데이터 스트림 환경에서 질의 처리 성능 향상을 위한 링 연산자 쓰레드 스케줄링 기법

김영기*, 신숭선*, 백성하*, 이동욱*, 정원일**, 배해영*
*인하대학교 컴퓨터정보공학과
**호서대학교 정보보호학과
e-mail : {ykkim,ssshin,shbaek,dwlee}@dblab.inha.ac.kr, wnchung@hoseo.edu, hybae@inha.ac.kr

Ring Operator Threaded-Scheduling for Improving Continuous Query Processing over Data Streams

Young-Ki Kim*,Soong-Sun Shin*,Sung-Ha Baek*,Dong-Wook Lee*,
Weon-Il Chung**, Hae-Young Bae*
*Dept. of Computer Science and Information Engineering, Inha University
**Dept. of Information Security Engineering, Hoseo University

요 약

최근 데이터 스트림을 관리하기 위해 DSMS(Data Stream Management System)가 계속적으로 연구되고 있다. 하지만 데이터 스트림은 방대한 양의 데이터를 처리하기 위하여 실시간성을 갖는 빠른 데이터 처리가 요구되며, 이러한 특성 때문에 데이터 처리의 효율성 증대를 위하여 메모리 관리가 중요하다. 기존 메모리 관리에 대한 연구는 쓰레드 스케줄링을 통한 관리 기법이 연구되었다. 하지만 기존의 연산자 쓰레드 스케줄링과 그래프 쓰레드 스케줄링은 쓰레드 관리의 유연성이 떨어지기 때문에 불규칙적인 데이터 스트림 환경에서는 부적합하다. 또한 하이브리드 멀티 쓰레드 스케줄링 기법은 두 가지 기법의 장점을 결합하였으나 가상 연산자의 일회성 사용으로 쓰레드 관리의 어려움이 있다. 따라서 본 논문은 공유와 재사용이 가능한 링 연산자를 이용한 링 연산자 쓰레드 스케줄링 기법을 제안한다. 본 기법은 링 연산자를 통해 새로 입력된 질의의 동일한 연산자 구성은 생성되어 있는 링 연산자를 공유하거나 재사용하여 불필요한 자원 소모와 자원 할당의 과부하를 줄임으로써 기존 기법에 비해 쓰레드 할당의 수를 감소 시켜 대량의 질의 처리 시 속도를 증가 시켰다.

1. 서론

최근 정보통신기술의 발달로 인해 이 시간에도 끊임 없이 발생하는 다양한 데이터가 여러 종류의 어플리케이션을 통해 사람의 생활 속에 많은 정보를 전달하고 있다. 이렇게 끊임 없이 발생하는 데이터를 데이터 스트림이라 하며 데이터 스트림을 관리하기 위해 DSMS (Data Stream Management System)가 개발되고 있다[1-3]. 하지만 데이터 스트림은 데이터 전송이 불규칙적이고 크기가 무한하며 전송율 또한 급변하는 특징이 있으며 방대한 양의 데이터를 처리하기 위하여 실시간성을 갖는 빠른 데이터 처리가 요구된다. 이러한 특성 때문에 데이터 처리의 효율성 증대를 위하여 현재 DSMS 상에서의 자원 관리를 위한 여러 기법이 연구 되며, 이러한 연구 중에서 쓰레드 스케줄링을 통한 자원 관리 기법이 계속적으로 연구 진행 중이다. 기존의 대표적인 자원 관리 연구로는 연산자 쓰레드 스케줄링 기법(Operator-Threaded Scheduling: OTS)과 그래프 쓰레드 스케줄링 기법(Graph-Threaded Scheduling: GTS)이 있다.

기존의 OTS 과 GTS 는 대표적인 쓰레드 스케줄링 기법이다[4]. 하지만 OTS 는 각 연산자에 쓰레드를 할당하기 때문에 많은 양의 연산을 처리하기에 부적합하고 GTS 는 하나의 쓰레드로 하나의 쿼리 그래프(Query Graph)를 처리하기 때문에 고비용의 연산자가 존재하게 되면 처리 속도가 지연되는 문제가 있다. 또한 두 가지 기법의 장점을 결합하기 위해 가상 연산자(Virtual Operator)의 개념을 이용하여 연산자를 파티션으로 분할 처리하는 하이브리드 멀티 쓰레드 스케줄링(Hybrid Multi-Threaded Scheduling: HMTS) 기법이 제안되었지만 많은 질의가 등록되고 삭제되는 데이터 스트림 환경에서의 파티션의 일회성은 빈번한 쓰레드의 생성과 삭제를 일으키기 때문에 자원 관리의 어려움이 있다[5].

본 논문에서는 앞 서 말한 GTS, OTS 의 연산자의 비용에 의존적인 자원 관리의 비효율성 문제와 HMTS 의 빈번한 자원 할당 및 해제 문제를 해결하기 위해 링 연산자(Ring Operator)를 이용한 링 연산자 쓰레드 스케줄링(Ring Operator-Threaded Scheduling: ROTS)기법을 제안한다. 링 연산자는 HMTS 의 가상

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

연산자에서 착안하여 여러 연산자를 원형으로 연결하여 반복 처리 가능하게 하였으며 앞 서 말한 HMTS의 문제를 해결하기 위해 재사용 및 공유가 가능 하도록 하였다. 또한 여러 연산자를 결합한 링 연산자에 쓰레드를 할당하여 기존의 OTS 와 GTS 의 문제점을 해결하고 HMTS 에 비해 쓰레드 수를 감소 시켜 질의 처리 성능을 높였다.

본 논문의 구성은 2 장에서 기존에 제안되었던 쓰레드 스케쥴링 기법을 설명하고 3 장에서는 ROTS 에 대하여 설명한다. 또한 4 장에서는 ROTS 에 대한 성능평가와 결과를 보이고 마지막으로 5 장에서 향후 연구 방향과 결론으로 끝을 맺는다.

2. 관련 연구

2. 1. 그래프 쓰레드 스케줄링 기법(Graph-Threaded Scheduling: GTS)

쿼리 그래프(Query Graph) 상의 연산에 대한 쓰레드 스케줄링의 한 방법으로 그래프 쓰레드 스케줄링 기법(Graph-Threaded Scheduling: GTS) 방식이 있다[5]. GTS 는 하나의 쿼리 그래프를 한 개의 쓰레드가 처리하는 방식이다. GTS 는 Source(e.g. RFID 리더, 센서, 모바일 기기 등)로부터 발생된 데이터를 하나의 쓰레드로 운영되는 쿼리 그래프 상의 해당질의의 연산자들에 의해 처리하여 Sink(e.g. 데이터베이스, 모바일 기기, PC 등)로 전달한다. 만약 쿼리 그래프 상에 처리 비용이 높은 연산자가 존재하지 않는다면 GTS 는 높은 효율을 보일 수 있다. 하지만 처리 비용이 높은 연산자는 하나의 데이터를 처리 하기 위해 많은 시간을 지연시킨다. 그러므로 하나의 쓰레드로 구성된 쿼리 그래프상에 고 비용의 연산자가 존재한다면 데이터 스트림의 특성상 입력 큐에 과부하가 일어날 수 있다. 따라서 데이터 스트림 환경에서 GTS 방식으로 쓰레드를 스케줄링 한다면 고비용 연산자에 의한 문제가 발생할 수 있기 때문에 적합하지 않다[5,6].

2. 2. 연산자 쓰레드 스케줄링 기법(Operator-Threaded Scheduling)

연산자 쓰레드 스케줄링 기법(Operator-Threaded Scheduling: OTS)는 GTS 와 달리 연산자에 쓰레드를 할당하는 쓰레드 스케줄링이다[10]. 연산자 쓰레드는 입력 큐(Input Queue)로부터 데이터를 받아와서 연산을 수행하고 그 결과를 출력 큐(Output Queue)에 저장한다. 멀티 프로세서 시스템은 다중 쓰레드를 동시에 제어가 가능 하기 때문에 OTS 는 멀티 프로세서 기반에서 이점을 갖는다. 하지만 OTS 는 각 연산자를 해당 하는 각각의 쓰레드로 수행하기 때문에 쿼리 그래프 내에 저비용의 연산자가 많을 경우 각 연산자 마다 쓰레드를 할당하여 자원을 낭비하게 된다. 따라서 OTS 를 사용하여 스케줄링 한다면 저비용 연산자가 많을수록 자원 낭비를 초래하게 되므로 데이터 스트림을 처리하는 DSMS 와 같은 시스템에는 부적합하다[5,6].

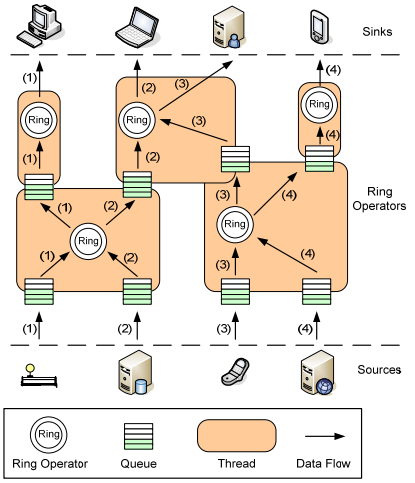
2. 3. 하이브리드 멀티 쓰레드 스케줄링 기법(Hybrid Multi-Threaded Scheduling: HMTS)

앞 서 소개한 GTS 와 OTS 의 장점을 결합하고 단점을 보완하기 위해 하이브리드 멀티 쓰레드 스케줄링(Hybrid Multi-Threaded Scheduling: HMTS)이 제안되었다[5]. HMTS 는 상황에 따라 GTS 또는 OTS 로 변경하여 쓰레드를 스케줄링 하기 위해 설계되었다. 이를 위해 HMTS 는 쿼리 그래프를 파티션(Partition)으로 분할한다. 파티션은 연산자들의 비용을 계산하여 분할하며 분할된 파티션은 하나의 가상 연산자로 통합되어 처리한다. 그렇기 때문에 기존에 연산자와 연산자의 사이에 존재하던 큐가 제거되어 메모리 소모를 감소시켰다. 또한 파티션 단위로 쓰레드 배정을 하기 때문에 GTS 와 OTS 의 장점을 수용하여 쓰레드 관리가 가능하다. 하지만 많은 양의 질의의 등록, 수정, 삭제가 발생하면 HMTS 는 일회성의 파티션 단위로 쓰레드를 할당하기 때문에 해당 파티션 쓰레드가 빈번하게 생성되고 삭제되어 자원 관리상 과부하가 발생한다. 따라서 대량의 데이터 스트림이 발생하고 많은 질의를 처리해야 하는 DSMS 에서는 자원 관리가 어렵다.

3. 링 연산자 쓰레드 스케줄링 기법 (Ring Operator-Threaded Scheduling: ROTS)

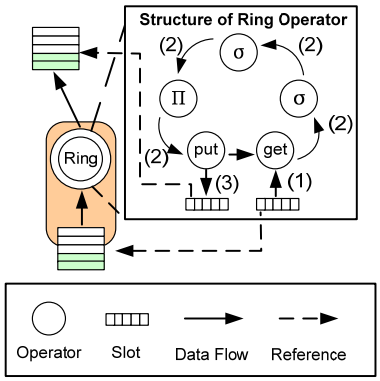
3. 1. ROTS 를 이용한 질의 처리 구조

링 연산자 쓰레드 스케줄링 기법(Ring Operator-Threaded Scheduling: ROTS)의 주요 요소는 링 연산자(Ring Operator)와 링 풀(Ring Pool), 링 관리자(Ring Manager)로 구성된다. 링 연산자는 질의 내의 연산자를 링 형태로 묶은 HMTS 의 가상 연산자(Virtual Operator)의 개념과 유사한 가상 연산자이다. 링 풀은 링 연산자를 미리 생성 해놓은 메모리 공간이며 링 관리자는 링 연산자와 링 풀을 관리한다.



(그림 1) ROTS 를 사용한 쿼리 그래프

(그림 1)은 스트림 환경에서 질의가 처리되는 과정을 ROTS 를 사용하여 나타낸 그림이다. 하단에는 데이터 스트림이 발생하는 데이터 소스(Source)가 위치해있으며 중간에는 연산자(Operator)들을 묶은 하나 이상의 링 연산자(Ring Operator)가 연결되어 있다. 또한 상단에는 질의 처리된 데이터가 전송될 목적지(Sink)가 위치한다.



(그림 2) 링 연산자의 구조와 수행 과정

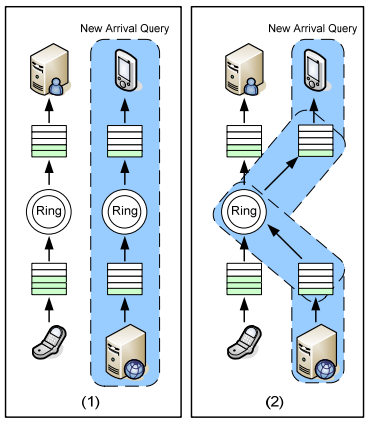
링 연산자는 HMTS 에서 제안한 가상 연산자(Virtual Operator)와 유사한 가상 연산자로 링 형태로 구성된다. 링 연산자는 하나 이상의 연산자로 구성되어 있으며 링 연산자를 구성하는 연산자는 기존의 DSMS 질의에 사용되는 연산자에 링 연산자에서 사용되는 두 가지 연산자가 추가되어 사용된다. 추가된 두 개의 연산자는 큐(Queue)로부터 데이터를 가져오는 get 연산자와 데이터를 저장하는 put 연산자이며 get 과 put 연산자는 각각 하나 이상의 슬롯(Slot)을 갖고 각 슬롯은 해당 큐를 참조한다.

질의 처리의 흐름은 소스에서 발생된 데이터 스트림이 입력 큐(Input Queue)에 저장되고 링 연산자는 입력 큐로부터 데이터를 가져와서 연산 후 결과를 출력 큐(Output Queue)로 저장한다. 그리고 다음 연산자가 그 결과로부터 데이터를 가져와 처리하여 저장하는 것을 반복하여 해당 목적지로 전송 하게 된다.

3. 2. ROTS 를 이용한 질의 처리

3. 2. 1. 링 연산자 생성

링 연산자의 생성 과정은 우선 새로운 질의가 들어오면 질의를 분석하여 각 연산자의 평균 처리 시간을 지정해둔 최소 임계 값 이하로 계속 더 하여 연결한다. 단, 각 연산자의 평균 처리 시간은 실행시간 동안 DSMS 로부터 제공받는다고 가정한다. 연결된 연산자에 get 과 put 연산자를 연결하고 각각의 슬롯에 입력 큐와 출력 큐를 참조시킨 후 하나의 링 연산자 전체의 평균 처리 시간의 합을 처리 값(Capacity)으로 저장한다. 완성된 링 연산자는 링 관리자에 고유 ID 로 등록되어 관리된다. 만약 만들어진 링 연산자와 동일한 구성을 가진 링 연산자가 링 관리자에 존재한다면 해당 링 연산자의 처리 값을 다시 더하였을 때 최대 임계 값 이하이면 등록되어있는 링 연산자로 대체하여 비어있는 다음 슬롯에 입력 큐와 출력 큐를 참조시킨다. 하지만 최대 임계 값을 초과하면 새로운 ID 를 부여하고 링 연산자를 등록한다.



(그림 3) 링 연산자 생성시 동일한 링 연산자가 등록되어 있지 않은 경우(1)와 등록되어 있는 경우(2)

3. 2. 2. 링 연산자의 수행 과정

링 연산자는 링 관리자에 의해 등록되어 사용되며 링 연산자의 수행 순서는 크게 세 단계로 진행 된다. 먼저 현재 슬롯의 입력 큐로부터 데이터를 가져오고(1) 가져온 데이터를 연결된 연산자들을 통해 순서대로 처리한다(2). 그리고 해당 출력 큐가 연결된 슬롯의 출력 큐에 저장한다(3). 그리고 다음 슬롯으로 이동하여 (1) ~ (3) 단계(그림 2)를 반복한다.

3. 2. 3. 링 연산자를 사용한 스케줄링의 특징

ROTS 는 생성된 링 연산자에 하나의 쓰레드를 할당하여 질의를 처리하게 된다. 이때 기존의 HMTS 의 가상 연산자과 다르게 한번 생성된 링 연산자를 다른 질의 처리에도 연결하여 쓰레드를 공유하는 것이 가능하기 때문에 쓰레드 할당을 감소 시켜 자원 관리의 효율성을 높이게 된다. 또한 해당 질의를 중지하더라도 링 연산자는 삭제하지 않고 새로 요구되는 질의에 링 연산자를 재사용하여 자원 할당에 따른 부하를 감소시킬 수 있으며 쓰레드는 메모리 사용을 의미하기 때문에 메모리 사용율 또한 감소 한다.

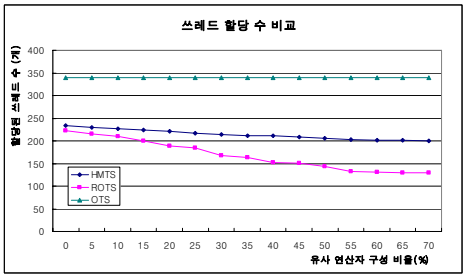
4. 성능 평가

성능평가에 사용된 시스템은 Intel® Pentium® 4 2.6Ghz CPU Chipset 을 사용하였으며 메모리는 4GB, 운영체제는 Windows XP 상에서 구동하였다. 평가 방법은 타 기법과 본 논문이 제안하는 ROTS 의 질의 수행을 위한 쓰레드 할당 수의 비교와 질의 처리 성능을 비교하여 본 기법의 우수성을 증명한다. 비교를 위해 두 기법의 시뮬레이터를 제작하여 평가 하였으며 시뮬레이터는 JAVA 기반으로 작성하였다. 평가에

사용된 질의 데이터 셋은 파싱(Parsing)된 연산자로 구성되어있으며 각 연산자의 평균 처리 값은 임의로 지정하여 사용하였다.

4. 1. 쓰레드 할당 수 비교

쓰레드 할당 수 비교를 위해 HMTS, OTS 와 본 제안기법에 질의 1,000 개를 입력하여 측정하였고 질의 내에 유사한 연산자 구성의 비율을 높여가며 입력하여 쓰레드 생성의 수를 비교하였다.

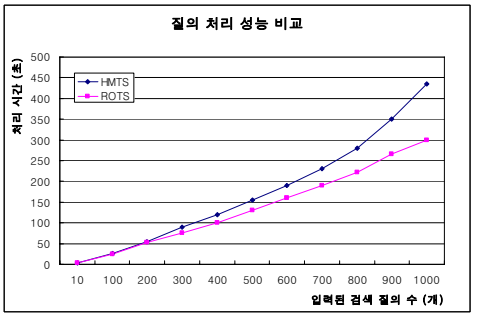


(그림 4) 유사 연산자 증가에 따른 쓰레드 할당 비교

(그림 4)는 HMTS, OTS 와 본 제안 기법의 쓰레드 할당 수 비교 결과 그래프이다. OTS 는 질의 내의 연산자 수 대로 쓰레드가 생성되기 때문에 유사 연산자 내포 비율과는 관련이 없다는 것을 알 수 있었고 HMTS 의 경우 저비용 연산자를 파티션으로 분할하여 쓰레드를 구성하여 쓰레드의 수가 크게 감소한 것을 알 수 있지만 유사 연산자 구성의 증가에 따른 변화는 다소 적었다. 그에 반해 ROTS 는 유사 연산자 비율에 따라 크게 감소하는 것을 볼 수 있었다. 하지만 일정 구간에는 큰 변화를 보이지 않는 것을 볼 수 있었는데 이는 하나의 쓰레드가 처리할 수 있는 링 연산자의 수가 한정되어 있기 때문에 새로운 링 연산자를 생성 하여 큰 변화를 주지 못한 것을 알 수 있다.

4. 2. 질의 처리 성능 비교

HMTS 와의 질의 처리 성능 비교를 두 기법에 동일한 검색 질의 10~1,000 개를 입력하여 초당 질의 처리량의 평균을 비교하였다.



(그림 5) 초당 질의 처리량의 평균 비교

(그림 5)의 그래프는 위에서 제시한 조건에서의 질의 처리 성능 비교이다. HMTS 는 질의의 수가 많아질수록 쓰레드의 수가 ROTS 보다 더욱 증가하여 질의 처리 속도가 늦어지는 것을 알 수 있다.

5. 결론 및 향후 연구

본 논문은 기존 쓰레드 스케줄링 기법의 연산자 비용에 의존적인 자원 관리의 비효율성 문제와 빈번한 자원 할당 및 해제로 인한 과부하 문제를 해결하기 위해 데이터 스트림 환경에서 질의 처리 성능 향상을 위한 링 연산자 쓰레드 스케줄링 기법을 제안하였다. 링 연산자 쓰레드 스케줄링 기법은 기존의 가상 연산자의 개념을 확장 시켜 공유 및 재사용 가능한 링 연산자를 사용하며 링 연산자는 여러 연산자를 하나로 묶어 하나의 연산자처럼 사용한다. 이미 생성된 링 연산자는 새로운 질의가 공유하여 사용 가능하기 때문에 자원 낭비를 감소 시키며 또한 한번 등록된 링 연산자는 재사용을 하여 불필요한 자원 할당 및 해제 시 과부하를 감소 시켜 기존 기법의 문제점을 해결했다.

향후 연구로는 링 연산자 구성의 최적화를 위한 연산자의 처리 비용 측정 기법과 링 연산자의 관리 및 연산자 맵핑의 효율성을 위한 인덱싱 기법에 대한 연구가 필요하다.

참고문헌

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. “Models and Issues in Data Stream Systems.” PODS, 2002

[2] S. Chandrasekaran, O. Cooper, A. Deshpande et al. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In Proc. of CIDR, 2003.

[3] S. Madden, M. J. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. In Proc. of ICDE, 2002.

[4] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In Proc. of ACM SIGMOD, pages 253–264, 2003.

[5] M. Cammert, C. Heinz, J. Kramer, B. Seeger, S. Vaupel, U. Wolske. Flexible Multi-Threaded Scheduling for Continuous Queries over Data Streams. In Data Engineering Workshop, 2007 IEEE 23rd International Conference on Publication April 2007 pages. 624-633

[6] D. J. Abadi, D. Carney et al. Aurora: A New Model and Architecture for Data Stream Management. VLDB Journal, 12(2). Pages. 120–139, 2003.

[7] R. Avnur, J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In Proc. of ACM SIGMOD, Pages. 261–272, 2000.

[8] S. Chandrasekharan, M. J. Franklin. “Streaming queries over streaming data.” VLDB, 2002, pp. 203-214

[9] Q. Jiang and S. Chakravarthy. Scheduling strategies for processing continuous queries over streams. In BNCOD, pages 16–30, 2004.

[10] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, M. Stonebraker. Operator scheduling in a data stream manager. In VLDB, pages 838–849, 2003.