맵리듀스에서 집계 질의 스트림의 효율적인 처리 기법

최현진, 이기용
숙명여자대학교 컴퓨터과학부
e-mail: gomsun09@naver.com, kiyonglee@sookmyung.ac.kr

Efficient Processing of an Aggregate Query Stream in MapReduce

Hyunjean Choi, Ki Yong Lee
Division of Computer Science, Sookmyung Women's University

요 약

최근 들어 맵리듀스는 빅데이터 처리의 표준 기술로 자리잡고 있다. 빅데이터 분석에 널리 쓰이는 질의 중 하나는 집계(aggregate) 질의이다. 본 논문에서는 서로 다른 집계 질의가 계속적으로 요청되는 환경에서, 맵리듀스를 사용하여 이들 질의를 효율적으로 처리하는 방법을 제안한다. 제안 방법은 여러 집계 질의를 하나의 효율적인 맵리듀스 잡(job)으로 묶어 일괄 처리함으로써, 단순 방법에 비해 시간당 처리되는 질의 수를 크게 증가시킨다. 성능 평가를 통해, 제안 방법은 단순 방법에 비해 처리 성능을 크게 향상시킴을 확인하였다.

1. 서론

최근 들어 Social Networking Service (SNS), WWW, 센서 네트워크, 네트워크 모니터링 등 다양한 분야의 데이터가 급증하면서 소위 빅데이터(Big Data)[1]에 대한 관심이 커지고 있다. 빅데이터란 용량이 매우 크거나, 증가 속도가 매우 빠르거나, 형태가 매우 다양해서 현존하는 기술로는 효율적으로 처리할 수 없는 데이터를 말한다[2].

빅데이터는 일반적으로 단일 컴퓨터로 처리하는 것이 불가능하다. 빅데이터 플랫폼이란 빅데이터를 다수의 컴퓨터에 분산하여 저장하고, 이를 다수의 컴퓨터로 병렬 처리할 수 있도록 하는 플랫폼을 말한다. 하둡(Hadoop)[3]은 현재 대표적인 산업계 빅데이터 표준 플랫폼으로 자리잡고 있다. 하둡을 비롯하여 대다수의 빅데이터 플랫폼은 빅데이터의 병렬 처리를 위해 맵리듀스(MapReduce)[4]라는 기법을 사용한다.

맵리듀스에서는 사용자가 수행하고자 하는 작업을 맵(Map)과 리듀스(Reduce)라는 두 개의 함수로 표현한다. 시스템은 이를 자동으로 다수의 컴퓨터에서 병렬로 수행한다. 더우기 수행 도중 어떤 컴퓨터에 이상이 발생하더라도, 해당 컴퓨터의 작업을 다른 컴퓨터에게 재할당함으로써 고장 감내(fault tolerance) 기능을 제공한다. 따라서 사용자는 복잡한 병렬 처리 메커니즘과 복잡한 고장 감내 메카니즘을 전혀 모르고서도, 대용량의 데이터를 다수의 컴퓨터를 활용하여 손쉽게 분석할 수 있다. 이러한 사용의 용이성과 편리성으로 인해, 맵리듀스는 빅데이터에 대한 표준 처리 기술로 자리잡고 있다.

한편 빅데이터의 분석을 위해 널리 쓰이는 질의 중 하나는 집계(aggregate) 질의이다. 집계 질의의 대표적인 예는 SQL 의 group-by 질의로서, 데이터를 주어진 애트리뷰트(attribute) 값에 따라 그룹으로 나누고 각 그룹에 대해 합(sum), 개수(count), 평균(average) 등과 같은 집계치를 구하는 질의이다.

최근 빅데이터에 대한 접근성이 크게 향상되면서, 누구나 빅데이터에 접근하여 질의를 던지고 그 결과를 활용할 수 있도록 하는 연구가 활발히 진행되고 있다. 예를 들어, 여러 사용자의 스마트폰 사용 내역이 서버에 로그(log)로 저장되고 있을 때, 각 사용자는 로그 데이터에 대해 집계 질의를 던져 자신과 동일한 위치, 동일한 시간대, 혹은 동일한 연령대에 있는 사용자들의 사용 패턴에 대한 정보를 얻을 수 있다. 이 예는 로그라는 빅데이터에 대해 여러 집계 질의가 계속해서 요청되는 예로 볼 수 있다.

본 논문에서는 주어진 데이터에 대해 서로 다른 집계 질의가 계속해서 요청되는 경우, 맵리듀스로 이를 효율적으로 처리하는 방법을 제안한다. 앞서 언급한 바와 같이, 맵리듀스를 이용하면 병렬 처리와 고장 감내가 자동으로 지원되는 질의 처리기를 쉽고 효율적으로 구현할 수 있다. 제안 방법은 여러 집계 질의를 하나의 효율적인 맵리듀스 잡(job)으로 묶어 일괄 처리함으로써, 단순 방법에 비해 시간당 처리되는 질의 수를 크게 증가시킨다. 성능 평가를 통해, 제안 방법은 단순 방법에 비해 처리 성능을 크게 향상시킴을 확인하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 본 논문

에서 다루는 문제를 정의하고, 3 장에서는 관련 연구를 간략히 살펴본다. 4 장에서는 제안 방법을 자세히 설명하고, 5 장에서는 성능 평가 결과를 보인다. 마지막으로 6 장에서는 결론을 맺는다.

2. 문제 정의

본 논문에서는 데이터가 다수의 튜플(tuple)로 이루어진 집합이며, 데이터의 스키마(schema)는 다음과 같다고 가정한다.

$$(D_1, D_2, ..., D_k, M)$$

D_1, D_2, ..., D_k 는 시간, 위치 등과 같이 데이터의 그룹화에 사용되는 애트리뷰트이며, M 은 집계 대상이 되는 값을 저장하는 애트리뷰트이다.

시스템에는 여러 사용자가 각각 데이터에 대해 서로 다른 집계 질의를 계속해서 던질 수 있다고 가정한다. 각 질의 q 는 다음과 같이 표현된다.

$$q = (d_1, d_2, ..., d_k, AGG(M))$$

위 질의 q 는 D_1, D_2, ..., D_k 애트리뷰트의 값이 각각 d_1, d_2, ..., d_k인 튜플들에 대한 집계 함수 $AGG(M)$의 값을 요청하는 질의이다. 예를 들어, q = ("2013-11", "서울", *SUM*(*Requests*))는 2013 년 11 월에 서울에서 발생한 총 요청의 수를 구하는 데 사용되는 질의이다. 각 질의는 서로 다른 d_1, d_2, ..., d_k 값을 가질 수 있으며, 본 논문에서는 설명의 편의를 위해 *AGG* 는 *SUM* 이라고 가정한다. 하지만 제안 방법은 *SUM* 외에 *COUNT*, *AVG*, *MAX*, *MIN* 과 같은 일반적인 집계 함수에도 적용될 수 있다.

본 논문에서는 q 와 같은 형태의 질의가 계속해서 요청될 때(이를 질의 스트림이라고 하자), 이들을 맵리듀스를 사용하여 효율적으로 처리하는 방법에 대해 논의한다. 방법의 성능은 시간당 처리되는 질의 수로 측정한다.

3. 관련 연구

어떤 작업을 맵리듀스로 수행하기 위해서는 해당 작업을 맵과 리듀스 두 개의 함수로 표현해야 한다. 맵 함수는 입력으로 하나의 키-값 쌍(key-value pair) (k, v)을 받아 하나 이상의 키-값 쌍 (k_1, v_1), (k_2, v_2), ...을 출력으로 내보낸다. 맵 함수의 수행이 모두 끝나면, 시스템은 맵 함수가 출력한 키-값 쌍들을 정렬하여 같은 키 값을 가지는 키-값 쌍들을 모아 (k', [v'_1, v'_2, ...]) 형태로 만든 후, 이를 리듀스 함수의 입력으로 전달한다. 리듀스 함수는 입력으로 하나의 키-값리스트 쌍 (k', [v'_1, v'_2, ...])을 받아 하나 이상의 키-값 쌍 (k'_1, v''_1), (k'_2, v''_2), ...을 출력으로 내보내며, 이것이 맵리듀스의 최종 수행 결과가 된다.

맵리듀스에서 데이터는 분산 파일 시스템의 파일 형태로 저장된다. 대용량의 파일은 고정된 크기(주로 64 MB)로 파티션되어 저장되며, 각 파티션을 split 이라 부른다. 각 split 은 고장 감내를 위해 하나 이상의 컴퓨터에 중복되어 저장된다.

맵리듀스에서 수행되는 작업을 맵리듀스 잡(job)이라 부른다. 맵리듀스 잡은 하나의 맵 함수와 하나의 리듀스 함수로 구성된다. 어떤 맵리듀스 잡의 수행이 요청되면, 시스템에는 정해진 수만큼의 맵 태스크(map task)와 리듀스 태스크(reduce task)가 생성된다. 각 맵 태스크는 맵 함수를 수행하는 주체이며, 각 리듀스 태스크는 리듀스 함수를 수행하는 주체이다. 각 맵 태스크는 입력 데이터를 저장하고 있는 파일의 한 split 을 할당 받아, 해당 split 에 저장된 데이터를 지정된 방식에 따라 키-값 쌍들로 변환하고, 각 키-값 쌍에 대해 사용자가 정의한 맵 함수를 호출한다. 맵 함수의 호출 결과로 생성된 키-값 쌍들은 다시 파일로 저장된다. 모든 맵 태스크의 수행이 완료되면, 각 맵 태스크는 맵 함수가 출력으로 내보낸 키-값 쌍들을 키 값으로 정렬하여 이들을 리듀스 태스크들에게 분배하여 전송한다. 이 때 어떤 키-값 쌍을 어떤 리듀스 태스크에게 전송하느냐는 사용자가 지정한 규칙에 따르며, 어떤 경우든 동일한 키 값의 키-값 쌍들은 동일한 리듀스 태스크로 전송된다. 맵 태스크의 수행 결과로 생성된 키-값 쌍들이 모두 리듀스 태스크들로 전달되고 나면, 리듀스 태스크의 수행이 시작된다. 각 리듀스 태스크는 전송받은 키-값 쌍들을 키 값으로 정렬한 뒤, 동일한 키 값을 가지는 키-값 쌍들을 모아 이로부터 키-값리스트 쌍을 생성하고, 각 키-값리스트 쌍에 대해 사용자가 정의한 리듀스 함수를 호출한다. 리듀스 함수의 호출 결과로 생성된 키-값 쌍들은 파일로 저장되며, 이것이 맵리듀스 잡의 최종 결과물이 된다.

지금까지 설명한 맵리듀스를 사용하여 다양한 형태의 질의를 보다 효율적으로 처리하는 방법에 대해서 현재 많은 연구가 진행되고 있다. 예를 들어, SQL 에서 제공하는 Select, Project, Join, Group-By 등과 같은 질의를 맵리듀스로 어떻게 처리해야 하는지에 대해서 많은 연구가 이루어지고 있다[5][6]. 하지만 2 장에서 정의한 형태의 집계 질의가 계속적으로 요청되는 상황에 대해서는 아직까지 많은 연구가 이루어지지 않았다. 따라서 본 논문에서는 2 장에서 정의한 집계 질의가 계속해서 요청되는 경우, 이들을 효율적으로 처리하는 방법을 제안한다.

4. 제안 방법

4.1 단순 방법

2 장에서 정의한 집계 질의가 계속해서 요청되는 경우, 맵리듀스에서 내부적으로 정렬이 수행된다는 점을 이용하여 이들을 매우 간단히 처리할 수 있다. 질의가 계속해서 요청되는 경우, 각각을 개별적으로 하나의 맵리듀스 잡으로 만들어 처리하면 매우 큰 비용이 발생한다. 따라서 본 논문에서는 질의가 미리 정의된 개수만큼 도착하면(P 개라고 하자), 이들을 하나의 맵리듀스 잡으로 묶어 일괄처리한다.

P 개의 질의가 새로 도착했다고 하고, 이들 P 개 질의의 집합을 Q 라고 하자. 이들을 한꺼번에 처리하기 위해 하나의 맵리듀스 잡이 수행된다. 맵 함수는 데

이터의 각 튜플 $t = (d_1, d_2, ..., d_k, m)$을 입력으로 받아, 단순히 $(\{d_1, d_2, ..., d_k\}, m)$ 형태의 키-값 쌍을 출력으로 내보낸다. 맵리듀스는 내부적으로 이들을 정렬하여 $(\{d_1, d_2, ..., d_k\}, [m_1, m_2, ...])$ 형태의 키-값리스트 쌍을 생성하여 이를 리듀스 함수의 입력으로 전달한다. 리듀스 함수는 각 키-값리스트 쌍 $(\{d_1, d_2, ..., d_k\}, [m_1, m_2, ...])$을 입력으로 받아, 만약 Q에 질의 $q = (d_1, d_2, ..., d_k, SUM(M))$가 포함되어 있다면 $(\{d_1, d_2, ..., d_k\}, m_1 + m_2 + ...)$을 q의 최종 결과로 출력하고, 그렇지 않다면 아무것도 출력하지 않는다. 지금까지 설명한 간단한 방법을 NAIVE라 하자.

하지만 NAIVE 방법은 다음과 같은 문제점을 가진다. 맵 함수의 결과로 출력된 키-값 쌍들은 정렬 작업을 거쳐 네트워크를 통해 리듀스 태스크로 전송된 후, 리듀스 태스크에 의해 다시 한 번 정렬된다. 따라서 데이터의 튜플 수가 많을수록, 맵 함수가 출력한 키-값 쌍들을 정렬하고 네트워크로 전송하는 비용이 커지게 된다. 매 P 개의 질의가 도착할 때마다 이러한 맵리듀스 잡을 반복해서 수행해야 하므로, 시스템의 처리 성능, 즉, 시간당 처리되는 질의 수가 저하된다.

4.2 제안 방법

앞서 언급한 NAIVE 방법의 문제점을 개선하기 위해서는, 맵 함수가 출력한 키-값 쌍들을 리듀스 태스크로 전달하는 과정에서 발생하는 비용을 줄여야 한다. 즉, 맵 함수의 출력 결과를 파일에 쓰고, 해당 파일에 저장된 키-값 쌍을 정렬하고, 이를 네트워크를 통해 리듀스 태스크로 전송하고, 리듀스 태스크에서 전달받은 키-값 쌍들을 다시 정렬하는 비용을 줄여야 한다. 따라서 제안하는 방법은 리듀스 태스크를 아예 제거하고, 맵 태스크 내에서 최종 결과를 출력하는, 맵 태스크만으로 이루어진 맵리듀스 잡을 수행한다. 따라서 제안 방법에서는 맵 함수가 출력한 키-값 쌍들을 정렬하고, 이들을 네트워크로 전송하는 비용이 제거된다. 다음은 제안 방법에 대한 설명이다.

제안 방법에서 P 개의 질의가 새로 도착하면, 각 맵 태스크는 수행을 시작하기 전에 메모리에 다음과 같은 형태의 테이블 T를 구축한다.

Query	Group	Value
q_1	$d_{11}, d_{12}, ..., d_{1k}$	0
q_2	$d_{21}, d_{22}, ..., d_{2k}$	0
...		0
q_P	$d_{P1}, d_{P2}, ..., d_{Pk}$	0

테이블 T의 각 행은 Q에 포함된 각 질의 $q_1, q_2, .., q_P$에 대응된다. Query 열은 각 질의의 ID를 나타내며, Group 열은 해당 질의에 명시된 $D_1, D_2, ..., D_k$ 애트리뷰트의 값을 나타낸다. 마지막으로 Value 열은 해당 질의의 최종 결과인 $SUM(M)$를 담게 되는 열이며, 0으로 초기화된다.

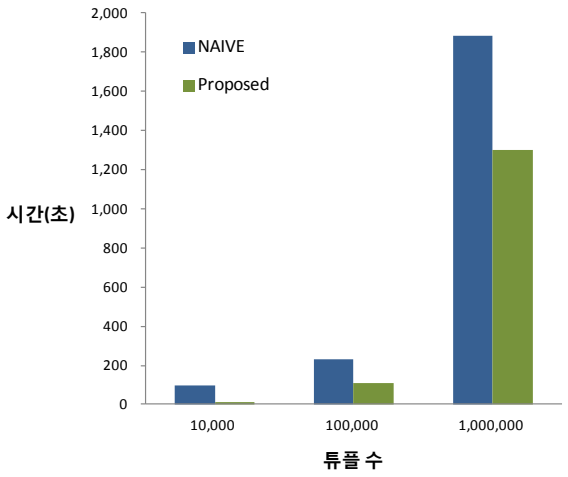
맵 함수는 데이터의 각 튜플 $t = (d_1, d_2, ..., d_k, m)$을 입력으로 받아, 테이블 T에서 Group 열의 값이 $d_1, d_2, ..., d_k$인 행을 찾는다. 만약 그런 행이 있으면 해당 행의 Value 열의 값을 m 만큼 증가시키고, 그런 행이 없으면 아무것도 하지 않는다. 따라서 테이블 T는 Q에 포함된 각 질의에 대해 현재까지의 $SUM(M)$ 값을 유지하고 있게 된다. 여기서 맵 함수는 아무런 키-값 쌍을 출력하지 않음에 유의하자. 데이터의 모든 튜플에 대해 맵 함수의 수행이 끝나면, 맵 태스크는 T의 내용을 질의의 최종 결과로 파일에 저장한다.

제안 방법은 리듀스 태스크를 전혀 수행하지 않고도 맵 태스크의 수행만으로 NAIVE와 동일한 결과를 얻을 수 있다. 맵 함수의 수행 비용도 각 튜플에 대해 메모리 상의 테이블 T를 갱신하는 것이므로 NAIVE에 비교하여 매우 큰 비용이 들지 않는다. 무엇보다 맵 태스크의 결과를 리듀스 태스크로 전달하는데 발생하는 비용, 즉, 키-값 쌍들의 정렬, 데이터 전송 등의 비용이 제거되므로, 매 P 개의 질의가 도착할 때마다 수행되는 맵리듀스 잡의 비용이 크게 줄어든다. 이에 따라 시간당 처리되는 질의의 수가 크게 향상된다. 더우기 테이블 T의 크기는 각 d_{ij}와 Value 열의 숫자를 저장하는데 B bytes가 든다고 했을 때, 총 $P \cdot (k + 1) \cdot B$ bytes에 불과하므로 큰 비용이 들지 않는다.

5. 성능 평가

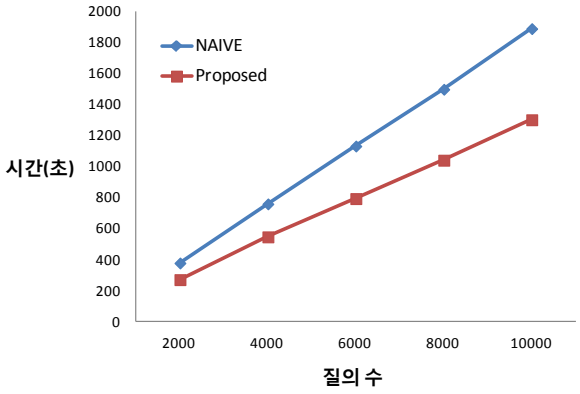
본 장에서는 제안 방법의 성능을 NAIVE 방법과 비교 평가한 결과를 제시한다. 두 방법의 성능 척도로는 동일한 개수의 질의를 처리하는데 걸린 총 시간을 측정하였다. 시간당 처리되는 질의 수는 총 처리 시간을 질의 수로 나눔으로써 쉽게 구할 수 있다. 실험에서는 두 방법 모두 10,000 개의 집계 질의를 처리하는데 걸린 총 시간을 측정하였으며, 500 개의 질의가 새로 도착할 때마다 각 방법에 따른 맵리듀스 잡을 수행하였다.

질의의 대상이 되는 데이터는 가상으로 생성한 것이며, 데이터에 포함된 각 튜플의 $D_1, D_2, ..., D_k, M$ 애트리뷰트의 값은 임의로 생성하였다. 데이터에 포함된 튜플 수는 10,000 개에서 1,000,000 개까지 변화시켰으며, 각 튜플의 크기는 1 KB 이고, 총 데이터의 크기는 최대 1 GB 였다. 각 질의의 $D_1, D_2, ..., D_k$ 애트리뷰트의 값은 데이터에 포함된 $D_1, D_2, ..., D_k$ 애트리뷰트의 값들 중에서 임의로 선택하였다. 실험에서는 Amazon EC2 서비스[7]에서 제공하는 6 대의 컴퓨터로 구성된 클러스터를 사용하였다. 맵 태스크와 리듀스 태스크의 수는 각각 10 개와 5 개로 하였다.



(그림 1) 튜플 수에 따른 성능 비교

(그림 1)은 데이터의 튜플 수를 변경시켜가며 두 방법의 성능을 비교한 결과이다. x 축은 데이터에 포함된 총 튜플 수이며, 10,000 개에서 1,000,000 개까지 증가시켰다. y 축은 주어진 데이터에 대해 총 10,000 개의 집계질의를 처리하는데 걸린 총 시간을 나타낸다. 4.2 절에서 설명한 바와 같이, 제안 방법은 매 P 개의 질의가 도착할 때마다 수행되어야 하는 맵리듀스 잡에서 리듀스 태스크를 사용하지 않고, 맵 태스크만으로 최종 결과를 생성하였다. 이에 따라 맵 태스크에서 생성된 키-값 쌍들을 리듀스 태스크로 전달하는데 발생하는 비용이 제거되어 전체적인 처리 성능이 크게 향상되었음을 알 수 있다.



(그림 2) 질의 수에 따른 성능 비교

(그림 2)는 시스템에 도착한 질의 수를 2,000 개에서 10,000 개까지 증가시켜가며, 도착한 질의를 두 방법에 따라 모두 처리하는 데 걸린 총 시간을 비교한 것이다. (그림 1)의 결과와 마찬가지로, 제안 방법은 NAIVE 방법에 비해 동일한 수의 질의를 처리하는데 훨씬 적은 시간이 드는 것을 확인할 수 있었다. 물론 제안 방법은 NAIVE 방법에 비해 각 맵 태스크가 메모리에 T 라는 별도의 테이블을 구축해야 하지만, 실험에서 T 의 크기는 약 9 KB 에 불과하였다. 따라서 제안하는 방법은 기존 방법에 비해 더 효율적으로 집계 질의의 스트림을 처리함을 확인할 수 있었다.

6. 결론

본 논문에서는 빅데이터 처리에 널리 사용되는 맵리듀스 환경에서, 계속해서 유입되는 집계 질의의 스트림을 효율적으로 처리하는 방법을 제안하였다. 제안 방법은 여러 집계 질의를 하나의 맵리듀스 잡으로 묶어 일괄적으로 처리한다. 또한 맵 태스크에서 부가적인 테이블을 메모리에 두고 활용함으로써, 리듀스 태스크를 전혀 거치지 않고도 올바른 질의 결과를 생성한다. 따라서 맵 태스크의 결과물을 리듀스 태스크로 전달하는 데 발생하는 비용을 크게 줄임으로써, 단순 방법에 비해 질의 처리 성능을 크게 향상시켰다. 실험 결과 제안 방법은 단순 방법에 비해 최대 7 배까지 성능을 향상 시킴을 확인하였다.

참고문헌

[1] http://en.wikipedia.org/wiki/Big_data
[2] Mark Beyer, "Gartner Says Solving 'Big Data' Challenge Involves More Than Just Managing Volumes of Data," Gartner, June 27, 2011.
[3] http://hadoop.apache.org/
[4] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," In Proceedings of OSDI '04, pp. 137-150, 2004.
[5] Spyros Blanas, Jignesh M. Patel, Vuk Ercegovac, Jun Rao, Eugene J. Shekita, Yuanyuan Tian, "A Comparison of Join Algorithms for Log Processing in MapReduce," In Proceedings of SIGMOD '10, pp. 975-986, 2010.
[6] http://hive.apache.org/
[7] http://aws.amazon.com/ec2/