제26회 한국정보처리학회 추계학술발표대회 논문집 제13권 제2호 (2006. 11)

AVL 트리를 활용한 스트림 데이터의 고속 집계 연산[1)]

김지현*, 김명
이화여자대학교 컴퓨터학과
e-mail:jhrosa@ewhain.net, mkim@ewha.ac.kr

Fast Aggregation of Stream Data Using AVL Trees

Ji-Hyun Kim*, Myung Kim
Dept. of Computer Science and Engineering, Ewha Womans University

요 약

스트림 데이터는 고속으로 생성되고 용량이 방대하여 저장하기 힘들며 데이터가 흘러가는 가운데 분석해야 하므로 기존 데이터 분석 방식을 그대로 사용하기는 어렵다. 본 연구에서는 스트림 데이터 분석 연산중의 하나인 다차원 집계 연산을 고속으로 처리하는 방법을 제안한다. 기존 연구들과 마찬가지로 스트림 데이터를 시간 차원 기준으로 윈도우 단위로 나누고, 각 윈도우마다 독립적인 집계 연산을 하도록 하였으며, 생성하고자 하는 집계 테이블들은 스트림 데이터가 입력되기 전에 미리 결정된다고 가정하였다. 정렬되지 않은 스트림 데이터를 고속으로 집계하기 위해 본 연구에서는 배열과 AVL 트리 구조를 혼합하여 사용하였다. 이 방법은 생성할 집계 테이블들 선택이 자유롭고, 집계 테이블들 전체가 메모리에 상주할 수 없을 정도로 큰 경우도 집계 연산을 실행할 수 있다는 장점을 갖는다. 제안한 방법의 효율성은 실험을 통해 입증하였다.

1. 서론

센서 네트워크 기술의 발달과 유비쿼터스 환경 구축이 실용화되면서 스트림 데이터를 분석하여 유용한 부가가치 정보를 얻고자 하는 노력이 최근 활발히 진행되고 있다. 스트림 데이터는 고속으로 끊임없이 생성되며, 생성 속도를 조절하기 어렵고, 분량이 방대하여 온라인으로 분석하기 힘들다는 특징이 있다 [1].

스트림 데이터 분석 연산들 중에는 다차원적 집계 연산(multidimensional aggregation)이 있다. 집계 연산 결과를 신속히 제공하기 위해 다차원 집계테이블을 미리 만들어 놓는 것이 일반적이며, 이 때 생성되는 집계테이블 집합을 큐브(cube)라고 한다[2]. 스트림 데이터로부터의 큐브 생성은 연산의 특성 상 데이터 전체를 읽은 후에라야 처리가 시작되므로, 끊임없이 흘러 들어오는 스트림 데이터를 대상으로 큐브를 생성하기는 쉽지 않다. 따라서 스트림 데이터에 집계 연산을 할 때는 데이터를 특정 단위로 (윈도우 단위) 잘라서 윈도우 마다 집계 연산을 하여 큐브의 일부만 생성하는 방법을 쓴다. 대표적인 연구로 StreamCube[3]와 Gigascope[4, 5]의 집계 연산 처리 방식을 들 수 있다.

StreamCube[3]의 단점은 롤업/드릴다운으로 연결될 수 있는 집계 테이블 시퀀스만 생성하며, 그들이 모두 하나의 거대한 트리로 표현되며 그 트리가 메모리에 상주할 수 있는 정도로 작다고 가정한다. Gigascope[4, 5] 역시 사용자가 원하는 집계 테이블들이 모두 주기억장치에 상주 가능한 정도로 작다고 가정한다.

본 연구에서는 이러한 큐브 생성 방법들의 단점을 보완한 큐브 생성 방법을 제시한다. 생성할 집계 테이블들은 제약 없이 사용자가 임의로 정할 수 있

1) 이 논문은 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임.
(2004-000-10149-0(2004)).

다. 집계 테이블 중에는 스트림 데이터로부터 직접 생성해야 하는 것이 있는데 이들이 메모리에 상주할 수 없을 정도로 커도 큐브 생성이 가능하며, 이외의 집계 테이블 생성에 드는 메모리 용량도 매우 작다.

본 연구에서는 정렬되어 있지 않은 스트림 데이터를 고속으로 집계하기 위해 AVL 트리와 배열을 혼합하여 사용한다. 제시한 방법의 효율성은 실험을 통해 검증하였다.

본 논문은 다음과 같이 구성된다. 2절에서 기존 연구에 대해 간략하게 분석하고, 3절에서 데이터의 다차원 분석에 대해 소개한다. 4절에서는 본 연구가 제시하는 큐브 생성 방법을 소개하고, 5절에서 성능 평가를 한 후에 6절에서 결론을 맺는다.

2. 기존 연구 분석

스트림 데이터를 집계하는 대표적인 연구로 StreamCube와 Gigascope의 집계 연산 처리 방식을 들 수 있다. 본 절에서는 이들을 간략하게 소개하기로 한다. StreamCube는 데이터 도메인 전문가가 정한 집계 테이블들만 생성한다. 데이터 도메인 전문가는 사용자가 관심을 둘 집계 테이블 시퀀스 S_1, S_2, ... , S_k를 미리 정해 준다. 여기서 S_{i+1}은 S_i로부터 집계 연산을 통해 생성이 가능하며, 사용자는 드릴다운(drill-down)이나 롤업(roll-up) 연산을 통해 이 시퀀스에 속한 집계 테이블들을 검색할 수 있다. StreamCube의 단점은 하나의 시퀀스로 연결될 수 있는 집계 테이블들만 생성한다는 점과 그들이 모두 하나의 거대한 트리로 표현되며 그 트리가 메모리에 상주할 수 있는 정도로 작다고 가정한다는 점이다.

Gigascope는 네트워크를 통과하는 데이터 패킷들을 모니터링하면서 그 패킷들의 소스, 목적지, 패킷 개수 등을 다차원적으로 신속하게 분석하는 연구이다. 스트림 데이터를 속도 빠른 캐쉬 메모리에서 집계하면서 가능한 만큼 압축하여 주기억 장치로 올려 보내는 연구이다. 이 방법은 네트워크 트래픽 데이터와 같이 클러스터링이 잘 되어 있는 데이터의 집계 연산에 효율적이다. 그러나 데이터가 다차원 공간에 고르게 분포되어 있는 경우는 캐쉬를 자주 비워야 하고, 주기억장치로 올려 보내지는 데이터가 정렬되어 있는 것이 아니므로 사용자가 원하는 집계 테이블들이 모두 주기억장치에 상주 가능한 정도로 작다는 가정을 하고 있으며 이것이 단점으로 지적될 수 있다.

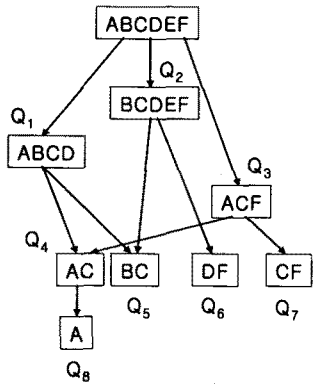
3. 스트림 데이터의 다차원적 분석

이 절에서는 본 연구에서 해결할 문제를 정의한다. 통신회사에서 고객의 무선 전화 통화 내역을 바탕으로 다양한 분석을 하려 한다고 가정해 보자. 고객 통화 내역이 발신인(A), 수신인(B), 발신 지역(C), 수신 지역(D), 통화 시작 시각(E)의 5개 차원으로 구성되어 있으며, 집계할 측정 데이터는 통화 시간이라고 하자. 이 데이터 스트림(또는 테이블)을 $ABCDE$로 부르기로 한다. 각 차원은 다음과 같은 계층 구조를 갖는다고 하자. 발신인 차원 A는 개인(A_1)과 연령별 그룹(A_2) 계층으로 나뉘고, 수신인 차원 B도 유사하게 B_1과 B_2로 나뉜다. 발신 지역 차원 C는 지역 범위에 따라 동(C_1), 구(C_2), 시(C_3) 계층을 가지며, 수신 지역 차원 D도 유사하게 D_1, D_2, D_3으로 나뉜다. 그리고 통화 시작 시각 E 차원은 분(E_1), 시(E_2), 일(E_3), 주(E_4) 계층을 갖는다. 이러한 데이터로부터 매 시간마다 특정 연령대의 사람들이 얼마나 전화를 걸었는가를 도시의 구 단위 범위로 분석해 보려면 $A_2C_2E_2$ 테이블을 생성하면 되고, 도시마다 시간대별로 전화 수신량을 분석해 보려면 D_3E_2을 생성하면 된다.

본 연구에서는 대용량 스트림 데이터가 매우 빠른 속도로 생성되어 도착한다는 가정 하에 큐브에 속한 집계 테이블들 중에서 사용자(또는 분석가)가 미리 관심이 있어서 등록해 놓은 집계 테이블들만 고속으로 생성하는 방법을 제안하고자 한다. 스트림 데이터는 시간 차원을 가지며, 데이터에 punctuation[6]이 포함되어 있어서 하나의 (시간) 윈도우가 끝나는 것을 알 수 있다고 가정하였다. 집계 연산 결과는 윈도우가 끝나는 곳마다 제공된다. 생성되는 집계 연산 결과는 모두 메모리에 남아 있거나 디스크의 파일로 정렬되어 출력될 수 있다.

4. 제안하는 집계 연산 방법

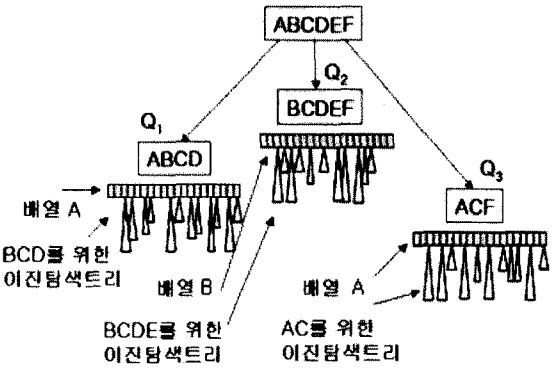
알고리즘의 이해를 돕기 위해 [그림 1]의 예제를 사용하기로 한다. 입력 데이터 스트림을 6차원 데이터로 보고, 이를 $ABCDEF$ 라고 하자. 사용자는 이 입력 데이터로부터 8개의 집계 테이블(또는 질의)인 $Q_1 \sim Q_8$를 생성하고자 한다. [그림 1]에서 집계 테이블간의 화살표는 집계 테이블이 어떤 집계 테이블들로부터 생성가능한가를 보여준다. 예를 들어, Q_4는 Q_1 또는 Q_3으로부터 생성될 수 있다.



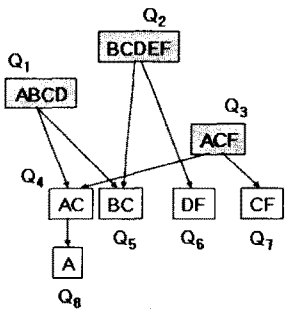
[그림 1] 집계 테이블들의 부모자식관계 ($Q_1 \sim Q_8$).

이제 집계 테이블 생성 방법을 소개하기로 한다. 알고리즘은 두 단계로 나뉜다. 단계 1에서는 입력 데이터 스트림으로부터 직접 생성해야 하는 집계 테이블들을 생성해 간다. 예제에서는 $Q_1 \sim Q_3$이 이에 해당한다. 이들은 [그림 2]와 같은 자료구조로 표현된다. 즉, 집계 테이블을 구성하는 차원 중의 하나는 1차원 배열로 표현되고, 이 배열의 각 원소는 하나의 AVL 트리를 가리키는 포인터를 갖는다. 예를 들어, 입력 데이터가 $a_i b_j c_k d_l e_m f_n$라고 하자. Q_1을 생성할 때 이 데이터는 $a_i b_j c_k d_l$ 셀에 집계되어 저장될 것이다. [그림 2]에서와 같이 차원 A가 배열로 선택된 경우 이 셀은 a_i를 나타내는 배열 원소 $A[i]$에 링크되어 있는 AVL 트리에 저장된다. AVL 트리에는 BCD의 셀들을 1차원적으로 펴놓은 순서대로 저장한다. 데이터 스트림을 읽어 가면서 $a_i b_j c_k d_l$ 셀이 여러 번 읽히게 되는데, 맨 처음에 읽히는 셀의 값은 AVL 트리에 노드로 삽입되고, 그 다음부터 입력되는 셀 값은 이미 생성된 노드에 집계된다. 데이터 입력이 진행되면서 AVL 트리는 계속 자라게 된다. 이런 자료구조를 사용하는 이유는 입력 데이터 스트림이 정렬되어 있지 않기 때문이다. AVL 트리는 정렬되어 있지 않은 (희박한) 데이터를 작은 양의 메모리를 사용하면서 신속하게 집계 연산하는데 유용하다 (여기서 사용되는 1차원 배열을 헤드 테이블(head table)이라고 부르기로 한다.) 헤드 테이블은 해쉬 테이블과 같은 효과를 주며 AVL트리의 깊이가 너무 깊어지는 것을 방지하는 역할을 한다.

윈도우 내의 모든 데이터가 읽히고 나면 단계 1의 집계 테이블의 생성이 완료된 것이고 이제 단계 2가 시작된다. 이 단계에서는 이미 생성되어 배열과 AVL 트리 형태로 저장되어 있는 집계 테이블들(예, $Q_1 \sim Q_3$)을 inorder tree traversal 방식으로 읽으면서 해당 테이블들은 디스크로 정렬된 형태로 출력하고, 동시에 아직 생성되지 않은 집계 테이블들(예, $Q_4 \sim Q_8$)을 생성한다. 단계 2에 제공되는 데이터는 정렬된 것이므로 이 단계에서는 기존의 비즈니스 데이터의 OLAP 큐브 생성에 사용되었던 효율적인 집계 테이블 생성 알고리즘[7]을 활용할 수 있다.



[그림 2] 스트림으로부터 직접 생성되는 집계 테이블들의 자료구조.



[그림 3] 단계 2에서 생성될 집계 테이블들 ($Q_4 \sim Q_8$).

단계 2의 집계 테이블 생성은 [그림 3]과 같이 계층적으로 진행된다. 집계 테이블들 중에는 단계 1에서 생성된 집계 테이블들로부터 직접 생성되는 것도 있지만 Q_8과 같이 부모 테이블이 단계 2에서 생성되는 것들도 있다. 집계 테이블 생성 시간은 부모 테이블을 한 번 스캔하는 시간이므로 부모 테이블의 크기에 비례한다. 따라서 Q_4나 Q_5와 같이 부모가 될 수 있는 집계 테이블의 여러 개인 경우는 부모 테이블의 차원 크기의 곱이 가장 작은 것을 택한다.

4. 집계 연산 알고리즘의 성능 평가

본 연구에서 제안하는 집계 알고리즘의 속도를 측정하기 위해 집계 알고리즘을 C 언어로 작성하여 1G 메모리가 장착된 3.0GHz 펜티엄에서 실행하였다. 스트림 데이터는 6차원 데이터로 파일에 저장되어 있다고 가정하였고 본 실험에서는 파일에 저장된 데이터 레코드의 수를 10M, 1M, 100K, 10K 에 대해 성능 평가를 해 보았다. 본 실험에서는 [그림 1]에 나타나 있는 큐브를 생성하였다. 즉 단계 1에서 집계 테이블 *ABCD*, *BCDEF*, *ACF*를 생성하였고, 단계 2에서 *AC*, *BC*, *DF*, *CF*, *A* 를 생성하였다.

[표 1]과 [표 2]에서 'B 트리 성능 최적인 경우'는 입력 레코드가 임의 순서라는 뜻이고 'B 트리 성능 최악인 경우'는 입력 레코드의 순서로 인해 생성될 B 트리가 매우 skew되는 경우를 뜻한다. [표 1]을 보면 레코드 순서가 임의일 때도 AVL 트리를 사용하는 경우가 B 트리를 사용하는 경우보다 낫다는 것을 알 수 있고, B 트리가 크게 skew되는 경우는 AVL 트리를 쓰는 것이 훨씬 효율적이라는 것을 알 수 있다. 또한 AVL 트리를 사용하는 경우 1초당 15만~20만 개의 레코드를 처리하면서 큐브를 생성하는 것을 알 수 있다.

[표 1] AVL 트리를 사용하는 경우와 B 트리를 사용하는 경우의 큐브 생성 시간 비교. (단위: 초)

	B 트리 성능 최적		B 트리 성능 최악	
입력 레코드수	AVL 트리 실행시간	B 트리 실행시간	AVL 트리 실행시간	B 트리 실행시간
10M	66.86	76.58	46.99	882.69
1M	5.44	6.30	4.61	50.84
100K	0.53	0.63	0.47	1.36
10K	0.08	0.08	0.08	0.01

[표 2]는 집계 테이블 *ABCD* 한 개만을 생성하는 경우의 실행시간이다. AVL 트리를 사용할 때는 입력 데이터 순서에 민감하지 않다는 것을 알 수 있다. 또한 AVL 트리를 사용할 때는 임의 순서 10M 분량의 데이터 읽어서 *ABCD*를 생성하는데 드는 시간이 45.53초이고, 남은 큐브를 생성하는 시간은 21.33초(즉, 66.86초-45.53초)이다. 큐브를 생성하는 시간보다 입력 데이터를 파일로부터 읽는 것이 상대적으로 더 시간이 많이 걸린다는 것을 알 수 있다.

[표 2] AVL 트리를 사용하는 경우와 B 트리를 사용하는 경우의 집계 테이블 *ABCD* 생성 시간 비교. (단위: 초)

	B 트리 성능 최적		B 트리 성능 최악	
입력 레코드 수	AVL 트리 실행시간	B 트리 실행시간	AVL 트리 실행시간	B 트리 실행시간
10M	45.53	46.17	41.16	454.92
1M	4.20	4.24	4.02	8.36
100K	0.44	0.44	0.42	0.86
10K	0.06	0.06	0.06	0.11

5. 결론

본 논문에서는 스트림 데이터를 고속으로 집계하는 알고리즘을 제안하였다. 정렬되지 않은 데이터를 집계하기 위해 배열과 AVL 트리 구조를 혼합하여 사용하는 것이 효율적임을 실험을 통해 보였다. 이 알고리즘은 대략 1초당 15만~20만 데이터 레코드를 고속으로 집계할 수 있으며, 알고리즘의 특성 상 입력 데이터 스트림으로부터 생성되는 집계 테이블이 메모리에 상주할 수 없을 정도로 커도 집계 연산이 가능하다.

참고문헌

[1] B .Babcock, S. Babu, M. Datar, R. Motwani, J. Widom. Models and Issues in Data Streams. *In Proc. ACM Symp. on Principles of Database Systems*, pp. 1-16, 2002.

[2] Sameet Agarwal, Rakesh Agrawal, Prasad M.Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramacrishnan, Sunita Sarawagi, "On the Computation of Multidimensional Aggregates," *Proc. of the 22nd VLDB Conference*, 1996

[3] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, Y. D. Cai, "Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams," *Distributed and Parallel Databases*, Vol. 18, pp. 173-197, 2005.

[4] R. Zhang, N. Koudas, B. C. Ook, D. Srivastava, "Multiple Aggregations Over Data Streams," *SIGMOD* 2005, pp. 299-310.

[5] C. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk, "Gigascope: A stream database for network applications," *SIGMOD*, 2003.

[6] P. A. Tucker, D. Maier, T. Sheard and L. Fegaras, "Exploiting Punctuation Semantics in Continuous Data Streams," *TKDE*, Vol. 15, No. 3, 2003.

[7] 김 명, 송지숙, "효율적인 큐브 생성 방법," 한국정보과학회 논문지 (데이터베이스), 제 29권 2호, pp. 99-109, 2002년 4월.