데이터 스트림에서 연속질의 처리를 위한 큐 메모리 관리 기법

신재완 O, 신숭선 O, 이동욱 O, 김경배 OO, 배해영 O

O 인하대학교 컴퓨터 정보 공학과, OO 서원대학교 컴퓨터교육과

{nut0216, hermit, dwlee}@dblab.inha.ac.kr, gbkim@seowon.ac.kr, hybae@inha.ac.kr

Queue Memory Management Method for Continuous Query Processing in Data Stream

Jae-Wan Shin O, Soong-Sun Shin O, Dong-Wook Lee O, Kyung-Bae Kim OO, Hae-Young Bae O

O Dept. of Computer Science and Information Engineering, Inha University

OO Dept. of Computer Education, Korea Seowon University

요 약

연속적이고 무한히 발생되는 데이터 스트림을 관리하는 데이터 스트림 관리시스템(DSMS)은 연속질의를 이용하여 스트림을 처리한다. 연속질의는 질의 별로 독립적인 큐를 유지하기 때문에 질의 개수가 증가함에 따라서 메모리 비용이 증가되며, 잦은 메모리 할당으로 인한 시스템의 성능 저하를 갖는다. 이러한 문제점을 해결하기 위한 기존의 연구로 메모리 풀을 이용한 메모리 관리 기법이 있다. 하지만 페이지의 크기가 고정되어 있기 때문에 각 질의마다 필요로 하는 데이터 스트림의 최적의 크기에 적합하게 할당되지 못하여 메모리를 낭비하는 문제점이 있다. 본 논문에서는 이러한 문제를 해결하기 위해 연속질의 처리를 위한 큐 메모리 관리 기법을 제안한다. 제안기법은 큐 관리 테이블에서 관리하는 각각의 큐 메모리들을 타임스탬프를 가지고 일정한 기간을 주기로 큐 메모리의 사용량을 분석한다. 분석된 큐 메모리들은 이전의 큐 메모리의 사용량과 현재 사용된 큐 메모리의 사용량을 비교함으로써 상한 값과 하한 값을 구함으로써 현재 큐 메모리에서 가지고 있는 사용량을 추가할 것인지, 줄일 것인지를 판단하여, 메모리의 사용량을 최적화 함으로써 시스템의 메모리 가용성을 향상한다. 제안 기법은 성능평가를 통해 메모리의 가용성이 기존의 방식에 비하여 향상된 성능을 보인다.

1. 서론

최근 고정된 저장 데이터가 아닌, 잠재적으로 무한하고 연속적인 입력에 초점을 맞춘 데이터 스트림 관리 시스템 개발에 관한 연구가 진행되고 있다. 데이터 스트림 관리 시스템에서 대부분의 질의는 연속질의 형태로 시스템에 등록된다. 입력되는 스트림 데이터는 그 크기가 잠재적으로 무한하므로 모두 저장되지 못하고, 한정된 메모리상에서 매우 빠르게 처리되며 그 결과는 사용자나 응용에 다시 스트림의 형태로 전달된다[1-3].

연속질의가 등록 되었을 경우 큐가 생성되며, 각각의 큐를 통하여 데이터를 처리한다. 연속질의는 이처럼 독립적인 큐를 유지하기 때문에 질의의 개수가 증가함에 따라서 메모리 비용이 증가하고, 메모리 할당으로 인한 작업부하가 발생한다. 또한 조인(Join)과 카티션프로덕트(Cartesian product)와 같은 많은 메모리 공간을 요구하는 연산이 수행됨에 따라서 메모리의 효율적인 관리가 필요하다[4]. 이러한 문제점을 해결하기 위한 기존의 연구로 메모리 풀을 이용한 메모리 관리 기법이 있다[7]. 메모리 풀 기법은 일정 공간의 메모리를 동적으로 할당 받아서 임의의 크기로 페이지를 분할 할여 각각을 관리하는 기법이다. 하지만 페이지의 크기가 고정되어 있기 때문에 각 질의마다 필요로 하는 데이터 스트림의 최적의 크기에 적합하게 할당되지 못하여 메모리를 낭비하는 문제점이 있다.

본 논문에서는 데이터 스트림 환경에서의 연속질의 처리를 위한 큐 메모리 관리 기법을 제안한다. 제안 기법은 큐 메모리 관리를 위해 큐 테이블을 이용하여 각각의 큐들을 관리한다. 큐를 분석하여 필요한 메모리 공간의 크기를 결정하기 위하여 큐는 큐 메모리 이용률, 메모리 크기, 현재 큐 사용량 등의 정보 구조를 갖는다.

본 연구는 건설교통부 첨단도시기술개발사업 - 지능형국토정보기술혁신사업과제의 연구비지원(07국토정보C05)에 의해 수행되었습니다.

이러한 구조를 통하여 일정한 주기를 기준으로 계속적으로 현재의 큐 사용량과 이전의 큐 사용량을 획득하여 상한 값과 하한 값을 구한다. 상한 값과 하한 값을 비교함으로써 현재 큐에서 가지고 있는 메모리의 용량을 추가할 것인지, 줄일 것인지를 판단하여, 메모리의 사용량을 최적화함으로써 시스템의 메모리 가용성을 향상한다.

제안 기법은 성능평가에서 큐 메모리 관리 기법을 통한 불필요한 메모리 할당으로 인하여 기존의 방식에 비하여 전체적인 메모리 공간의 효율성과 이용율의 높은 향상을 보인다.

본 논문의 구성은 다음과 같다. 2절에서는 데이터 스트림 환경에서의 연속질의 처리 과정과 연속질의 처리를 위한 기존의 메모리 관리 기법을 설명한다. 3절에서는 본 논문에서 제안하는 연속질의 처리를 위한 큐 메모리 관리 구조 기법을 보인다. 4절에서는 제안한 기법의 성능 측정 결과를 평가하고, 마지막으로 5절에서는 본 논문의 결론 및 향후 연구를 보인다.

2. 관련 연구

2.1 데이터 스트림에서의 연속 질의 처리

기존의 데이터베이스 관리시스템은 SQL(Standard Query Language)질의를 사용하여 영속적인 데이터를 처리한다. 그러나 데이터 스트림은 시간에 따라 값이 변하고 연속적으로 생성되는 대용량의 데이터이기 때문에, 기존의 SQL로 처리하는 것은 적합하지 않다. 그래서 데이터 스트림 모델에 적합한 새로운 형태의 질의인 연속질의(CQL: Continuous Query Language)가 제안 되었다. 데이터 스트림에서 연속질의 처리는 큰 관심을 받고 있으며, 데이터 스트림과 관련된 대표적인 연구들로 STREAM과 Aurora가 있다[2-3].

연속질의는 끊임없이 삽입되는 대용량의 데이터 스트림을 처리하기 위해 윈도우를 이용하며, 이것을 통하여 데이터를 처리 할 수 있는 일정한 데이터를 획득한다. 윈도우는 크게 시간기반 윈도우와 튜플기반 윈도우로 나누어 진다. 시간 기반 윈도우는 일정 시간을 기준으로 삽입된 데이터를 획득하는 윈도우이며, 튜플기반 윈도우는 일정한 공간에 삽입 되어진 튜플의 크기를 기반으로 데이터를 획득하는 윈도우 이다. 이런 각각의 윈도우에서 획득한 결과는 큐에 저장되며 시스템에 등록된 연속질의를 처리를 위하여 사용된다. 또한 처리된 결과는 사용자에게 전달될 수 있으며, 다른 연산을 위한 입력 값으로 이용된다[6].

2.2 연속질의에서의 메모리 관리 기법

연속질의에서의 메모리 할당 기법으로 각 질의들이 메모리를 요구할 때 마다 필요한 크기만큼 동적 할당을 해주는 동적 할당을 이용한 큐 관리 기법이 있다. 이 기법은 빠르게 입력되는 데이터 스트림의 질의 결과를 저장하기 위해서 메모리를 요구할 때 마다 필요한 공간을 동적 할당하고 불필요한 메모리 공간은 해제하는 메모리 관리 기법이다. 필요한 크기의 메모리만을 동적 할당해 주기 때문에 사용하지 않는 공간을 최소화 할 수 있는 최적의 관리 기법이다. 하지만 동적 메모리 할당은 많은 위험이 발생한다. 동적 할당은 힙 공간을 사용하는데 힙 공간의 부족으로 메모리 할당을 실패하거나 메모리 할당 연산 실패로 메모리 할당을 하지 못하는 경우 심각한 위험성을 가져오게 된다. 또한 동적 메모리 할당 연산은 많은 연산 비용을 요구하기 때문에 작업 부하가 발생하여 스트림 데이터 처리에 영향을 주게 된다. 이러한 문제점을 해결하기 위해 제안된 기법이 미리 동적 할당을 하여 공동 메모리 풀을 유지한 후 메모리 요구에 따라 분배하여 주는 메모리 풀 관리 기법이다. 메모리 풀 관리 기법은 미리 동적 할당을 수행하여 메모리 풀을 유지한 후 할당을 해주는 큐 관리 기법이다. 이 기법은 메모리 공간을 공동으로 사용, 관리할 수 있는 공동 메모리 풀을 이용하여 미리 필요한 크기의 공간을 할당하며 큐의 공간이 부족한 경우 페이지 단위로 할당된다. 메모리 공간이 부족할 때 동적 할당 연산의 수행 없이 필요한 크기의 메모리를 할당 받을 수 있으며, 동적 할당을 수행할 때 발생하는 위험성과 작업 부하를 줄일 수 있는 이점이 있다[7]. 하지만 공동 메모리 풀을 유지하여 메모리를 관리 하는 방법은 연산자들의 질의 결과 크기의 급속한 변화와 새로운 질의가 추가 또는 삭제되어 연산자 수가 급속히 변하는 경우 메모리 풀의 작업 부하가 발생하는 문제점이 발생한다. 이러한 작업 부하를 감소시키기 위해서는 각 질의에서 요구하는 메모리 공간의 크기를 미리 예측하여 할당하기 위한 기법과 큐를 관리 필요하다.

3. 연속질의 처리를 위한 큐 메모리 관리 기법

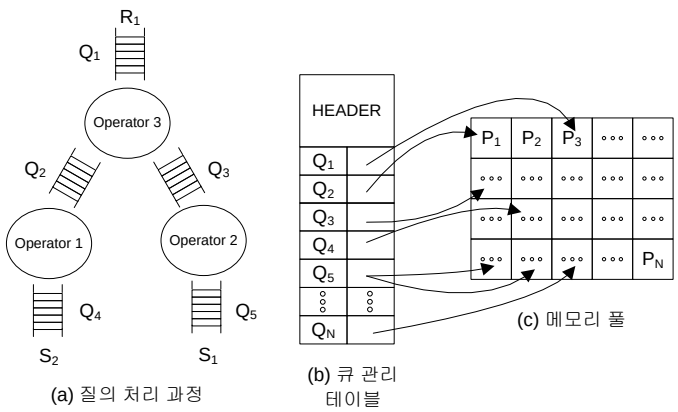
본 장에서는 데이터 스트림 환경에서 연속질의를 위한 큐 메모리 관리 기법을 설명한다. 먼저 제안기법을 위한 관리 구조를 설명한 후, 큐 메모리 관리를 위하여 사용하는 알고리즘을 설명한다.

3.1 확장된 큐 테이블을 이용한 연속질의 처리 구조

[그림 1]은 하나의 질의가 삽입되었을 경우 확장된 큐 테이블을 이용한 연속질의 처리의 구조에 대하여 나타낸다.

[그림 1]의 (a)는 하나의 연속질의가 들어 왔을 경우의 질의 처리 과정을 나타낸다. S_1, S_2 각각의 윈도우를 통해 들어오는 스트림 데이터이다. Operator1,

Operator2 는 연산자를 의미하며 스트림 처리에서 이용되는 다양한 연산자가 사용될 수 있다.



[그림 1] 연속질의 처리를 위한 메모리 할당

Q_1, Q_2, Q_3, Q_4, Q_5 는 하나의 질의 결과를 얻기 위한 각각의 큐를 나타낸다. 이 큐를 통하여 연속질의의 연산을 처리한다. R_1 은 하나의 연속질의로부터 출력되는 최종 결과 값이다. 결과 값은 시스템 설정에 따라서 사용자에게 전달 되거나, 디스크에 저장될 수 있다.

(b)는 큐들을 관리하는 테이블이다. 큐 관리 테이블은 하나의 연속 질의가 입력 되었을 경우 발생되는 큐들을 묶어서 관리하는 역할을 한다. 이는 질의가 철회 되었을 경우 질의에 연관된 모든 큐들을 반환하여 메모리의 효율성을 높인다.

(c)는 시스템에서 가지고 있는 일반적인 메모리 할당 공간을 나타낸다. 일반적인 메모리 할당 공간을 메모리 풀을 사용하여 페이지(Page) 형식으로 관리함으로써 보다 효율적으로 메모리를 관리할 수 있다.

하나의 연속질의가 입력되었을 경우 질의 처리를 위한 각각의 연산자와 연산이 필요한 데이터를 저장하고 있는 큐들이 생성된다. 각각의 큐들은 큐 테이블을 통하여 관리되며, 큐에 있는 정보를 통하여 이전의 큐 사용량과 현재의 큐 사용량을 비교할 수 있다.

[그림 2]는 큐 사용량을 비교하기 위하여 정의된 큐의 구조 이다.

알고리즘 수행을 위해 필요한 큐의 메모리 정보들을 유지하기 위한 관리 구조이다.

MAX	MIN	QueSize	UseSize	Log

[그림 2] 메모리 할당을 위한 큐 관리 구조

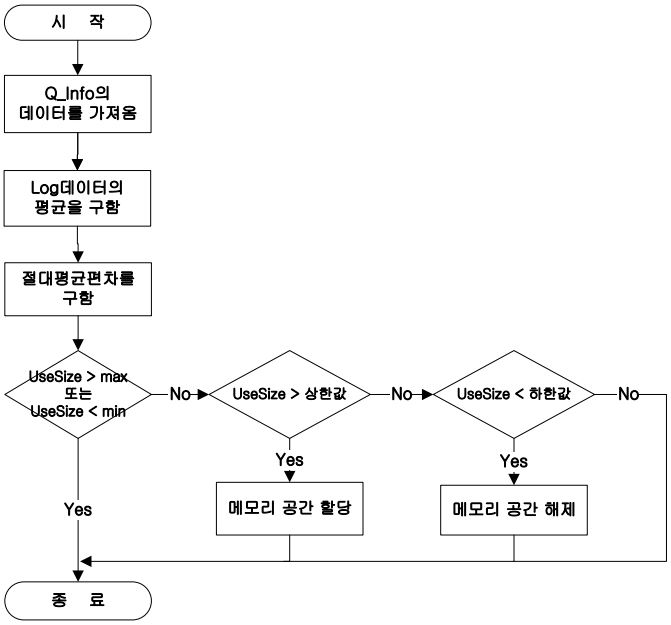
[그림 2]는 큐 메모리 관리 테이블에 저장된 각각의 큐 관리 구조이다. 각각에 대한 설명은 다음과 같다.

MAX는 최대로 사용될 수 있는 큐 메모리의 사용량이며, MIN은 최소로 사용되는 큐 메모리의 사용량이다. QueSize는 현재 사용되고 있는 큐 메모리의 크기이며, UseSize는 현재 사용되는 메모리의 사용량이다. 또한 Log는 기존에 사용되었던 메모리 사용량을 저장한 값이다.

다음 절에서는 이러한 큐 메모리 관리 구조를 이용한 큐 메모리 관리 기법에 대하여 나타낸다.

3.2 큐 메모리 관리 기법

[그림 3]은 메모리 할당 기준 값을 이용하여 메모리를 할당하는 알고리즘이다.



[그림 3] 메모리 할당 순서도

[그림 3]의 순서도는 UseSize를 기준으로 모든 큐에 대하여 메모리 할당 및 해제를 시도한다. 큐의 메모리 할당은 큐의 관리 정보 분석을 통하여 이루어진다. 우선 큐의 Log 데이터를 읽고 평균과 절대평균편차를 구한다. 이렇게 구한 기준 값은 현재 사용되는 메모리 크기(Usesize)와 비교하여 상한 값보다 큰 경우 메모리 공간 할당을 수행하고, 하한 값 보다 작은 경우 메모리 해제 작업이 이루어 진다. 또한 UseSize는 최대값인 max를 넘지 못하고, 최소값인 min보다 작아지지 않는다. 이때 사용되는 상한 값과 하한 값은 제안된 수식을 이용하여 계산된다.

큐의 메모리 사용량은 입력되는 데이터 스트림의 종류와 연속질의에 입력된 연산자의 종류에 따라 다양한 크기를 갖는다. 이렇게 다양한 크기를 갖는 메모리 사용량을 분석하기 위해 매초 마다 기존에 사용되었던 메모리 사용량을 Log에 기록한다. 메모리 할당 및 해제를 예측하기 위해서는 메모리 사용량의

변화를 분석해야 한다. 메모리 사용량 변화의 분석은 Log에 기록된 메모리 사용량의 변화율을 계산하여 값을 얻고, 이렇게 계산된 값은 메모리 할당 및 해제를 위한 기준에 사용된다. 메모리 사용량의 변화는 절대평균편차를 이용하여 구할 수 있다. 절대평균편차는 데이터의 평균값을 중심으로 퍼진 정도의 평균값을 나타내며, 이를 계산하기 위한 방법을 [수식 1]과 [수식 2]를 통해 설명한다.

[수식 1]은 절대평균편차의 계산의 위해 사용되며, log 데이터를 이용 평균값을 구한다. [수식 1]의 $Q_1, Q_2, \ldots, Q_n$은 n개의 큐, $S_{n1}, S_{n2}, \ldots, S_{ni}$은 Q_n에서 사용되었던 메모리 사용량, S_{ni}의 i는 log에 기록된 메모리 사용량 정보의 개수라고 할 때 큐에서 사용되는 메모리 사용량의 평균 값을 구할 수 있다.

$$\text{평균}(Q_n) = \frac{1}{i}\sum_{k=1}^{i} S_{ni} = (S_{n1} + S_{n2} + \ldots + S_{nj})/i \quad \text{[수식 1]}$$

[수식 2]는 절대평균편차를 구하는 식으로 $S_1, S_2, \ldots, S_n$은 큐의 메모리 사용량, $\overline{q}$은 큐에서 사용되는 메모리 사용량의 평균 이라고 할 때, 메모리 사용량의 절대표준편차를 구할 수 있다.

$$\text{절대평균편차} = \frac{1}{n}\sum_{i=1}^{n} |S_i - \overline{q}| \quad \text{[수식 2]}$$

메모리 할당은 메모리 사용량의 평균값과 절대평균편차를 합한 값을 기준으로 이루어지고 이를 상한 값이라고 부른다. 메모리 해제는 메모리 사용량의 평균값과 절대평균편차를 제거한 기준으로 이루어지고 이를 상한 값이라고 부른다.

4. 성능 평가

본 장에서는 제안 기법의 평가를 위한 실험 환경에 대해 설명하고, 입력되는 스트림의 크기, 연속질의에 등록된 연산자의 종류 등의 환경 변화에 따른 제안 기법과 기존 기법의 성능 비교를 수행한다.

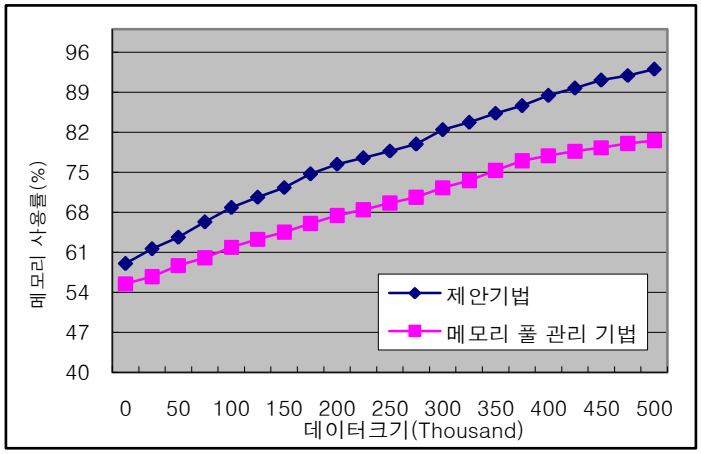
4.1 실험환경

실험평가에서 사용된 시스템 환경은 CPU가 Pentium 4 3.0 GHz, 메모리는 2GB, 운영체제는 WINDOW XP를 사용하였다. 실험에 사용된 데이터는 스트림의 입력 속도를 증가 시키기 위해 최대 초당 500,000개의 튜플을 인위적으로 생성할 수 있는 프로세스에 의해 생성되는 데이터이다. 튜플은 다양한 환경에서의 실험을 위해 고정길이 데이터와 가변길이 데이터를 모두 포함하였고, 80 Byte ~ 1024 Byte 크기의 데이터를 랜덤 생성 하였다. 또한 연산자의 종류에 따른 변화를 측정하기 위해 여러 개의 연속질의에 다양한 연산자를 포함한 실험을 하였다.

4.2 성능평가

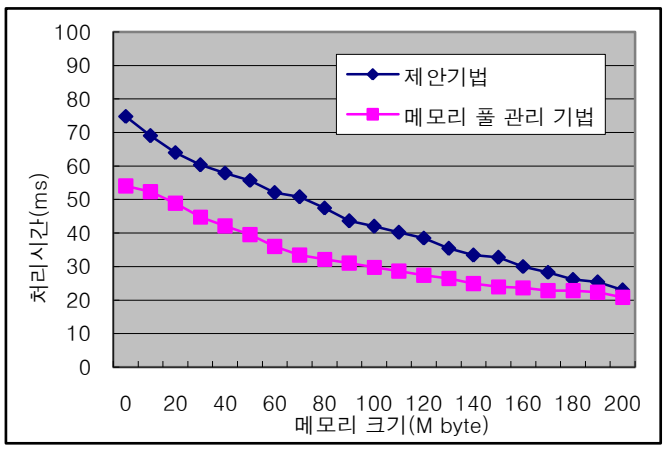
본 실험 평가에서는 입력되는 스트림의 크기에 따른 메모리 사용 비율과 전체 메모리 크기에 따른 메모리 사용 비율을 주요 평가 인자로 하였다.

연속질의는 데이터 입력과 동시에 질의 처리를 위한 메모리 공간을 할당한다. 이렇게 할당된 메모리 공간은 데이터를 처리하는데 이용된다. [그림 4]는 데이터 크기 증가에 따른 메모리 사용률을 비교한다.



[그림 4] 데이터 크기 증가에 따른 메모리 사용률

메모리 사용률 비교를 위한 실험결과는 기존 기법인 메모리 풀 관리 기법은 데이터 크기와 상관없이 평균적인 메모리 이용율을 보인 반면, 제안 기법은 데이터의 크기가 증가함에 따라 메모리 사용률이 증가하였다. 제안기법의 메모리 사용률 증가는 데이터의 크기가 늘어남에 따라 처리해야 하는 데이터 양이 커지고, 이를 처리하기 위해 많은 공간의 메모리를 요구함에 따라 제안 기법의 메모리 사용률이 높아진다.



[그림 5] 전체 메모리 크기 증가에 따른 처리시간

[그림 5]는 연속질의 처리에 사용되는 전체 메모리 공간의 크기를 달리 하면서 메모리 공간 크기 증가에 따른 처리시간의 변화를 비교하였다.

전체 메모리 크기 증가에 따른 데이터 처리시간 비교는 한정된 메모리 공간은 얼마나 효율적으로 활용하여 데이터 처리를 빠르게 하는가를 비교한다. 실험결과 적은 메모리 공간에서는 기존의 메모리 풀 관리 기법보다 제안 기법이 빠른 처리시간을 보인다. 하지만 전체 메모리 공간이 증가함에 따라 비슷한 처리시간을 보였다. 이는 메모리 공간이 충분한 환경에서는 제안 기법의 효율성이 기존 기법과 비슷함을 보인다. 실험결과 제안기법은 적은 메모리 공간에서 많은 데이터를 처리해야 하는 환경에서 좋은 성능으로 보였다.

5. 결론 및 향후 연구

본 논문에서는 데이터 스트림 환경에서 메모리의 가용성을 향상 시키기 위하여 연속질의 처리를 위한 큐 메모리 관리 기법을 제안하였다. 제안된 알고리즘은 큐 테이블을 사용함으로써 각각의 큐 메모리를 타임스탬프 기반으로 관리하여 메모리의 사용량을 얻으며, 이를 통하여 메모리의 상한 값과 하한 값을 구한다. 상한 값과 하한 값의 비교를 통하여 큐 메모리의 사용량을 증가하거나 감소하여 메모리의 효율성을 향상하였다. 또한 성능 평가를 통하여 제안 기법이 기존의 기법에 비하여 메모리의 가용성 측면에서 성능이 향상됨을 보였다.

제안 기법은 현재 사용되는 많은 데이터 스트림 관리 시스템이 사용되는 모든 환경에 적용 가능할 것으로 판단되며, 현재 많은 연구가 진행되고 있는 데이터 스트림 마이닝의 연구에도 사용가능 할 것으로 예상된다.

향후 연구로는 제안된 알고리즘에서 임의의 타임스탬프 기간이 아닌 최적의 타임스탬프를 찾아내는 기법이 필요하다.

참고문헌

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. “Models and issues in data stream systems,” In Proceedings of PODS, 2002.

[2] 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, 2003.

[3] A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom, “STREAM: The Stanford Stream Data Manager,” Proceedings of the 2003 ACM SIGMOD International conference on Management of Data, p. 665, 2003.

[4] B. Neil, and H. John, “The Dynamics of Changing Dynamic Memory Allocation in a Large-Scale C++ Application,” In Proc. ACM Sym. p. 1-2. 2006.

[5] P. Bonnet, J. Gehke, and p. Seshadri, “Towards Sensor Database Systems,” In Proc. 2th Int. Conf. on Mobile Data Management, pp. 3-14, 2001.

[6] B Babcock, S. Babu, M. Data, R. Motwani, “Chain: Operator Scheduling for Memory Minimization in Data Stream System,” In Proc. ACM Sym. P. 1-10. 2003.

[7] A. Arasu, S. Babu. Shivnath, and J. Widom, “The CQL continuous query language: semantic foundations and query execution,” Springer-Verlag, p. 134-138. 2005.