2=1 이라는 식이 만족하면서 결과 값 1로 5~6 범위의 데이터 버퍼를 다시 생성한다. 이때 버퍼의 용량이 적은 버퍼의 기준 값 범위를 +1 시킨다.

인덱스 값의 검출은 질의에 대한 예상 결과 값 R_e을 기준 값 Base Value 으로 나누어 계산한다. 식 (4)는 인덱스 값을 검출하는 것을 나타낸다.

$$I_v = R_e / B_v \qquad (4)$$

I_v(Index Value) : 인덱스 값
R_e(Expected Result Value) : 예상 결과 값
B_v(Base Value) : 기준 값

4. 시뮬레이션

4.1 시뮬레이터 구현 및 환경

시뮬레이션을 위하여 시뮬레이터를 구현하였는데, 데이터 생성을 RFID 리더를 통해서 데이터 생성을 해야 하지만, 데이터 유입량 조절을 위해서 데이터 Generator를 직접 구현하였다. 구현 환경은 OS는 Windows 2003 이고 개발 킷은 닷넷2.0 FramWork의 C#을 이용하였다. 데이터 Generator는 난수 발생기를 이용하여 생성 된다.

그리고 데이터 Generator는 쓰레드를 이용하여 데이터 생성 속도를 조절할 수 있으며, EPC코드에서 Serial Number의 최대 데이터 수인 2^{36} 값 이하로 1이상의 16진수의 난수 값을 발생시킨다. [그림 4]는 데이터 Generator를 구현한 코드를 나타낸다.

```
// EPC의 Serial Number 데이터를 생성하여 부하제한기로 보냄
public void Data_Generator()
{
    while (true)
    {
        // 데이터 발생량을 조절하기 위한 Sleep 타임
        Thread.Sleep(1);

        // Random 데이터 발생
        Random rd = new Random();
        UInt64 nRnum = rd.Next(Convert.ToUInt64(Math.Pow(2, 36)));

        string strHexRnum = Convert.ToString(nRnum, 16);    // 16진수 변환

        Overflow_Limit(strHexRnum);     // 부하제한기에 데이터 전송
    }
}
```

[그림 4] 데이터 Generator 구현 코드

데이터를 생성하게 되면 10진수로 나타내었을 때 최대 값은 '68719476736'의 데이터가 나타나게 되고 16진수로 나타내었을 때는 '1000000000'을 나타낸다. 즉, 데이터 생성 범위는 0< Gen_Data <1000000000 이 된다.

생성 된 데이터는 부하제한기인 Overflow_Limit()에 유입되고 제안 된 내용의 시뮬레이션이 진행된다. 시뮬레이션에서는 버퍼의 구현을 Queue를 사용하였으며, 각 Queue는 EnQueue()와 DeQueue()를 이용하여 데이터를 입력하고 출력할 수 있다. [표 3]은 시뮬레이션에 사용된 핵심적인 함수를 나타내고 있다.

[표 3] 시뮬레이션 주요 함수 설명

함수명	설 명
Data_Generator()	2^{32}보다 적은 범위에서 난수를 발생함
Overflow_Limit()	부하제한기, 데이터 유입조절과 부하시 데이터 버림
Buffer Manager()	각 Queue를 관리, 기준값(Base Value) 생성 및 동적 큐 생성 및 삭제 관리
Buffer_Init()	초기 큐 자원 설정 및 초기화
Query_Precess()	질의가 등록되 있으며, 큐 인덱스를 통해 결과 값을 찾음

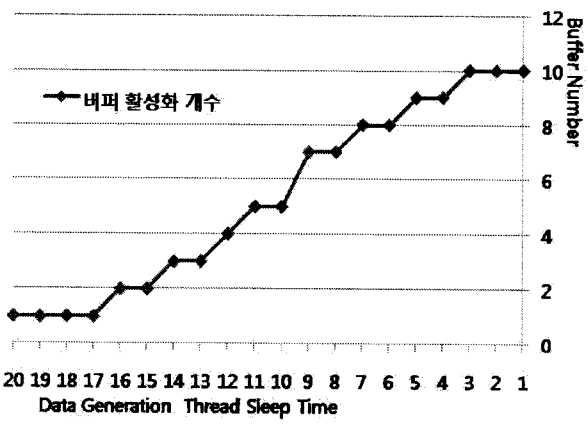
시뮬레이션 최대 버퍼는 10개로 한정 한다.

질의 데이터는 생성 가능한 데이터 범위 안에서 사용자가 임의로 데이터를 지정 하여 입력 한다. 시뮬레이션 수행 작업에 있어서 질의 데이터는 임의로 16진수 '1234567890'을 입력 하였다.

4.2 시뮬레이션 분석

시뮬레이션 시나리오는 다음과 같이 이루어진다. 첫 번째로 데이터 유입량 변화 시 동적 버퍼가 증감 하는 것을 테스트 한다. 두 번 째는 버퍼가 단순히 한 개로 이루어져 FIFO (First Input First Out) 방식으로 입력될 때와 본 논문에서 제안한 다중 버퍼 알고리즘을 이용하였을 때, 데이터 유입속도에 비례한 처리 속도를 살펴본다.

첫 번째로 데이터 유입량 변화는 쓰레드의 슬립타임을 20에서 1까지 1씩 줄여가며 데이터 유입량을 높인다. 그리고 버퍼가 동적으로 활성화 되는 개수를 체크 한다. 이 때, 프로세스의 할당 시간은 20으로 고정 설정한다.

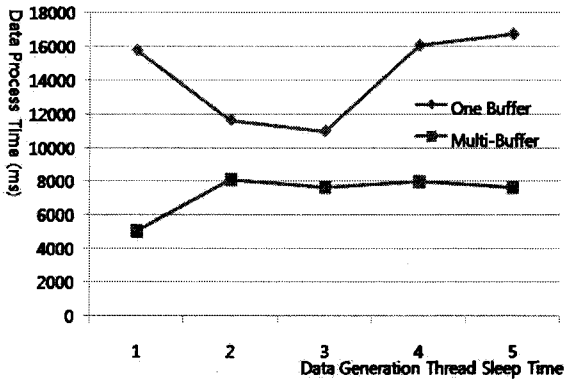


[그림 5] 데이터 유입량에 따른 동적 버퍼 증감 수

[그림 5]는 고정 된 프로세스 타임을 가지고 있을 때 데이터 유입량이 높아질수록 버퍼 개수의 변화량을 나타내고 있다. 쓰레드 타임이 감소한다는 것은 데이터 생성량이 많아진다는 뜻이고 데이터 생성 속도가 빨라질 수록 버퍼가 늘어나는 것을 볼 수 있다.

두 번째 시뮬레이션 설정은 버퍼가 한 개로 구성 되어 있을 때는 버퍼가 수용할 수 있는 데이터 수를 100으로 설정하고 다중버퍼의 경우에는 10개의 버퍼가 생성 되도록 메모리를 설정하고 각 버퍼는 10개의 데이터가 들어갈 수 있도록 설정한다. 그 이유는 같은 자원상태에서 데이터를 측정하기 위해서 이다.(100 * 1 = 10 * 10) 그리고 데이터 생성 시 찾고자 하는 데이터가 빨리 생성될 수 있으므로 확률적으로 데이터를 뽑기 위해서 각각의 상황을 10번씩 테스트 한다. 데이터 발생 속도는 1부터 5까지 테스트 하고 프로세스 수행 시간은 5로 고정 설정한다.

[그림 6]은 단일 버퍼와 다중 버퍼 간의 데이터 처리 시간의 차이를 보여주고 있다. 단일 버퍼 사용할 때 보다 다중 버퍼 사용 시 고속으로 데이터 처리 하는 것을 볼 수 있었다. 평균치를 그래프 화하여 나타냈는데 데이터 쓰레드 타임이 높은 5 값을 가질 수록 즉, 데이터 생성 속도가 낮을 수록 약간 높은 프로세스 시간을 가지는 이유는 원하는 데이터가 난수 발생 시 늦게 발생 되어 나타내는 결과로 유추할 수 있었다. 하지만 다중 버퍼 사용 시 전체적으로 속도가 빨라진 것을 확인할 수 있다.



[그림 6] 단일 버퍼와 다중 버퍼 데이터 처리 시간 비교

5. 결론 및 향후 연구

RFID 기술이 발전하면서 RFID 데이터를 수집하는 데이터 스트림 관리 시스템 연구 분야에 많은 관심이 쏟아지고 있다. 기존 데이터 스트림 관리 시스템 연구 분야에서 부하 시 데이터 처리 부분과 질의 프로세스 관련한 연구 분야가 주로 이루어져 왔다.

본 논문에서는 다중 버퍼를 이용하여 데이터 전송 시 효율적으로 데이터를 처리할 수 있고, 고속으로 데이터를 전송할 수 있는 알고리즘을 제안하였다. 본 연구의 대상 데이터를 RFID 데이터인 EPC 데이터를 사용함으로써 RFID 미들웨어에서 데이터 수집 모듈에 적용 가능함을 보였다. 그리고 다중버퍼에서 동적으로 버퍼의 추가와 삭제가 가능 하도록 제안되었다. 또한 질의 응답 속도를 높이기 위해 각 버퍼에 입력될 수 있는 데이터의

범위를 정하여 질의 결과를 찾을 때 해당 버퍼에서만 데이터를 찾게 함으로써 빠른 결과를 얻을 수 있었다.

향후 연구에는 데이터가 버퍼에 들어와서 정체 현상이 벌어질 때, 버퍼 안에 데이터를 버리거나 삭제하는 알고리즘이 필요하다. 타임 스탬프(Time Stamp)같은 필드를 이용하여 오래된 데이터의 삭제하는 알고리즘이나 우선순위 기반으로 낮은 우선순위 데이터를 버리는 알고리즘에 대해서 연구가 필요하다.

6. Reference

[1] Sean Clark and Ken Traub, etc.,"Auto-ID Savant Specification 1.0," Version of 1 Sep. 2003

[2] 원종호, 이미영, 김명준, "유비쿼터스 컴퓨팅 환경을 위한 RFID 기반 센서 데이터 처리 미들웨어 기술 동향," 전자통신동향분석, 제19권 제5호, pp.21-30, 2004.10

[3] A.arasu, B.babcock, S.Babu, J.McAlister, and J.Widom, "Characterizing Memory Requirements for Queries over Continuous Data Streams," ACM Tras. Database Syst., vol.29, no.1, pp.162-194, 2004.

[4] B.babcock, S.Babu, M.Datar, R.Motwani, and J.Widow, "Model and Issues in Data Stream System," Proc. of ACM PODS, pp.1-16, 2002

[5] J.Gehrke (ed.) "Special Issue on Data Stream Porcessing," IEEE Data Eng. Bull., 2003.

[6] http://www.cs.brown.edu/research/aurora

[7] http://www.cs.wisc.edu/Niagara

[8] http://www-db.stanford.edu/stream

[9] Tag Data Standard Work Group, "EPC^{TM} Tag Data Standards Version 1.1 Rev. 1.24," Standard Specification, Apr. 1, 2004.

[10] 박재석, 조형래, "데이터 스트림에 대한 연속 질의를 위한 우선순위 기반의 의미적 부하 제한," 데이터베이스 연구, 제21권 제1호, pp.73-86, 2005.04

[11] 류연승, 고건, "가변 데이터 율을 갖는 연속미디어 데이터를 위한 동적 버퍼 관리 기법," 정보과학회 논문지, 제25권 제3호, pp. 257-276, 1998.3

[12] 이미영, 김명준, "이벤트 기반 서비스 기술 동향," 전자통신동향분석, 제21권 제5호, pp.61-68, 2006.10