논문 2012-49-9-22

맵리듀스를 이용한 정렬 기반의 데이터 큐브 분산 병렬 계산 알고리즘

(Sort-Based Distributed Parallel Data Cube Computation Algorithm using MapReduce)

이 수 안*, 김 진 호**

(Suan Lee and Jinho Kim)

요 약

최근 많은 응용 분야에서 대규모 데이터에 대해 온라인 다차원 분석(OLAP)을 사용하고 있다. 다차원 데이터 큐브는 OLAP 분석에서 핵심 도구로 여긴다. 본 논문에서는 맵리듀스 분산 병렬 처리를 이용하여 효율적으로 데이터 큐브를 계산하는 방법을 연구하고자 한다. 이를 위해, 맵리듀스 프레임워크에서 데이터 큐브 계산 방법으로 잘 알려진 PipeSort 알고리즘을 구현하는 효율적인 방법에 대해서 살펴본다. PipeSort는 데이터 큐브의 한 큐보이드에서 동일한 정렬 순서를 갖는 여러 큐보이드를 한 파이프라인으로 한꺼번에 계산하는 효율적인 방식이다. 이 논문에서는 맵리듀스 프레임워크에서 PipeSort의 파이프라인을 구현한 네 가지 방법을 20대의 서버에서 수행하였다. 실험 결과를 보면, 고차원 데이터에 대해서는 PipeMap-NoReduce 알고리즘이 우수한 성능을 보였으며, 저차원 데이터에 대해서는 Post-Pipe 알고리즘이 더 우수함을 보였다.

Abstract

Recently, many applications perform OLAP(On-Line Analytical Processing) over a very large volume of data. Multidimensional data cube is regarded as a core tool in OLAP analysis. This paper focuses on the method how to efficiently compute data cubes in parallel by using a popular parallel processing tool, MapReduce. We investigate efficient ways to implement PipeSort algorithm, a well-known data cube computation method, on the MapReduce framework. The PipeSort executes several (descendant) cuboids at the same time as a pipeline by scanning one (ancestor) cuboid once, which have the same sorting order. This paper proposed four ways implementing the pipeline of the PipeSort on the MapReduce framework which runs across 20 servers. Our experiments show that PipeMap-NoReduce algorithm outperforms the rest algorithms for high-dimensional data. On the contrary, Post-Pipe stands out above the others for low-dimensional data.

Keywords : Multidimensional Data Cube, MapReduce, Distributed Parallel Computing, PipeSort

Ⅰ. 서 론

최근 정보 기술과 웹의 발전으로 많은 응용에서 대용량 데이터의 저장 및 분석이 요구되고 있다. 데이터의 효율적 분석을 위해 데이터 큐브[1]가 연구되었다. 데이터 큐브는 다차원 데이터 분석에 필요한 OLAP 연산을

* 학생회원, ** 정회원-교신저자, 강원대학교 컴퓨터과학과
(Kangwon National University)

※ 이 연구는 한국연구재단의 일반연구자지원사업 (과제번호: 2011-0011824)의 지원을 받았음
접수일자: 2012년6월11일, 수정완료일: 2012년9월14일

효율적으로 지원하기 위해, 여러 차원들에 대해 모든 가능한 집계 값을 유지하는 다차원 자료 구조이다. 데이터 마이닝, 데이터 웨어하우스 분야에서 의사 결정을 위한 다차원 분석에 데이터 큐브를 널리 활용하고 있다. 하지만 데이터 큐브를 구성하기 위해서는 많은 수의 집계 연산을 수행해야 하므로 계산에 많은 시간이 소요된다. 예를 들어, 만약 데이터 테이블의 튜플이 T이고, 데이터 큐브의 차원이 D개라면 가장 단순한 Naive 방식의 경우 $T \times 2^D$의 많은 계산 비용이 필요하게 된다[1].

이 논문에서는 맵리듀스를 사용하는 대규모 병렬처리 환경에서 효율적인 큐브 계산 방법을 연구하고자 한다. 이를 위해서 전체 큐브 계산에 효과적인 방법으로 알려진 PipeSort 알고리즘을 기반으로 맵리듀스 환경에서 분산 병렬처리하는 여러 가지 방법들을 제안하고 이들의 병렬처리 효과를 비교 분석하였다. 제안한 맵리듀스 데이터 큐브 알고리즘은 (1) 맵리듀스의 맵 함수에서 파이프 기법을 사용하는 PipeMap-Reduce 알고리즘, (2) 리듀스 함수에서 파이프 기법을 사용하는 Map-PipeReduce 알고리즘, (3) 리듀스 함수 없이 맵 함수에서만 파이프 기법을 사용하는 PipeMap-NoReduce 알고리즘, (4) 파이프 기법을 맵리듀스에서 사용하지 않고, 후처리로 사용하는 Post-Pipe 알고리즘으로 구분할 수 있다.

제안한 데이터 큐브 맵리듀스 알고리즘들은 병렬 처리 특성이 서로 다르기 때문에, 다양한 실험을 통해서 어떤 방법이 맵리듀스 환경에 적합한지 평가하였다. 이 실험을 통해 제안한 방법들이 분산 병렬 처리를 하지 않은 데이터 큐브 계산 알고리즘이 계산하지 못하는 대규모 고차원 데이터에 대해서 계산할 수 있었고, 대규모 다차원 데이터에 대한 큐브 계산에 실용적으로 사용이 가능하도록 효과적으로 분산 병렬 처리됨을 확인하였다. 실제 맵리듀스 프레임워크를 이용하여 개발된 다양한 관점의 맵리듀스 알고리즘들을 비교함으로써, 큐브 계산에서 병렬 처리 효과를 높일 수 있는 방안을 제시하였다. 본 논문의 결과를 통해 맵리듀스 프레임워크에서 데이터 큐브 계산에 대한 지표를 제시한다.

Ⅱ. 연구 배경

1. 데이터 큐브

데이터 큐브는 대용량 데이터를 다차원으로 모델링

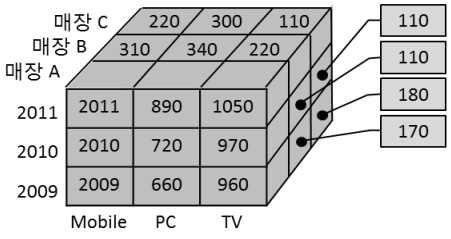
연도	판매량
2011	2560
2010	2210
2009	2100

(a) 1 차원

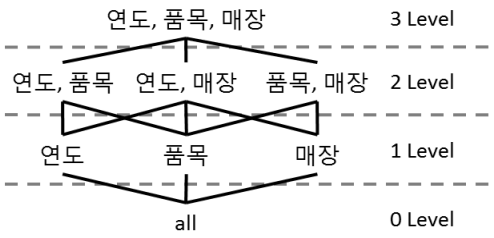
연도	Mobile	PC	TV
2011	890	1050	620
2010	720	970	520
2009	660	960	480

(b) 2 차원

	매장 A			매장 B			매장 C		
연도	Mobile	PC	TV	Mobile	PC	TV	Mobile	PC	TV
2011	360	410	280	310	340	220	220	300	110
2010	320	390	230	210	310	180	190	270	110
2009	260	380	200	220	310	170	180	270	110



(c) 3 차원



(d) 큐브 격자

그림 1. 데이터 큐브 구조
Fig. 1. Data Cube Architecture.

하여 다차원 분석 집계 결과를 신속하게 제공하기 위한 데이터의 집합이다. 큐브는 분석의 대상이 되는 데이터인 측정값과 분석의 관점에 해당하는 차원으로 구성되어 있으며, 2차원, 3차원, 또는 그 이상의 차원으로 구성된다. 큐브는 데이터베이스의 속성(차원)들과 해당 속성들에서 관심이 있는 측정값으로 구성된다.

만약 연도, 품목, 매장에 대한 차원과 총 판매수에 대한 측정값으로 데이터 큐브를 구성하면, 그림 1과 같이 나타낼 수 있다. 그림 1(a)는 연도 차원을 중심으로 총 판매수에 대한 데이터 큐브를 보여주고 있으며, 그림 1(b)는 연도와 품목의 두 차원에 대한 총 판매수를 보여주고 있다. 그리고 그림 1(c)는 연도, 품목, 매장의 세 차원에 대한 총 판매수를 나타낸다. 이와 같이 여러 조합 가능한 큐브를 고려할 수 있으며, 이들 조합 가능한 차원을 나타내는 큐보이드(cuboid)는 그림 1(d)와 같이 격자 구조를 가진다. 큐보이드는 데이터 큐브에 포함된 각각의 GROUP-BY를 의미한다.

2. 분산 병렬 컴퓨팅과 맵리듀스

대규모 분산처리는 대규모로 구성된 클러스터 환경에서 대용량 데이터를 분산 처리하는 기술이다. 대용량 데이터 처리에 적합한 시스템을 위해 분산 파일 시스템, 분산 데이터 관리 시스템 등이 개발되고 있다. 대규모 분산 처리는 대규모 데이터 처리에 적합한 데이터 웨어하우스, 데이터 마이닝, 정보 검색, 그리고 바이오인포메틱스와 같은 분야에 유용하다.

분산 파일 시스템은 클라이언트 측에서 서버에 저장된 데이터에 접근하여 마치 자신에게 저장되어 있는 데이터인 것처럼 처리할 수 있는 클라이언트/서버 기반의 파일 시스템이다. 분산 파일 시스템은 기존의 분산 파일 시스템과 다르게 거대하고, 고성능이어야 한다. 이를 통해 지속적인 데이터 증가에 효율적으로 대비해야 하고, 빈번하게 발생되는 고장에 대해서도 대처가 가능한 매커니즘이 필요하다.

대표적인 분산 파일 시스템은 구글에서 개발한 GFS(Google File System)[2]과 GFS와 동일한 구조와 기능으로 개발된 HDFS(Hadoop Distributed File System)[3]가 있다. HDFS는 오픈 소스 소프트웨어 개발 프로젝트인 Hadoop[4]에서 분산 컴퓨팅 프레임워크를 지원하기 위해 개발된 분산 파일 시스템이다. 현재 Amazon, IBM, Yahoo 등과 같은 기업들이 클라우드 컴퓨팅의 기반 플랫폼으로 널리 활용하고 있다.

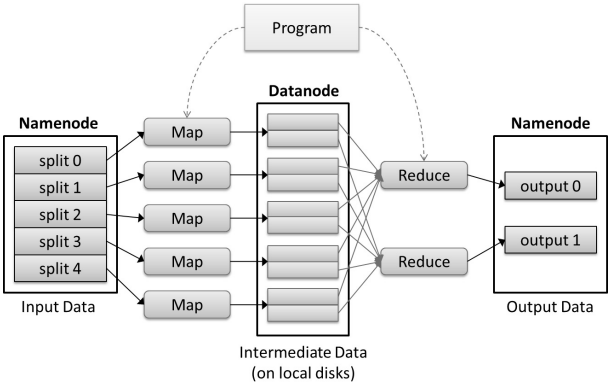


그림 2. 맵리듀스 구조
Fig. 2. MapReduce Architecture.

대규모 데이터의 분산 병렬 처리를 위한 기술로 맵리듀스가 있다. 구글에서 2004년도에 맵리듀스 병렬 처리 시스템을 논문[5]으로 공개하고, 구글의 다양한 서비스에 적용하였으며, 이를 기반으로 많은 업체에서 개발되었다. 기존의 MPI(Message Passing Interface)와 같은 병렬 처리 모델은 고성능의 컴퓨팅을 요구하는 분야에 적합한 반면 대규모의 데이터 처리에 적용할 경우에는 문제가 많았다. 맵리듀스는 데이터량에 따른 확장성, 노드간 데이터 이동시 네트워크 트래픽 최소화 등의 요구사항을 고려하여 만들어졌으며, 현재 대규모 데이터 처리 분야에서 맵리듀스 병렬 처리 모델은 거의 표준이 되고 있다. 맵리듀스는 그림 2와 같이 네임노드의 입력데이터는 맵 함수를 통해 각 컴퓨터의 로컬 디스크인 데이터 노드에 분산되고, 각 컴퓨터에 분산된 중간 데이터(intermediate data)는 정렬된 후, 리듀스 함수를 통해 수집되어 출력 데이터에 방출된다. 맵리듀스에서는 Hash 자료 구조와 유사한 〈key, value〉의 쌍으로 데이터를 처리 및 저장한다. 그리하여 쉽게 분산, 수집이 가능하고, 같은 키 그룹으로 정렬이 이루어진다.

3. 관련 연구

데이터 큐브 계산은 일반적으로 대용량 데이터를 대상으로 하며, 이때 사용하는 많은 집계 연산은 높은 비용이 요구된다. 이에 따라 데이터 큐브 계산을 효율적으로 수행하기 위한 많은 방법들이 연구되었다. 대표적인 연구로는 효율적인 데이터 큐브 생성을 위한 연산자를 소개한 연구[1]가 있고, 탐욕적(greedy) 알고리즘 기반으로 데이터 큐브를 구현하는 연구[6]가 수행되었다.

또한, 정렬 기반의 PipeSort 알고리즘[7]과 해시 기반의 PipeHash 알고리즘[7]이 제안되었는데, 이 연구에서는 큐브 계산에 있어서의 최적화 방법 (1) smallest-parent, (2) cache-results, (3) amortize-scans, (4) share-sorts, 그리고 (5) share-partitions을 제시하였다. 그리고 희박(sparse)한 데이터 큐브의 효율적 계산을 위해서 분할 정복 방식의 알고리즘을 사용한 파티션 큐브(partitioned-cube)[8]가 연구되었다.

데이터 큐브 계산의 고비용 문제를 해결하기 위해, 여러 컴퓨터를 이용한 병렬 계산이 고려되고, 연구되었다[9~10]. 특히 최근에는 맵리듀스 기술을 이용하여 수천, 수 만대의 컴퓨터를 이용하여 분산 병렬 처리를 쉽게 구현할 수 있게 되었다. 이러한 맵리듀스를 기술을 이용하여 대규모 병렬 처리 기술을 이용하여 데이터 큐브를 효율적으로 계산하는 방법에 대해서도 연구가 시도되고 있다. 하지만 이들 방법들은 빙산 큐브나 bottom-up 방법에 주로 치중하였으며, 전체 큐브 계산에 널리 사용하는 top-down 방법을 맵리듀스 환경에서 효율적으로 병렬 처리하는 방법에 대해서는 연구하지 않았다.

최근 대규모 데이터 큐브 구축을 위해 맵리듀스를 활용한 연구들이 많이 진행되고 있다. 맵리듀스를 이용한 데이터 큐브 연구에는 closed cube를 구축하는 연구[11]와 MapReduceMerge 기반의 데이터 큐브 구축 연구[12]가 진행되었다. 또한, 데이터 큐브의 집계값이 top-k(), mode() 그리고 median()과 같은 holistic의 특성을 가질 경우, value partitioning과 batch areas 기법을 사용하여 효율적으로 분산 큐브를 구축 하는 MRCube 연구[13]가 있다.

Ⅲ. 분산 병렬 데이터 큐브 계산

본 논문에서 제안한 분산 병렬 데이터 큐브 계산은 맵리듀스를 이용하여 데이터 큐브를 계산하는 과정에서

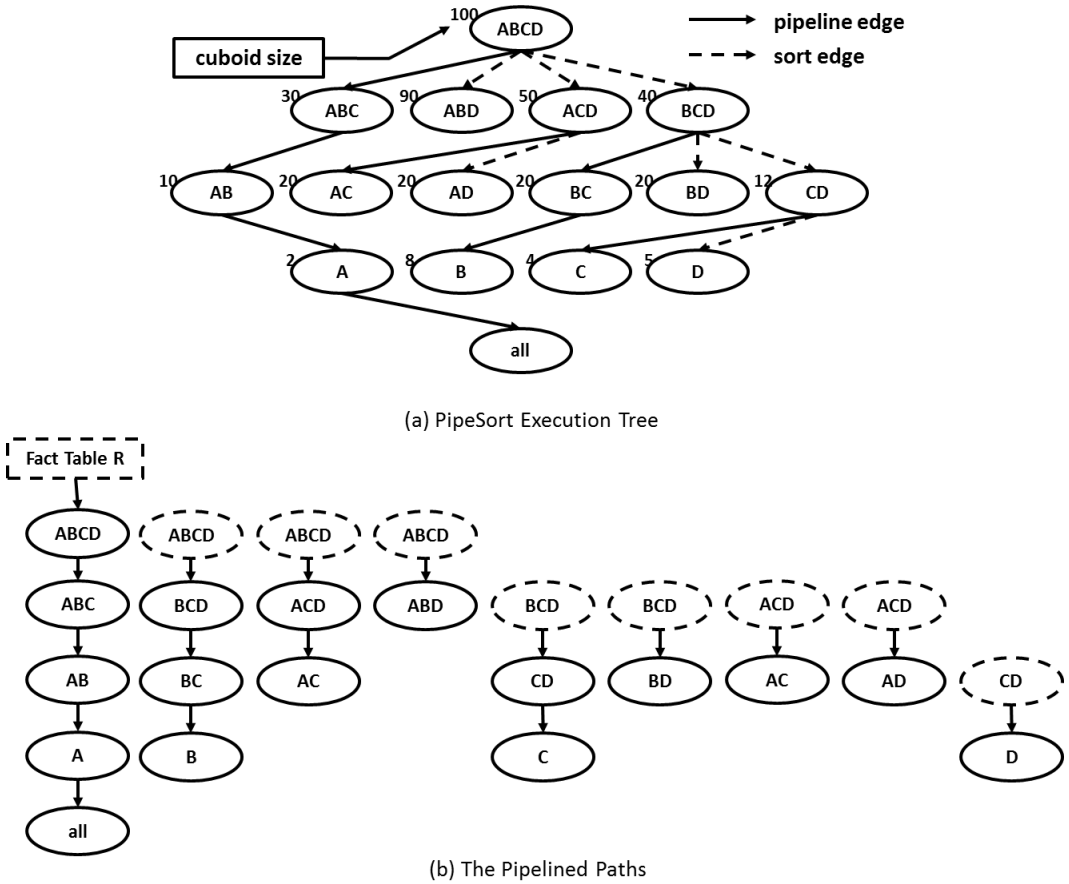


그림 3. PipeSort의 구조와 실행 트리 예제
Fig. 3. Examples of PipeSort Structure and Plan Tree.

PipeSort 알고리즘의 파이프 기법을 사용한다. 그리고 파이프 기법을 어느 부분에서 사용하느냐에 따라 구분하여 총 네 개의 알고리즘을 개발하였다. 맵 함수에서 파이프 기법을 사용하는 (1) PipeMap-Reduce 알고리즘, 리듀스 함수에서 파이프 기법을 사용하는 (2) Map-PipeReduce 알고리즘, 맵 함수에서 파이프 기법을 사용하지만, 리듀스 함수가 없는 (3) PipeMap-NoReduce 알고리즘, 그리고 파이프 기법을 맵리듀스를 이용하지 않고, 후처리로 처리하는 (4) Post-Pipe 알고리즘이 있다.

1. PipeSort 알고리즘

PipeSort 알고리즘은 정렬 기반의 데이터 큐브 계산 방법으로 PipeSort의 파이프 기법은 정렬된 데이터에 대해서 한 번에 구할 수 있는 큐보이드 집합을 함께 계산할 수 있다. 그림 3을 보면, (a) PipeSort 실행 트리와 (b) 파이프라인 경로를 볼 수 있다. 그림 3(a)는 큐보이드의 크기를 고려하여 큐브 격자 구조를 실행 트리로 생성한 것이다. 여기서 노드간의 연결은 자식 큐보이드가 부모 큐보이드의 접두사에 해당할 경우, 즉, 재정렬이 필요하지 않는 경우에는 파이프라인 연결(pipelined edge)이 되고, 접두사에 해당하지 않는, 즉, 재정렬이 필요한 경우에는 정렬 연결(sort edge)이 된다. 그림 3(b)는 노드간의 연결 경로를 나타낸 그림이다.

그림 4는 3차원 데이터에 대한 파이프 기법 예제이다. 그림을 보면, 왼쪽 테이블에 있는 데이터를 이용해 파이프라인인 ABC, AB, A, 그리고 all 큐보이드를 구하는 예제이다. 측정값을 COUNT로 지정한다. 먼저, 첫 번째 튜플인 [1, 1, 1]은 ABC, AB, A, all에 대해 각각 <1 1 1, 1>, <1 1 *, 1>, <1 * *, 1>, <* * *, 1>로 계산된다. 두 번째 튜필인 [1, 1, 2]는 ABC에 있는 <1 1 1, 1>의 cell과 서로 다르기 때문에 <1 1 1, 1>은 결과로 내보내고, <1 1 2, 1>을 입력한다. 나머지 AB, A, all은 각각 <1 1 *, 2>, <1 * *, 2>, <* * *, 2>로 계산된다. 세 번째 튜플인 [1, 2, 1]은 ABC에 있는 <1 1 2, 1>과 cell이 다르기 때문에 <1 1 2, 1>은 결과로 내보내고, <1 2 1, 1>을 입력한다. 또한, AB도 cell이 다르기 때문에 <1 1 *, 2>는 결과로 내보내고, <1 2 *, 1>을 입력한다. 나머지 A와 all은 각각 <1 * *, 3>, <* * *, 3>으로 계산된다. 네 번째 튜플인 [1 2 1]은 ABC, AB, A, all에 대해 각각 <1 2 1, 2>, <1 2 *, 2>, <1 * * 4>, <* * *, 4>로 계산된다. 다섯 번째 튜플인 [2, 1, 1]은 ABC에 있는 <1 2 1, 2>과 cell이 다르기 때문에 <1 2 1, 2>는 결과로 내보내고, <2 1 1, 1>을 입력한다. 또한, AB도 cell이 다르기 때문에 <1 2 *, 2>는 결과로 내보내고, <2 1 *, 1>을 입력하며, A도 마찬가지로 <1 * *, 4>를 결과로 내보내고, <2 * *, 1>을 입력한다. 이러한 과정을 전체 테이블의 데이터를 한 번 스캔하여 수행한다.

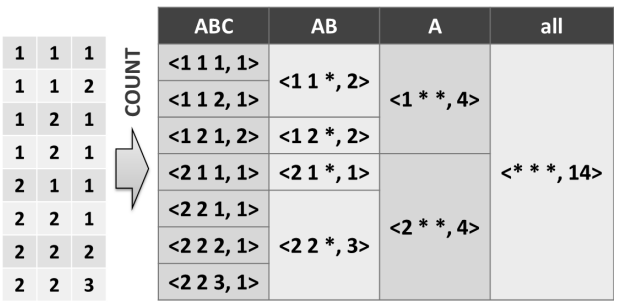


그림 4. 파이프 기법 예제
Fig. 4. An Example of Pipe Method.

2. PipeMap-Reduce

PipeMap-Reduce 알고리즘은 파이프 기법을 맵 함수에서 수행한 뒤, 리듀스 함수를 수행한다. 즉, 입력 데이터에 대해서 파이프로 연결된 모든 큐보이드를 계산한 뒤, 맵 함수에서 방출하게 된다. 그림 5는 ABC, AB, A, all 큐보이드에 대해 맵 함수에서 파이프 처리를 하는 예제이다. 맵 함수에서 파이프 처리를 통해 각 큐보이드의 결과가 방출되었으며, 리듀스 함수를 통해 정렬된 결과가 방출된다.

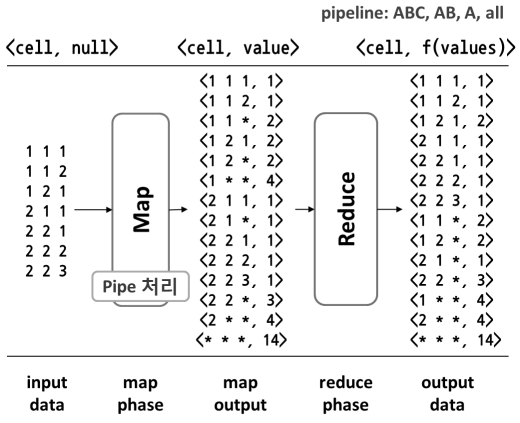


그림 5. PipeMap-Reduce 알고리즘 예제
Fig. 5. An Example of PipeMap-Reduce Algorithm.

3. Map-PipeReduce

Map-PipeReduce 알고리즘은 파이프 기법을 리듀스 함수에서 수행한다. 즉, 입력 데이터에 대해 맵 함수는 단순히 값을 측정한 뒤 방출하고, 리듀스 함수에서 파이프 기법을 사용하여 최종적으로 결과 큐보이드를 방출한다. 그림 6은 맵 함수를 통해 데이터 값에 측정값을 계산 한 뒤, 리듀스 함수를 통해, ABC, AB, A, all 큐보이드가 계산되어 결과로 방출하는 예제이다.

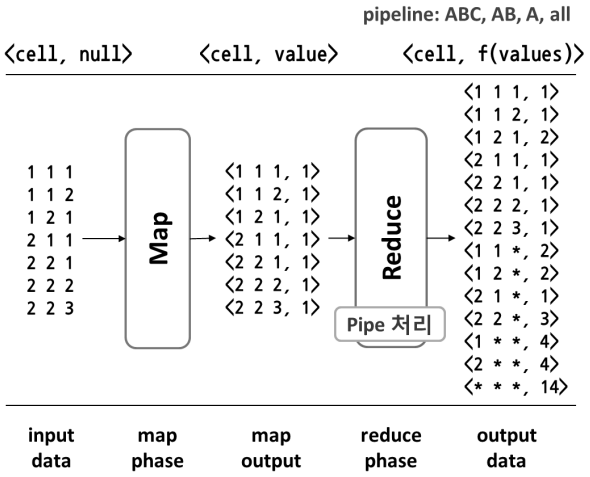


그림 6. Map-PipeReduce 알고리즘 예제
Fig. 6. An Example of Map-PipeReduce Algorithm.

4. PipeMap-NoReduce

PipeMap-NoReduce 알고리즘은 PipeMap-Reduce 함수와 동일하게 맵 함수에서 파이프 기법을 수행하지만, 리듀스 함수를 수행하지 않는다. 즉, 맵 함수에서 바로 결과 데이터로 방출하게 된다. 그림 7을 보면, 리듀스 과정이 없이 바로 맵 함수에서 결과 데이터를 방출한다. 하지만 리듀스 과정이 없으므로 빠르지만 결과 데이터가 정렬되지 않은 형태로 저장된다.

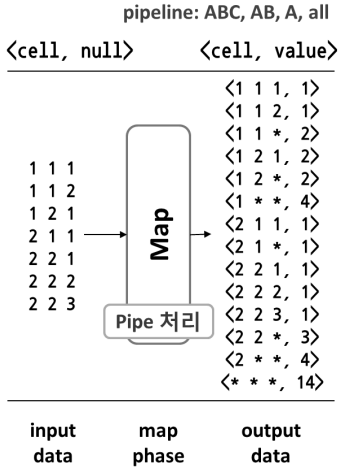


그림 7. PipeMap-NoReduce 알고리즘 예제
Fig. 7. An Example of PipeMap-NoReduce Algorithm.

5. Post-Pipe

Post-Pipe 알고리즘은 파이프 기법을 맵리듀스 처리가 끝난 후에 수행한다. 즉, 파이프 기법을 맵리듀스의 분산 병렬 효과 없이 수행한다. 그림 8을 보면, 입력 데이터를 맵 함수와 리듀스 과정을 거쳐 결과를 저장하고, 해당 결과를 이용하여 파이프 기법을 처리한다. 데이터가 작거나 저차원에서 효과적으로 동작하지만, 병렬 효과는 적어진다.

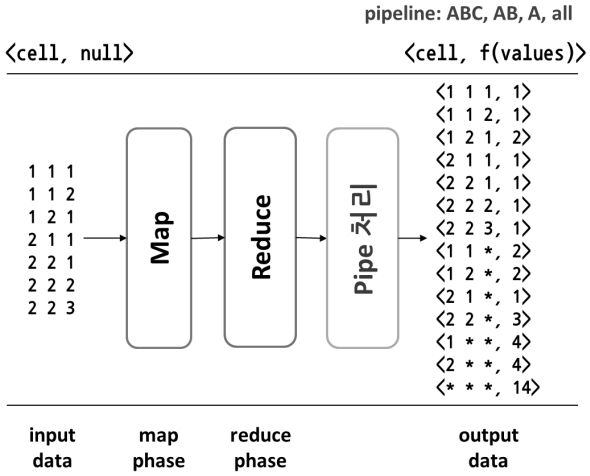


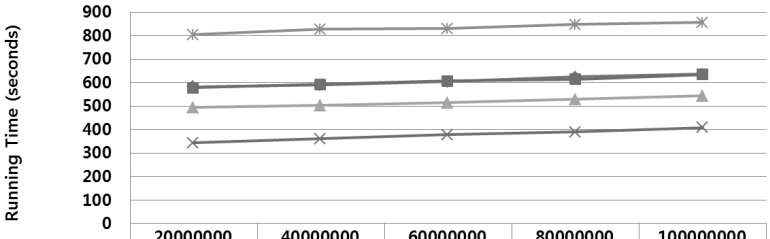
그림 8. Post-Pipe 알고리즘 예제
Fig. 8. An Example of Post-Pipe Algorithm.

Ⅳ. 실 험

1. 실험 환경

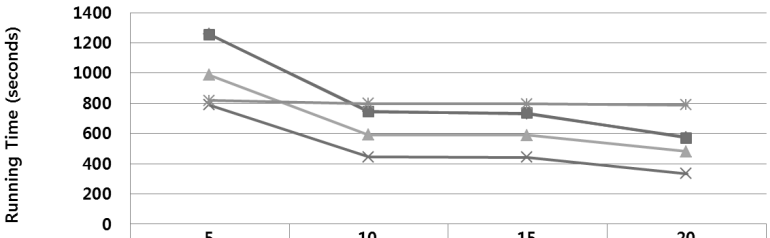
분산 병렬 데이터 큐브를 계산하기 위해 리눅스를 설치한 컴퓨터에 HDFS와 Hadoop의 맵리듀스를 수행하여 다차원 데이터 및 데이터 큐브를 구축한다. 본 논문에서는 총 20대의 컴퓨터를 사용하였고, 각 컴퓨터는 Intel Pentium 4 3.0GHz Dual Core, 512MB RAM, 150GB HDD, 1Gbps Network의 하드웨어를 가진다. 사용한 Hadoop은 0.20.2를 사용하였고, Java는 1.6을 사용하였다.

(a) 데이터 수의 증가



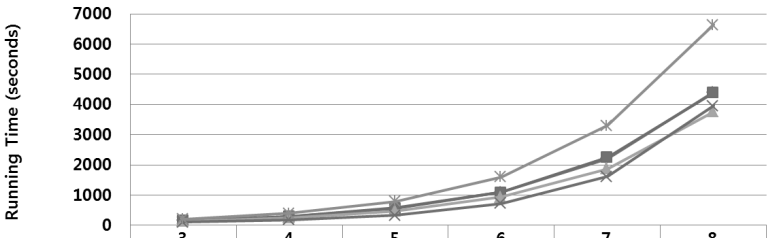
	20000000	40000000	60000000	80000000	100000000
PipeMap-Reduce	581.708	589.724	603.888	623.953	635.929
Map-PipeReduce	578.639	592.629	606.893	614.769	634.841
PipeMap-NoReduce	493.571	502.773	514.592	528.722	543.752
Post-Pipe	344.218	361.429	377.769	390.425	408.297
MRTDC	803.562	826.348	830.389	846.667	856.464

(b) 노드의 증가



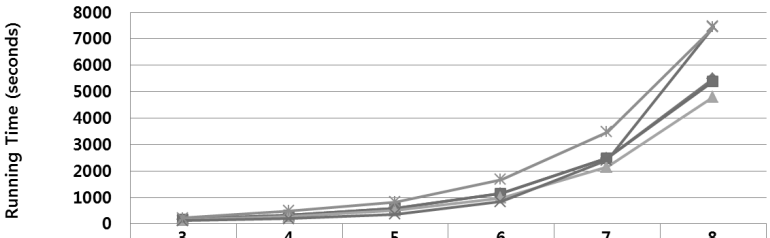
	5	10	15	20
PipeMap-Reduce	1258.255	745.016	730.008	574.692
Map-PipeReduce	1254.101	744.918	734.154	570.63
PipeMap-NoReduce	987.63	590.652	588.801	479.6
Post-Pipe	788.91	443.257	442.393	334.176
MRTDC	819.027	798.954	796.231	788.552

(c) 데이터 차원의 증가 (10M)



	3	4	5	6	7	8
PipeMap-Reduce	158.663	301.407	574.692	1100.724	2214.404	4405.569
Map-PipeReduce	156.583	300.33	570.63	1099.588	2254.239	4389.639
PipeMap-NoReduce	138.56	256.541	479.6	943.257	1868.34	3768.707
Post-Pipe	108.142	178.959	334.176	723.68	1617.808	3956.511
MRTDC	204.269	396.65	789.282	1604.049	3306.727	6635.54

(d) 데이터 차원의 증가 (50M)



	3	4	5	6	7	8
PipeMap-Reduce	169.706	328.671	595.887	1152.627	2482.915	5501.685
Map-PipeReduce	170.742	323.694	597.006	1153.467	2482.249	5391.1
PipeMap-NoReduce	151.676	272.651	509.559	985.243	2150.916	4810.047
Post-Pipe	121.33	198.62	368.662	839.285	2408.107	7467.468
MRTDC	225.314	493.004	826.349	1671.887	3483.529	7442.416

그림 9. 데이터 큐브 계산 시간
Fig. 9. Computation Time of Data Cube.

2. 데이터 수의 증가

데이터의 크기는 20,000,000부터 100,000,000까지 증가하여 실험하였으며, 실험 결과는 그림 9(a)와 같다. 실험을 보면, Post-Pipe 알고리즘이 가장 빠른 속도를 보였고, PipeMap-Reduce와 Map-PipeReduce는 매우 유사한 결과를 보였다. 실험에 추가된 MRTDC 알고리즘[14]은 비교를 위해 추가되었고, 실험한 네 개의 알고리즘 모두 MRTDC 알고리즘 보다는 빠른 결과를 보였다.

3. 노드의 증가

노드의 증가에 따라 실험한 결과는 그림 9(b)와 같다. 5대부터 20대까지 노드를 달리 하여 실험하였으며, 노드가 증가할수록 분산 병렬 처리할 PC가 많아짐으로 속도가 증가한다. 결과를 보면, Post-Pipe 알고리즘이 속도가 가장 빠르며, 5개의 노드에서 MRTDC 알고리즘과 비교하여 Post-Pipe 알고리즘이 더 효과적이고, 10개의 노드 이상부터는 네 개의 알고리즘 모두 효과적이다. 이런 결과는 MRTDC에 비해서 분산 병렬 효과가 더 좋은 것을 알 수 있다.

4. 데이터 차원의 증가

데이터 차원의 증가에 따른 실험은 3차원부터 8차원까지 증가하면서 데이터 크기를 10M와 50M로 구분하여 실험하였다. 데이터 크기가 10M인 차원 증가 실험은 그림 9(c)와 같고, 크기가 50M인 차원 증가 실험은 그림 9(d)와 같다. 두 실험 모두 비교를 위해 MRTDC 알고리즘을 추가하였다. 그림 9(c)를 보면, Post-Pipe 알고리즘이 빠른 속도를 보이다가 8차원 에서는 PipeMap-NoReduce 알고리즘보다 느려지는 것을 알 수 있다. 하지만 MRTDC 알고리즘보다 네 개의 알고리즘이 우수한 결과를 보인다. 그림 9(d)를 보면, Post-Pipe 알고리즘이 7차원에서는 PipeMap-NoReduce보다 속도가 느리다가 8차원에서는 다른 알고리즘과 MRTDC 알고리즘보다 느린 것을 알 수 있다. 그 이유는 차원이 증가하면서 처리할 양이 크게 증가하는데, 단일 서버에서 동작하는 Post-Pipe 알고리즘의 경우, 다른 분산 병렬 알고리즘에 비해서 느린 것으로 판단할 수 있다.

Ⅴ. 결론 및 향후 연구

본 논문에서는 데이터 큐브의 효과적인 분산 병렬 계산을 위해서 맵리듀스를 이용하였고, 맵리듀스의 적용 방법에 따라 네 개의 알고리즘을 개발하였다. 파이프 기법을 맵 함수에 적용한 (1) PipeMap-Reduce 알고리즘, 파이프 기법을 리듀스 함수에 적용한 (2) Map-PipeReduce 알고리즘, 리듀스 함수 없이 파이프 기법을 맵 함수에 적용한 (3) PipeMap-NoReduce, 파이프 기법을 후처리에서 수행한 Post-Pipe 알고리즘이 있다. 실험을 통해, 고차원에서는 PipeMap-NoReduce 알고리즘이 좋으며, 저차원에서는 Post-Pipe 알고리즘이 우수한 것을 알 수 있다. 현재 논문의 실험 결과를 토대로 PipeMap-NoReduce 알고리즘과 Post-Pipe 알고리즘을 결합한 새로운 알고리즘을 개발하고 있으며, 맵리듀스에 가장 적합한 데이터 큐브 분산 병렬 처리 알고리즘을 연구하고 있다.

참 고 문 헌

[1] Gray, J., et al., “Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals,” In Proc. Conf. on Data Engineering, New Orleans, LA, pp. 152-199, Feb. 1996.
[2] Ghemawat, S., Gobioff, H., and Leung, S. T., “The Google File System,” In Proc. 19th on Operating Systems Principles, Bolton Landing, NY, pp. 29-43, Dec. 2003.
[3] Hadoop, http://hadoop.apache.org/.
[4] HDFS, http://hadoop.apache.org/hdfs/.
[5] Dean, J. and Ghemawat, S., “MapReduce: Simplified Data Processing on Large Clusters,” Communication of the ACM, Vol. 51, No. 1, pp. 107-113, Jan. 2008.
[6] Harinarayan, V., Rajaraman, A., and Ullman, J. D., “Implementing Data Cubes Efficiently,” In Proc. Int’l Conf. on Management of Data, ACM SIGMOD, Montreal, Canada, pp. 205-216, June, 1996.
[7] Agarwal, S., et al., “On the Computation of Multidimensional Aggregates,” In Proc. the 22nd Int’l Conf. on Very Large Data Bases, pp. 506-521, Sept. 1996.
[8] Ross, K A., and Srivastava, D., “Fast Computation of Sparse Datacubes,” In Proc. the 23rd Int’l Conf. on Very Large Data Bases, pp. 116-125, Aug. 1997.
[9] Chen, Y., Dehne, F. A. A. Eavis, T., and

Rau-Chaplin, A., “PnP: Parallel And External Memory Iceberg Cube Computation,” Distributed and Parallel Databases, Vol. 23, No. 2, Apr. 2008.
[10] T. Ng, R., Wagner, A., and Yin, Y., “Iceberg-cube computation with PC clusters,” In Proc. Int’l Conf. on Management of Data, ACM SIGMOD, Santa Barbara, CA, pp. 25-36, June, 2001.
[11] Jinguo, Y. Jianging, X. Pingjian, Z. and Hu, C. “A Parallel Algorithm for Closed Cube Computation,” In Proc. 7th Int’l Conf. on Computer and Information Science, Portland, OR, pp. 95-99, May, 2008.
[12] Yuxiang, W. Aibo, S. and Junzhou, L. “A MapReduceMerge-based Data Cube Construction Method,” In Proc. 9th Int’l Conf. on Grid and Cooperative Computing, Nanjing, China, pp. 1-6, Nov. 2010.
[13] Arnab, N. Cong, Y. Philip, B. and Raghu, R. “Distributed Cube Materialization on Holistic Measures,” In Proc. 27th Int’l Conf. on Data Engineering, Hannover, Germany, pp. 183-194, Apr. 2011.
[14] Suan, L. Yang-Sae Moon, Jinho. K. “Distributed Parallel Top-Down Computation of Data Cube using MapReduce,” In Proc. 3rd Int’l Conf. on Emerging Databases, Inchoen, Korea, pp. 303-306, Aug. 2011.

저 자 소 개

이 수 안(학생회원)
2008년 강원대학교 컴퓨터과학과 학사 졸업.
2010년 강원대학교 컴퓨터과학과 석사 졸업.
2012년 현재 강원대학교 컴퓨터과학과 박사과정 재학.
<주관심분야 : 데이터 웨어하우스, OLAP, 데이터 마이닝, 정보검색, 빅 데이터 분석, 맵리듀스, 소셜 네트워크, 모바일>

김 진 호(정회원)-교신저자
1982년 경북대학교 전자공학과 학사 졸업.
1985년 KAIST 전산학과 석사 졸업.
1990년 KAIST 전산학과 박사 졸업.
2012년 현재 강원대학교 컴퓨터과학과 교수.
<주관심분야 : 데이터 웨어하우스, OLAP, 데이터 마이닝, 정보검색, 소셜 네트워크>