맵리듀스에서 데이터 큐브의 효율적인 계산 기법 *

이기용, 박소정, 박은주, 박진경, 최연정
숙명여자대학교 컴퓨터과학부
e-mail : kiyonglee@sookmyung.ac.kr, ttojeong@gmail.com, peunju92@naver.com, jinkyunggg419@daum.net, yomii2@naver.com

Efficient Computation of Data Cubes in MapReduce

Ki Yong Lee, Sojeong Park, Eunju Park, Jinkyung Park, Yeunjung Choi
Division of Computer Science, Sookmyung Women's University

요 약

맵리듀스(MapReduce)는 대용량 데이터의 병렬 처리에 사용되는 프로그래밍 모델이다. 데이터 큐브(data cube)는 대용량 데이터의 다차원 분석에 널리 사용되는 연산자로서, 주어진 차원 애트리뷰트들의 모든 가능한 조합에 대한 group-by 를 계산한다. 차원 애트리뷰트가 n 개일 때, 데이터 큐브는 총 2^n 개의 group-by 를 계산한다. 본 논문은 맵리듀스 환경에서 데이터 큐브를 효율적으로 계산하는 방법을 제안한다. 제안 방법은 2^n 개의 group-by 를 분할하고 이들을 $\lceil n/2 \rceil$개의 맵리듀스 잡(job)을 통해 단계적으로 계산한다. 제안 방법은 각 맵리듀스 잡에서 맵 함수가 출력하는 중간결과의 크기를 최소화함으로써 총 계산 비용을 크게 줄인다. 실험을 통해 제안 방법은 기존 방법에 비해 데이터 큐브를 더 빠르게 계산함을 보인다.

1. 서론

최근 들어 서버 로그, 소셜 네트워크 서비스(SNS), 환경/교통/네트워크 모니터링 등 다양한 분야에서 데이터가 급증하면서 소위 빅데이터(Big Data)[1]에 대한 관심이 매우 커지고 있다. 빅데이터란 용량이 매우 크거나, 증가 속도가 매우 빠르거나, 형태가 매우 다양해서 현존 기술로는 효율적으로 처리하기 어려운 데이터를 말한다[2]. 이러한 빅데이터를 쉽고 효율적으로 처리하기 위해 맵리듀스라는 기법이 제안되었다[3]. 맵리듀스는 여러 컴퓨터에 분산되어 저장된 데이터를 손쉽게 병렬처리 할 수 있도록 해주는 프로그래밍 모델이다.

맵리듀스를 사용하기 위해 사용자는 자신이 수행하고자 하는 작업을 맵(Map)과 리듀스(Reduce)라는 두 개의 함수로 표현해야 한다. 이렇게 한 쌍의 맵과 리듀스 함수로 수행되는 작업을 맵리듀스 잡(job)이라 한다. 그러면 시스템은 이들 받아 자동으로 여러 컴퓨터를 사용하여 병렬적으로 수행한다. 더욱이 작업 수행 도중 어떤 컴퓨터에 문제가 발생하더라도, 맵리듀스는 해당 컴퓨터가 맡은 작업을 다른 컴퓨터로 대신 수행함으로써 고장 감내(fault tolerance) 기능을 제공한다. 따라서 사용자는 복잡한 병렬 처리 기법이나 고장 감내 기술을 전혀 모르고서도 다수의 컴퓨터를 활용하여 대용량의 데이터를 손쉽게 처리할 수 있다. 이러한 사용의 편리성과 용이성으로 인해 맵리듀스는 현재 대용량 데이터 처리의 표준 기술로 자리 잡았다.

데이터 큐브(data cube)는 대용량의 데이터를 다차원으로 분석하기 위해 널리 사용되는 연산자이다. 차원 애트리뷰트들이 주어졌을 때, 데이터 큐브는 그들의 모든 가능한 조합에 대한 group-by 를 각각 계산한다. 즉, 차원 애트리뷰트가 n 개일 때, 데이터 큐브는 그들의 모든 가능한 2^n 개의 조합에 대한 group-by 를 각각 계산한다. 따라서 데이터 큐브는 2^n 개의 group-by 를 계산해야 하므로 일반적으로 계산 비용이 매우 크다.

본 논문에서는 맵리듀스 환경에서 데이터 큐브를 효율적으로 계산하는 방법을 제안한다. 맵리듀스 환경에서 데이터 큐브를 효율적으로 계산하기 위한 연구는 이미 진행된 바가 있다[4][5]. 하지만 기존 방법은 2^n 개의 group-by 모두를 단 하나의 맵리듀스 잡으로 처리하기 때문에, 맵 함수가 출력으로 내보내는 중간결과의 크기가 매우 커진다는 단점이 있다. 그에 따라 이들을 디스크에 저장하고, 네트워크로 전송하고, 정렬하는데 드는 비용이 커져 총 계산 시간이 증가된다. 반면에 제안 방법은 2^n 개의 group-by 를 여러 그룹으로 분할하고, 이들을 $\lceil n/2 \rceil$개의 맵리듀스 잡을 통해 단계적으로 계산한다. 이 때 각 맵리듀스 잡에서 맵 함수가 출력으로 내보내는 중간결과의 크기를 최소화함으로써 각 맵리듀스 잡의 수행 비용을 최소화한다. 이에 따라 제안 방법이 수행해야 하는 맵리듀스 잡의 수는 증가하지만, 각 맵리듀스 잡의 수행 비용이 크게 줄어들기 때문에 총 계산 속도는 크게 향상된다. 실제 수행 시간을 측정한 실험을 통해, 제안 방법이 기존 방법에 비해 데이터 큐브의 계산 속

*본 논문은 2013 년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2012R1A1A1001269)

도를 크게 향상시킴을 보인다.

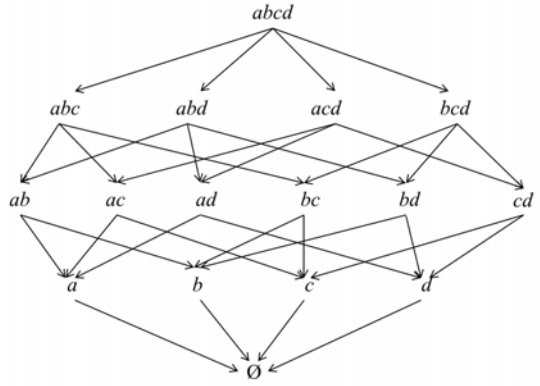
본 논문의 구성은 다음과 같다. 2 장에서는 데이터 큐브 및 맵리듀스에 대한 배경 지식을 설명하고, 3 장에서는 관련 연구를 살펴본다. 4 장에서는 제안 방법을 설명하고, 5 장에서는 실험 결과를 제시한다. 마지막으로 6 장에서는 결론을 맺는다.

2. 배경 지식

2.1 데이터 큐브(data cube)

데이터 큐브는 차원 애트리뷰트들이 주어졌을 때, 그들의 모든 조합에 대한 group-by 를 각각 계산하는 연산자이다. 예를 들어, 5 개의 애트리뷰트 *a*, *b*, *c*, *d*, *m* 으로 이루어진 릴레이션 *F*(*a*, *b*, *c*, *d*, *m*)에 대해, 차원 애트리뷰트가 *a, b, c, d* 이고 측정(measure) 애트리뷰트가 *m* 인 데이터 큐브는 총 2^4 = 16 개의 group-by, 즉, *abcd*, *abc*, *abd, acd, bcd, ab, ac, ad, bc, bd, cd, a, b, c, d*, ∅ 를 계산한다. 본 논문에서는 각 group-by 를 그의 그룹화 애트리뷰트만으로 간략히 나타낸다. 여기서 ∅는 그룹화 애트리뷰트가 없는 group-by 를 나타낸다. 데이터 큐브를 구성하는 각각의 group-by 를 큐보이드(cuboid)라 부른다.

데이터 큐브는 흔히 큐브 격자(cube lattice)라고 불리는 격자 다이어그램으로 표현된다[6]. (그림 1)은 앞서 예로든 차원 애트리뷰트가 *a, b, c, d* 인 데이터 큐브에 대한 큐브 격자를 나타낸다. 큐브 격자에서 어떤 큐보이드 *q* 에서 다른 큐보이드 *q'*로 향하는 간선(edge)은 *q* 로부터 *q'*를 계산할 수 있다는 의미이다. 예를 들어, 큐보이드 *a* 는 *ab* 나 *ac* 를 *a* 로 한 번 더 그룹화하여 구할 수 있다.



(그림 1) 큐브 격자 (cube lattice)

2.2 맵리듀스(MapReduce)

맵리듀스에서 어떤 작업을 수행하기 위해 사용자는 해당 작업을 맵과 리듀스 두 함수로 표현해야 한다. 이렇게 한 쌍의 맵과 리듀스 함수로 수행되는 작업을 맵리듀스 잡이라 한다. 맵 함수는 입력으로 하나의 키-값 쌍(key-value pair) (*k*, *v*)을 받아 하나 이상의 키-값 쌍들 (k_1, v_1), (k_2, v_2), ...을 출력으로 내보낸다. 맵 함수의 수행이 모두 끝나면, 맵리듀스는 맵 함수가 출력한 모든 키-값 쌍들을 키 값으로 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-값 리스트 쌍 (*k*, [v_1, v_2, ...])을 받아 하나 이상의 키-값 쌍 (k'_1, v'_1), (k'_2, v'_2), ...을 출력으로 내보내며, 리듀스 함수가 출력한 키-값 쌍들이 맵리듀스의 최종 수행 결과가 된다. 맵 함수의 수행을 담당하는 프로세스를 맵 태스크라고 부르며, 리듀스 함수의 수행을 담당하는 프로세스를 리듀스 태스크라고 부른다.

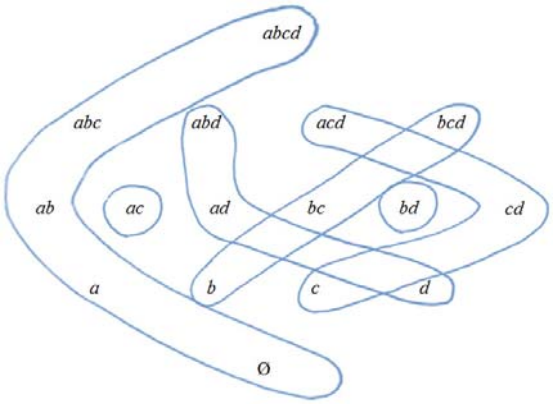
맵리듀스 잡을 수행할 때 맵 함수가 출력하는 키-값 쌍의 수가 많으면 많을수록, 이들을 디스크에 저장하고, 네트워크를 통해 리듀스 태스크로 전달하고, 키 값으로 정렬하는데 드는 비용이 커진다. 따라서 맵리듀스 잡의 수행 비용을 줄이기 위해서는 맵 함수가 출력하는 키-값 쌍의 수를 줄이는 것이 중요하다.

3. 관련 연구

맵리듀스로 데이터 큐브를 계산하는 가장 단순한 방법은 다음과 같다. 2.1 절에서 예로든 릴레이션 *F*(*a, b, c, d, m*)에 대해 차원 애트리뷰트가 *a, b, c, d* 인 데이터 큐브를 계산한다고 하자. 맵 함수는 *F* 의 한 튜플(tuple)을 입력으로 받아 각 큐보이드에 대해 하나씩 총 2^4 = 16 개의 키-값 쌍을 출력한다. 각 키-값 쌍은 각각 대응하는 큐보이드의 그룹화 애트리뷰트의 값을 키로 가지며, *m* 을 값으로 가진다. 예를 들어, 입력으로 들어온 *F* 의 한 튜플 < v_a, v_b, v_c, v_d, v_m>에 대해, 맵 함수는 ((v_a, v_b, v_c, v_d), v_m), ((v_a, v_b, v_c, *), v_m), ((v_a, v_b, *, v_d), v_m), ((v_a, *, v_c, v_d), v_m), …, ((*, *, *, *), v_m) 등 총 16 개의 키-값 쌍들을 출력한다. 각각은 리듀스 함수가 큐보이드 *abcd*, *abc*, *abd*, *acd*, …, ∅를 계산하는데 사용된다. 리듀스 함수는 ((v_a, v_b, v_c, v_d), [v_{m1}, v_{m2}, …]) 형태의 키-값 리스트 쌍을 입력으로 받아, v_{m1}, v_{m2}, …로부터 사용자가 지정한 집계 함수(SUM, AVG, MAX, MIN, COUNT 등)를 계산하고 그 결과를 출력으로 내보낸다. 그 결과는 해당 키 값이 속한 큐보이드의 한 튜플이 된다. 예를 들어, 리듀스 함수가 ((v_a, v_b, v_c, *), [v_{m1}, v_{m2}, …]) 형태의 키-값 리스트 쌍을 입력으로 받았다면, 리듀스 함수가 출력한 결과는 큐보이드 *abc* 의 한 튜플이 된다. 하지만 이 방법은 맵 함수가 *F* 의 각 튜플에 대해 2^4 = 16 개의 키-값 쌍을 출력해야 하므로 *F* 의 튜플 개수를 |*F*|라 할 때, 맵 함수에 의해 총 16|*F*|개의 키-값 쌍이 생성된다. 따라서 이들을 디스크에 저장하고, 네트워크를 통해 리듀스 태스크로 전달하고, 키 값으로 정렬하는데 매우 큰 비용이 발생하게 된다.

따라서 맵리듀스에서 데이터 큐브를 보다 효율적으로 계산하기 위한 연구가 진행되었다. [4]는 (그림 2)와 같이 2^n 개의 큐보이드들을 여러 batch 로 분할하여 계산하는 방법을 제안하였다. 분할할 때는 어느 한 리듀스 태스크에게 지나치게 많은 키-값 쌍들이 전달되지 않도록 분할한다. Batch 의 개수를 *B* 라고 하자. 맵 함수는 *F* 의 한 튜플을 입력으로 받아 각 batch 에 대해 하나씩 총 *B* 개의 키-값 쌍을 출력한다. 이 때, 각 키-값 쌍은 대응하는 batch 에 속한 큐보이드들 중 큐브 격자에서 가장 위에 위치한 큐보이드의 그룹화

애트리뷰트의 값을 키 값으로 가진다. 리듀스 함수는 키-값 리스트 쌍을 입력으로 받아 해당 키가 나타내는 큐보이드 뿐만 아니라 그 큐보이드가 속한 batch 의 모든 큐보이드들을 계산한다. 이 때는 단일 컴퓨터에서 데이터 큐브의 계산에 전통적으로 사용되는 BUC 알고리즘[7]을 사용한다. 이 방법은 맵 함수가 F 의 각 튜플에 대해 B 개의 키-값 쌍을 출력해야 하므로, 맵 함수에 의해 총 $B|F|$개의 키-값 쌍이 생성된다. 이것은 앞서 설명한 단순 방법에서 발생하는 $2^n|F|$개보다는 일반적으로 적은 수이지만, batch 의 수에 비례해서 맵리듀스 잡의 수행 비용이 커지게 된다.



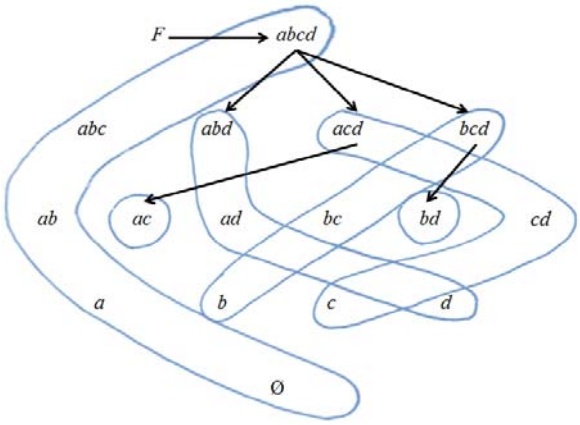
(그림 2) 큐보이드의 batch 분할 예

[5]는 [4]와 유사한 방법을 제안하였다. 다만 [5]는 2^n 개의 모든 큐보이드들을 하나의 맵리듀스 잡으로 계산하기 위해 필요한 batch 의 최소 수가 ${}_nC_{\lceil n/2 \rceil}$라는 것을 증명하였으며, 그에 따라 batch 의 수를 ${}_nC_{\lceil n/2 \rceil}$로 고정하였다. (그림 2)는 2^n 개의 큐보이드를 ${}_nC_{\lceil n/2 \rceil}$개의 batch 로 분할한 예에 해당한다. 따라서 이 방법은 맵 함수가 F 의 각 튜플에 대해 ${}_nC_{\lceil n/2 \rceil}$개의 키-값 쌍을 출력해야 하므로, 맵 함수에 의해 총 ${}_nC_{\lceil n/2 \rceil}|F|$개의 키-값 쌍이 생성된다. 이 방법은 batch 의 수를 최소화함으로써 하나의 맵리듀스 잡으로 2^n 개 큐보이드 모두를 처리하는데 필요한 비용을 최소화하였지만, 여전히 맵 함수에 의해 ${}_nC_{\lceil n/2 \rceil}|F|$개의 키-값 쌍들이 생성된다는 문제점이 있다.

4. 제안 방법

본 장에서는 제안 방법을 설명한다. 제안 방법은 하나의 맵리듀스 잡이 아닌 $\lceil n/2 \rceil$개의 맵리듀스 잡으로 데이터 큐브를 계산한다. 제안 방법은 $\lceil n/2 \rceil$개의 맵리듀스 잡을 사용하지만, 각 맵리듀스 잡의 수행 비용을 최소화함으로써 총 수행 비용을 크게 감소시킨다.

2.1 절에서 예로든 릴레이션 $F(a, b, c, d, m)$에 대해 차원 애트리뷰트가 a, b, c, d 인 데이터 큐브를 계산한다고 하자. 제안 방법은 큐브 격자의 위에서 아래로 진행하면서 큐보이드들을 단계적으로 계산한다. (그림 3)은 제안 방법의 수행을 보여주는 예이다.



(그림 3) 제안 방법의 수행 예

제안 방법에서 첫 번째 맵리듀스 잡은 다음을 수행한다. 맵 함수는 F 의 각 튜플에 대해 $abcd$ 에 대한 키-값 쌍을 출력으로 내보낸다. 리듀스 함수는 $abcd$ 에 대해 생성된 키-값 쌍들로부터 $abcd$, abc, ab, a, $\varnothing$를 계산한다. 두 번째 맵리듀스 잡에서 맵 함수는 $abcd$ 의 각 튜플에 대해 abd, acd, bcd 에 대한 키-값 쌍을 출력으로 각각 내보낸다. 리듀스 함수는 abd 에 대해 생성된 키-값 쌍들로부터 abd, ad, d 를 계산하고, acd 에 대해 생성된 키-값 쌍들로부터 acd, cd, c 를 계산하고, bcd 에 대해 생성된 키-값 쌍들로부터 bcd, bc, b 를 계산한다. 마지막 세 번째 맵리듀스 잡에서 맵 함수는 acd 의 각 튜플에 대해 ac 에 대한 키-값 쌍을 출력으로 내보내고, bcd 의 각 튜플에 대해 bd 에 대한 키-값 쌍을 출력으로 내보낸다. 리듀스 함수는 ac 에 대해 생성된 키-값 쌍들로부터 ac 를 계산하고, bd 에 대해 생성된 키-값 쌍들로부터 bd 를 계산한다. 따라서 3 개의 맵리듀스 잡으로 $2^4 = 16$ 개의 모든 큐보이드를 계산할 수 있다. 제안 방법은 3 개의 맵리듀스 잡을 사용하지만, abd, acd, bcd 등 F 에 비해 크기가 매우 작은 큐보이드들을 사용하여 다른 큐보이드들을 계산하므로 맵 함수가 생성하는 키-값 쌍의 개수를 크게 줄일 수 있다. 따라서 각 맵리듀스 잡의 수행 비용이 줄어들어 총 수행 비용이 크게 감소된다.

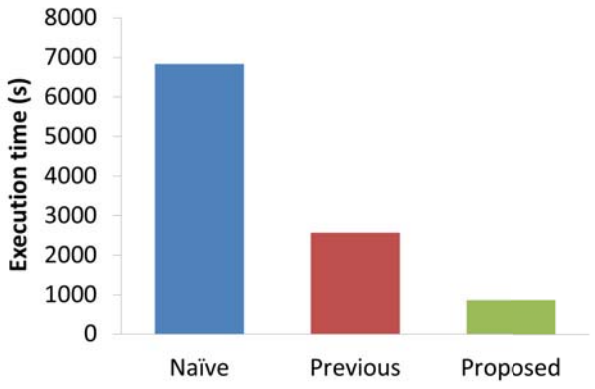
이제 제안 방법을 일반화한다. 제안 방법은 먼저 2^n 개의 큐보이드들을 ${}_nC_{\lceil n/2 \rceil}$개의 batch 로 분할한다. 그룹화 애트리뷰트의 개수가 i 인 큐보이드들의 집합을 $Level(i)$라 하자. 제안 방법은 $Level(n)$에서 $Level(0)$로 진행하며 $Level(i)$와 $Level(i-1)$에 속한 큐보이드들 간의 bipartite matching 을 구한다($i = n, n-1, \ldots, 1$). 이 때 큐브 격자에서 간선으로 연결된 큐보이드들에 대해서만 서로 연결이 가능하다. 이렇게 연결이 완료되면 총 ${}_nC_{\lceil n/2 \rceil}$개의 batch 가 생성된다. 이렇게 ${}_nC_{\lceil n/2 \rceil}$개의 batch 를 생성한 후, $\lceil n/2 \rceil$개의 맵리듀스 잡을 차례대로 수행한다. 첫 번째 맵리듀스 잡에서 맵 함수는 F 의 각 튜플에 대해 $Level(n)$에 속한 큐보이드에 대한 키-값 쌍을 내보낸다. 리듀스 함수는 이들로부터 해당 큐보이드가 속한 batch 의 모든 큐보이드들을 계산한다. $i(2 \le i \le \lceil n/2 \rceil)$번째 맵리듀스 잡에서 맵 함수는 $Level(n-i+2)$에 속한 큐보이드들을 입력으로 하여

Level($n - i + 1$)에 속한 큐보이드들 중 아직 계산되지 않은 큐보이드들에 대한 키-값 쌍들을 출력으로 내보낸다. 리듀스 함수는 *Level*($n - i + 1$)에 속한 큐보이드들 중 아직 계산되지 않은 큐보이드들에 대해 생성된 키-값 쌍들로부터 *Level*($n - i + 1$)에 속한 큐보이드들과 그들이 속한 batch 의 모든 큐보이드들을 계산한다. (그림 3)은 이러한 수행 예를 보여준다. 이렇게 하면 $\lceil n/2 \rceil$개의 맵리듀스 잡으로 2^n 개의 모든 큐보이드를 계산할 수 있다.

5. 실험 결과

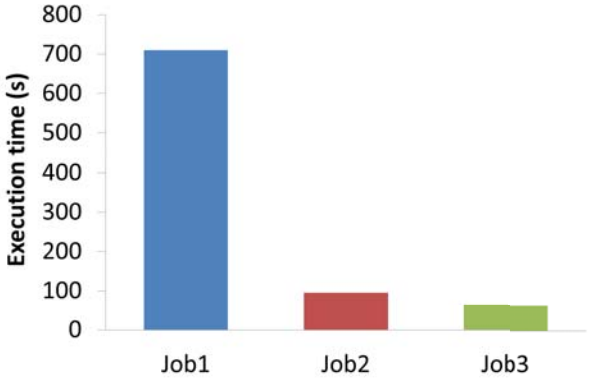
본 장에서는 제안 방법의 성능을 기존 방법과 비교한 실험 결과를 보인다. 성능 척도로는 데이터 큐브를 계산하는데 드는 총 시간을 측정하였다. 실험은 Amazon EC2 서비스[8]에서 제공하는 10 대의 컴퓨터로 구성된 클러스터를 사용하였다. 맵 태스크와 리듀스 태스크의 수는 각각 10 개와 5 개로 하였다.

실험 데이터로는 가상의 데이터를 생성하였다. 데이터를 저장하고 있는 릴레이션 F 는 6 개의 애트리뷰트 *a, b, c, d, m, dummy* 로 이루어져 있으며, *a, b, c, d* 는 차원 애트리뷰트, *m* 은 측정 애트리뷰트, *dummy* 는 임의의 값을 담는 애트리뷰트이다. *a, b, c, d, m* 의 크기는 각각 4 bytes, *dummy* 는 30 bytes 이며, 따라서 F 의 한 튜플의 크기는 50 bytes 가 된다. F 의 튜플 수는 총 10,000,000 개이며, *a, b, c, d, m* 의 값은 각각 [1, 100]의 범위에서 균등 분포로 임의로 선택하였다. 데이터 큐브는 *a, b, c, d* 를 차원 애트리뷰트로 하여 생성하였으며, 총 $2^4 = 16$ 개의 큐보이드로 구성된다.



(그림 5) 실험 결과

(그림 5)는 3 장에서 설명한 단순 방법(Naive), [5]에서 제안된 방법(Previous), 본 논문에서 제안한 방법(Proposed)으로 데이터 큐브를 계산하는 데 걸린 총 시간을 나타낸다. 3 장에서 설명한 바와 같이, Naive 는 맵 함수가 F 의 각 튜플에 대해 $2^4 = 16$ 개씩 총 $16 \times 10{,}000{,}000$ 개의 키-값 쌍을 내보낸다. 따라서 이들을 디스크에 저장하고, 네트워크로 전송하고, 정렬하는데 많은 비용이 발생한다. 한편, Previous 는 F 의 각 튜플에 대해 ${}_4C_{[4/2]} = 6$ 개씩 총 $6 \times 10{,}000{,}000$ 개의 키-값 쌍을 내보내므로, Naive 에 비해서 적은 비용으로 데이터 큐브를 계산할 수 있다. 위 두 방법에 비해 제안 방법은 3 개의 맵리듀스 잡을 사용하지만 각 맵리듀스 잡의 비용은 기존 방법에 비해 매우 작다. (그림 6)은 제안 방법에서 3 개의 맵리듀스 잡에 걸린 시간을 각각 나타낸다. 따라서 수행해야 하는 맵리듀스 잡의 개수는 증가하지만, 총 수행 시간은 감소함을 알 수 있다. 이에 따라 제안 방법이 기존 방법에 비해 데이터 큐브를 더 빠르게 계산함을 확인할 수 있다.



(그림 6) 제안 방법에서 각 맵리듀스 잡의 수행 시간

6. 결론

본 논문은 맵리듀스 환경에서 데이터 큐브를 효율적으로 계산하는 방법을 제안하였다. 제안 방법은 기존 방법에 비해 수행해야 하는 맵리듀스 잡의 개수는 증가하지만, 각 맵리듀스 잡에서 맵 함수가 출력으로 내보내는 키-값 쌍의 수를 대폭 감소시킴으로써 총 수행 비용을 크게 줄였다. 가상 데이터를 사용한 실험에서 제안 방법은 기존 방법에 비해 총 계산 시간을 약 1/3 로 감소시킴을 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] Jeffrey Dean, Sanjay Ghemawat, "MapReduce: simplified data processing on large clusters," In Proceedings of OSDI '04, pp. 137-150, 2004.
[4] Arnab Nandi, Cong Yu, Philip Bohannon, and Raghu Ramakrishnan, “Data Cube Materialization and Mining over MapReduce,” IEEE Transactions on Knowledge and Data Engineering, vol. 24, no. 10, pp. 1747-1759, 2012.
[5] Zhengkui Wang, Yan Chu, Kian-Lee Tan, Divyakant Agrawal, Amr EI Abbadi, Xiaolong Xu, “Scalable Data Cube Analysis over Big Data,” CoRR, abs/1311.5663, 2013.
[6] Venky Harinarayan, Anand Rajaraman, Jeffrey D. Ullman, “Implementing Data Cubes Efficiently,” In Proceedings of ACM SIGMOD, pp. 205-216, 1996.
[7] Kevin Beyer, Raghu Ramakrishnan, “Bottom-Up Computation of Sparse and Iceberg Cube,” In Proceedings of ACM SIGMOD, pp. 359-370, 1999.
[8] http://aws.amazon.com/ec2/