ETRI

2013
Electronics and
Telecommunications
Trends

대규모 데이터 분석을 위한 MapReduce 기술의 연구 동향

The MapReduce framework for Large-scale Data Analysis: Overview and Research Trends

이경하 (K.-H. Lee) 스마트미디어플랫폼연구실 선임연구원
박원주 (W.J. Park) 스마트미디어플랫폼연구실 선임연구원
조기성 (K.S. Cho) 스마트미디어플랫폼연구실 실장
류 원 (W.Ryu) 지능형융합미디어연구부 부장

* 본 연구는 미래부가 지원한 2013년 정보통신 · 방송(ICT) 연구개발사업의 연구결과로 수행되었음.

MapReduce는 다양한 형식의 대용량 데이터를 병렬 처리하는데 있어 효과적인 도구로 인식되고 있다. 특히 MapReduce의 오픈 소스 구현인 Hadoop은 여러 분야에서 널리 이용되고 있으며, 가장 대표적인 빅데이터 솔루션으로 현재까지 많은 주목을 받아오고 있다. 하지만, MapReduce는 그 구조적 특징으로 인한 이점과 함께 여러 제약과 단점들을 가진다. 이에 따라 MapReduce의 개선을 위한 많은 연구와 시스템 개량이 학계와 산업계에서 동시에 수행되어 왔다. 본고에서는 대용량 데이터 분석을 위한 MapReduce 프레임워크의 특성과 이를 개선하기 위한 최근의 연구 내용들을 소개한다. 또한 향후의 대용량 데이터 처리는 어떠한 모습을 취하게 될 것인지를 예측해 본다.

Ⅰ. 서론

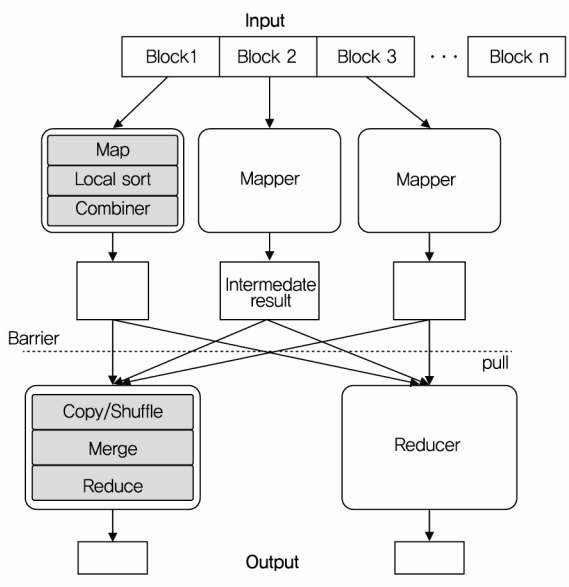
최근의 대용량 데이터 처리는 대량의 컴퓨팅 자원을 이용한 병렬 처리를 기본으로 하고 있다. 이는 Linux나 오픈 소스 S/W와 같이 보다 저렴해진 OS와 S/W, 그리고 보다 강력해진 성능의 low-end 서버들, 보다 빨라진 네트워크 기술의 이점에 기인한다. 이러한 대용량 데이터에 대한 병렬 처리는 기존의 단일 머신 상에서 DBMS로는 어려웠던 대용량 데이터 처리를 보다 저비용으로 실현할 수 있는 방안을 마련하여 주었다[1].

Google에 의해 고안된 MapReduce기술[2]은 대표적인 대용량 데이터 처리를 위한 병렬 처리 기법 중 하나로 최근까지 많은 각광을 받고 있다. 특히 MapReduce의 오픈소스 Java 구현인 Hadoop[3]은 많은 분야에서 대용량 데이터 처리를 위한 핵심 기술로 큰 인기를 누리고 있다. 이러한 MapReduce의 성공은 단순하면서도 scale-out을 통한 확장성이 용이하고, 내고장성(fault-tolerance)을 지원하는 MapReduce의 특징에 기인하지만, 성능이나 효율성 측면에서는 내부적으로 한계 또한 존재하는 것이 사실이다.

본 분석의 목적은 현재까지 MapReduce의 기술적 특성과 MapReduce의 한계를 극복하기 위한 주요 연구들과 시스템 개량에 관한 최신의 동향을 소개하는데 있다. 본고의 구성은 다음과 같다. 2장에서는 MapReduce프레임워크의 구조적 특징을 소개하고 3장에서는 MapReduce의 장점과 단점에 관하여 논한다. 4장에서는 MapReduce의 성능 개선을 위한 최근의 여러 연구들과 개선 방향에 대해서 소개한다. 5장에서는 MapReduce 이후의 최근의 대용량 데이터 솔루션들을 살펴보고 결론을 맺는다.

Ⅱ. MapReduce 프레임워크의 구조적 특징

MapReduce는 프로그래밍 모델과 동시에 이 모델을



(그림 1) MapReduce의 시스템 구조[1]

구동하는 프레임워크 모두를 의미한다((그림1)참조). MapReduce는 병렬 처리에 있어 요구되는 스케줄링 등의 복잡한 사항들을 사용자가 고민하지 않게 함으로써 쉬운 병렬 처리 방법을 제공한다. MapReduce 프로그래밍 모델은 사용자가 Map과 Reduce 두 함수를 이용하여 데이터를 처리하도록 한다. MapReduce의 입력은 각 레코드가(*key, value*)의 쌍으로 구성되는 일련의 리스트 형태를 갖는다. Map 함수는 이(*key, value*)쌍의 리스트를 읽어 다시 (*key2, value2*) 형태의 중간 결과를 출력한다. 이 중간 결과들은 *key2*를 기준으로 그룹화되어 동일한 *key2*값에 대하여 (*key2, values[]*)형태로 구성되고, Reduce 함수는 각 *key2*에 대한 값 리스트들에 대해 집계 연산(aggregation)을 수행, 최종 결과를 출력한다.

MapReduce의 입력과 출력에는 블록 기반 분산 파일 시스템을 이용한다. 블록 기반 분산 파일 시스템에서 파일들은 고정 크기의 블록 단위(기본 64MB)로 관리되며, 각 블록들은 내고장성 지원을 위해 적재 시 복제되어 각 블록들에 대해 기본적으로2개의 복사본(replica)들을 갖는다. 하나의 MapReduce 작업은 크게 Map()함수를 수행하는 단계와 Reduce()함수를 수행하는 두 단계로 구

성된다. Map 단계에서 입력 파일의 각 블록은 Map()를 수행하는 각 태스크(매퍼)에게 전달되고 각 매퍼는 Map() 함수를 각 블록 내 (*key, value*) 레코드들에 적용한다. Map() 수행 결과는 이를 수행한 각 노드의 로컬 디스크에 기록되는데, 이는 시스템 장애 시 중간 결과에 대한 내고장성 지원을 위한 것이다. Reduce()를 수행하기에 앞서 매퍼는 중간 결과에 대하여 *key2*를 기준으로 그룹화를 수행하는데, 이것은 *key2*값에 대한 정렬(sorting)을 통해 이루어진다.그리고 정렬을 통해 그룹화된 (*key2, value2*) 쌍들에 대하여 Reduce()를 수행할 태스크(리듀서)로의 전송에 앞서Combine 함수를 이용할 수도 있다. 이는 집계 연산을 Map 단계에서 미리 수행함으로써, 매퍼에서 리듀서로 데이터를 전달하는데 요구되는 I/O 비용을 감소시키기 위해서 선택적으로 이용된다.

리듀서들은 모든 매퍼가 종료된 시점에 HTTPS를 통해 각 매퍼들의 로컬 디스크에 기록된 중간 결과들을 가져온다. 이 때 각리듀서는 자신이 처리하도록 할당된 *key2*값에 대한 모든 값 리스트들을 모든 매퍼로부터 가져온다. 이 과정이 마치 카드를 섞는 것과 유사해 셔플링(shuffling)이라 명명하며, 이렇게 모인 각 키에 대한 모든 값들은 하나의 리스트로 병합(merge)된다. 이후 각 *key2* 값에 대하여 Reduce() 함수를 적용하고 그 결과를 분산 파일시스템에 기록한다.

이상과 같은 처리 흐름상에서 MapReduce는 다음과 같은 특징들을 갖는다[1][8][9].

1. MapReduce는 런타임 계획(runtime scheduling)에 기반한 태스크 수행이라는 특징을 갖는다. MapReduce는 어떤 태스크가 어떤 노드에 실행될지에 대한 수행 계획을 미리 지정하지 않는다. 태스크를 완료한 노드들은 다른 입력 블록을 할당 받는다. 이러한 구성은 더 빠른 노드가 더 많은 입력 블록을, 더 느린 노드는 더 적은 블록을 처리하게 함으로써 자연스럽게 작업 부하를 균등하게 한다. 또한 처리가 지연되는 노드의 태스크는 유휴한(idle) 다른 노드에 중복 할당하여 수행, 경쟁시키는 투기적 기법(speculative execution)도 이용한다. MapReduce 프레임워크에서 각 태스크는 다른 태스크와의 어떠한 의사소통도 필요로 하지 않는다. 이에 따라 태스크 간 동기화에 따른 성능 저하를 회피한다.
2. MapReduce는 각 노드가 동일 시점에 동일 작업을 다른 데이터들에 대해 수행하는 데이터 병렬화 (data parallelism)에 기반한다. 데이터 입력은 분산 파일 시스템에 적재 시 고정 크기의 블록으로 분할되며 각 매퍼는 동일한 Map() 함수를 자신에게 할당된 입력 블록의 레코드들에 적용한다.
3. MapReduce는 데이터를 태스크에 할당하는데 있어 데이터 이동(data shipping) 개념을 따른다. 예를 들어 매퍼의 출력을 리듀서에 전달하는데 있어 네트워크를 통한 데이터 복사가 이루어지며, 그 이전에는 로컬 디스크에 기록된다.
4. MapReduce의내부인 I/O 패턴은 외부 정렬-병합(external sort-merge)과해시-기반 분할(hash-based partitioning)이다. (key, value) 쌍들을 key값을 기준으로 그룹화하고 병합하기 위해 외부 정렬 기법을 이용한다. 또한 셔플링 시 각 리듀서에게 중간 결과들을 분할하는데 있어서 키 값에 대한 해싱 값을 통한 리듀서로의 할당(예, hash(key2) mod R, R은 리듀서의개수)을 수행하는 특징을 갖는다.

Ⅲ. MapReduce의 이점과 제약사항

Ⅱ 장에서 요약한 바와 같은 MapReduce의 구조적 특징은 기존 데이터 처리의 근간을 이루어왔던 DBMS와는 차이가 있다. 기존의 상용 DBMS들은 'one size fits all'[4] 전략으로 모든 데이터 저장/처리에 대한 솔루션을 DBMS 하나에 담고자 하였다. 이에 따라, 특정 데이터 처리, 특히 대용량 데이터 처리에 요구되는 병렬 데이터 처리에 있어서는 기존 DBMS는 적합한 솔루션으

로 인식되지 못해왔다. MapReduce는 이러한 데이터 처리에 관한 효과적인 솔루션으로 인식되어 널리 이용되었다. 하지만, 이와 같이하여 MapReduce는 그 효율성(efficiency)면에서는 상당한 논란이 있어왔다. DBMS 진영에서는 분산 DBMS와 비교하여 Hadoop이 데이터 분석 업무에 있어 2~50배 이상 더 느리다는 것을 지적하였다[5]. 특히, 각 노드당I/O 효율성 면에서는 매우 나쁜 성능을 기록하였는데[6], 이는 MapReduce가 기존 DBMS와는 반대되는 철학을 가지고, 내고장성과확장성(scalability)을 우선시 하여, 빈번한 디스크로의 데이터 저장(체크포인팅)과 데이터 이동 정책을 채택한 데에 기인한다. 즉, MapReduce는 I/O 효율성의 희생을 통한 내고장성과 확장성 확보를 우선하였다 할 수 있다. 이런 구조적 차이에 따른 MapReduce의 이점과 제약 사항을 요약하면 다음과 같다.

1. MapReduce의 이점

- 단순함과 편리함 Map과 Reduce두 함수 의 구현을 통한 자동화된 병렬 처리를 지원함으로써 모델이 단순하고 개발이 용이한 장점이 있다. 많은 병렬 처리 문제들이 두 함수의 구현을 통해 손쉽게 해결이 가능하다.
- 유연성 MapReduce는 데이터 모델이나 스키마 정의를 요구하지 않으므로, 평문과 같은 비구조적 데이터 처리를 DBMS 에서보다 쉽게 다룰 수 있다. 뿐만 아니라, 스키마가 없으므로 단순히 분산 파일 시스템에 파일을 올리는 것으로 데이터 적재가 완료되므로 대량의 데이터를 한 번 처리하는 경우에 효과적이다.
- 저장 계층과의 독립성 MapReduce는 저장 계층과의 독립성을 보장하므로 분산 파일 시스템 이외의 어떠한 저장 시스템 위에서도 동작할 수 있다.
- 내고장성과 확장성 MapReduce는 강력한 내고장성을 제공한다. Google의 경우 평균적으로 한 MapReduce작업 당 1.2 개의 장애를 경험하는데,이때도 MapReduce 작업은 종료되지 않고 지속적으로 수행, 완료된다[2]. 이러한 내고장성에 기반하여 MapReduce는 매우 높은 확장성을 갖는다. 예를 들어, 2008년 Yahoo! Search Webmap의 경우 10,000 대 이상의 Linux 상에서 동작하는 Hadoop 응용이었다.

2. MapReduce의 제약사항

- 고차원 언어의 부재 MapReduce는 DBMS에서의 SQL(Structured Query Language)와 같은 고차원 언어와 질의 계획 작성에 따른 질의 최적화 기술을 지원하지 않는다.이에 따라 코드의 재사용이 어려운 한계가 있다.
- 데이터 모델과 스키마의 부재 데이터 모델이 없다는 점은 처리할 데이터의 모델에 관계없이 유연하게 프로그램을 개발할 수 있다는 장점과 동시에 데이터 모델링의 이점을 버리는 것이다. 스키마가 존재하지 않으므로 데이터 입력 시 각 아이템의 무결성(integrity) 검사를 위해 매 입력 처리 시 각 레코드에 대한 파싱이 요구되며 이는 반복된 데이터 처리 시 성능 저하를 야기한다.
- 고정된 데이터 흐름 MapReduce는 단일 입력과 단일 출력을 갖는 알고리즘의 병렬 처리를 목적으로 고안 되었다. 이에 따라 복잡한 알고리즘들은 하나의 MapReduce 작업으로 구현되기 어렵다. 대표적인 경우로는 1)조인 연산처럼 여러 개의 입력을 갖는 다항 연산자, 2)루프문과 같이 반복적인 데이터 처리 등이 이에 해당한다. 또한 MapReduce는 데이터의 일괄 작업 처리(batch processing) 목적으로 고안되어, 처리 전에 데이터와 작업이 모두 미리 준비되어 있어야 한다.
- 낮은 효율성 MapReduce는 내고장성과 확장성의 지원을 목적으로 잦은 체크포인팅(checkpointing)과 데이터 전달 정책을 채택하였다. 이에 따라 데이터

처리 중에 많은 디스크와 네트워크 I/O를 발생시킨다. 또한, 데이터의 그룹화를 위해 외부 정렬-병합 기법을 채택함에 따라 Map과 Reduce는 이전 단계가 종료되기 전에는 다음 단계의 작업을 처리할 수 없으므로 파이프라인 병렬화(pipeline parallelism)를 실현시킬 수가 없다. 또한 블록 단위의 재시작과 단순한 런타임 수행계획 등으로 인해 이문제가 더욱 악화되는 경향이 있다.

Ⅳ. 개선 연구들

이 장에서는 Ⅲ장에서 언급된 MapReduce의 제약 사항들을 극복하기 위한 MapReduce의 기능과 성능을 개선하기 위한 최근의 연구 내용들과 개량 시스템에 대해서 세부 항목별로 소개한다.

1. 고차원 언어의 지원

MapReduce를 위한 고차원 언어들은 해당 언어로 작성된 스크립트의 MapReduce 작업으로의 자동화된 변환을 지원하며, 크게 선언적 언어(declarative language)와 데이터플로우 형태의 언어로 구분된다. 선언적 언어로는 Hadoop기반의 DW 도구인 Hive에서 사용하는 HiveQL과 또한 JAQL, Tenzing등이 존재하며, 데이터플로우로는 Apache Pig가 존재한다. 이중 가장 대표되는 두 언어는 Hive와 Pig이며 둘의 차이는 〈표 1〉에서 보인다.

〈표 1〉Hive와 Pig의 비교

	Hive	Pig
언어	SQL과 유사한 선언적 언어	데이터플로우
데이터 모델	Nested	Nested
UDF	지원	지원
데이터분할	지원	미지원
질의최적화	규칙 기반	규칙 기반
메타데이터	지원	미지원

2. 스키마의 지원

이전 장에서 언급한 바와 같이 MapReduce는 스키마 지원을 하지 않는다. 하지만, Google의 Protocol buffer나 XML, JSON, ApacheThrift와 같은 포맷으로 입력 데이터를 구성함으로써 이 문제를 해결한다. 이들 포맷은 중첩, 반구조적 데이터 모델을 지원하며, 자기기술적(self-describing) 포맷이므로 추가적으로 데이터 크기가 크게 증가할 수 있다. 이에 따라 데이터 압축이 같이 사용되기도 한다.

3. 보다 유연한 데이터 흐름의 지원

가. 반복 연산에 대한 효과적 처리

많은 알고리즘들은 루프 연산의 수행을 요구하나 MapReduce는 루프 연산을 효과적으로 처리하지 못한다. MapReduce는 루프의 실행과 종료를 위한 상태 정보를 기록하지 못하며, 매 루프마다 동일한 데이터를 처음부터 읽고 처리함으로써 I/O 효율성도 낮다. 이러한 문제를 해결하기 위해 고안된 시스템으로는 HaLoop, Twister, Pregel, PrIter등이 존재한다.

HaLoop과 Twister는 반복 연산 처리를 위한 대표적인 시스템으로 루프 수행 동안에 불변의 데이터와 변동하는 데이터를 확인하고, 불변의 데이터들은 한번 읽은 뒤캐싱(caching) 방식으로 반복 읽기를 피한다. 또한, 매 루프마다 같은 임무를 수행하는 Map 또는 Reduce 태스크가 반복해서 생성되었다 종료되는 것을 막음으로써 성능향상을 꾀한다.

Pregel의 경우 많은 반복 연산을 필요로 하는 Page-Rank와 같은 그래프 데이터 처리를 목적으로 한다. Pregel은 BSP(Bulk Synchronization Processing)모델에 기반한 것으로 각 노드는 각각의 입력을 가지고 있으나 다음 태스크로의 데이터 전송은, 단지 다음 반복 연산을 위해 요구되는 메시지만을 전송하는 방식을 취한

다. PrIter는 반복문 처리에 있어 각 데이터 처리 시점에 대한 우선순위를 부여함으로써 선택적인 데이터 처리를 가능하게 한다.

나. 파이프라이닝및 즉각적인 결과 출력

MapReduce에서는 각 단계의 모든 태스크가 종료되어야만 다음 단계로 진입할 수 있다. 하지만, 정확한 결과값 보다는 근사치에 대한 빠른 응답이 더 요구되는 경우도 존재한다. MapReduce Online은 온라인 집계와 연속 질의 처리를 지원하는 MapReduce의 개량형으로 각 매퍼가 중간 결과들을 태스크 종료 전에도 미리 리듀서에 보내는 방식으로 변형하였다. 이를 통해 리듀서는 모든 Map태스크가 종료되지 않은 상태에서도 대략적인 집계 결과를 내놓을 수 있다. SCALLA는 MapReduce의 이런 제약이 정렬-병합에 기반한 그룹화에 기인한 문제임을 지적하고, 이를 해싱(hashing)으로 대체함으로써 각 매퍼의 결과를 바로 해시 테이블에 넣고 이에 대한 온라인 집계를 수행하는 방식으로 즉각적인 결과를 출력하도록 하였다.

다. 자유로운 데이터처리 흐름

MapReduce는 단일 입력을 처리하도록 고안되었지만, 조인과 같은 이항연산자들은 두 개의 입력을 요구한다. Map-Reduce-Merge는 reduce 단계 후에 merge 단계를 추가함으로써, 하나의 MapReduce 작업으로 조인을 처리하도록 데이터흐름을 개선하였다.

Clustera, Nephele/PACT, Dryad, Storm, S4 등의 시스템은 아예 사용자가 임의로 자유롭게 데이터 흐름을 정의하고 처리하도록 고안된 MapReduce와는 별도의 시스템들의 예이다. 이들은 한 작업을 DAG(Directed Acyclic Graph)형태 또는 복잡한 연산을 위한 파이프라인 형태로 구성하고, 각 노드에서는 DAG의 한 정점(vertex)에 정의된 연산을 처리하는 방식으로 병렬처리를 지원한다.

4. I/O최적화

MapReduce는 I/O 효율성 대신에 내고장성과 확장성을 고려하여 설계된 시스템으로 매우 높은 I/O 비용을 요구한다. 이에 따라 많은 연구들이 데이터 처리 시의 I/O 비용을 낮추기 위해 수행되었으며, 이들을 분류하면 아래와 같다.

가. 컬럼 단위 데이터 배치

컬럼 단위 데이터 배치(columnar data layout)기법은 OLAP질의 처리 시 불필요한 컬럼의 읽기를 피하기 위해 튜플들을 기존 행 단위(row-wise)가 아닌 컬럼 단위(column-wise)로 저장하는 방식이다[7]. 이 기법은 OLAP 질의 처리와 같은 데이터 분석 업무에 있어 I/O 성능 향상에 큰 이점이 있어 이에 대한 많은 연구가 수행되어 왔다. Llama, Cheetah, CIF, RCFile, Trojan 데이터 배치 기법등의 연구들이 MapReduce를 위한 컬럼 단위 데이터 배치 기법에 관한 대표적인 연구들이다.

Llama는 CFile라 불리는 컬럼 단위의 데이터 배치를 위한 파일을 이용한다. Llama에서 데이터는 여러 수직 그룹으로 분할되고 각 그룹의 데이터는 임의 컬럼 값을 기준으로 정렬되어 CFile로 저장된다. Cheetah는 컬럼 단위 배치 이외에 블록 단위의 여러 압축 기법을 지원한다. Cheetah는 PAX 배치 기법[7]을 블록 단위로 적용하였다. 이에 따라 각 블록에는 임의 행들이 포함할 컬럼 값들이 컬럼 단위로 나열된다. RCFile 또한 PAX 배치 기법과 유사한 방식을 취하는 컬럼 단위 배치 기법이지만, 블록 단위의 압축은 제공하지 않으며, 수직 분할된 컬럼 값들에 대해서만 압축을 허용한다. RCFile은 Hive, Pig 등의 시스템에서 현재 데이터 배치 기법으로 널리 이용되고 있다. CIF 또한 컬럼 단위 배치 기법으로,파일을 먼저 수평 분할 한 후, 한 파티션 안의 컬럼들

을 분할하여, HDFS(Hadoop Distributed File System) 상의 한 디렉토리에이진 파일들로 저장한다. Dremel의 경우는 중첩 데이터 모델을 지원하는데 데이터를 컬럼 기반 배치 기법에 따라 분할하고 이에 대한 임의의 질의(ad-hoc query)를 지원하도록 고안된 Google의 시스템이다. Trojan 데이터 배치기법은 복제 블록의 생성 시 복제 블록 내에서의 데이터 배치를 행 단위, 컬럼 단위, 그 외 여러 혼합으로 하게 하고, 질의 처리 시 이들 중에서 효과적인 데이터 배치를 선택하도록 한다. 이외에 최근에는 오픈 소스 진영에서 Parquet과 Trevni와 같은 컬럼 단위데이터 배치 기법을 구현한 라이브러리가 존재하는데 이는 각각 Dremel, CIF를 참조하여 구현된 것이다.

추가적으로, Google의 BigTable은 HDFS에 기반한 새로운 유형의 NoSQL솔루션으로 여러 컬럼들을 '컬럼패밀리'로 명명된 하나의 그룹으로 묶어 기본 작업 단위로 한다. CoHadoop은 HDFS 상에서의 블록 배치에 있어 서로 관계되는 데이터들을 동일 노드 상에 같이 배치(collocation)하여 데이터 처리 시 네트워크 비용을 줄이고자 고안되었다.

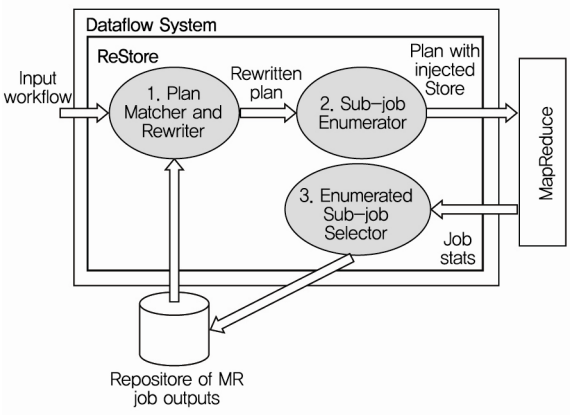
나. 인덱싱

MapReduce는 기본적으로 대량의 데이터에 대한 단일 처리를 목적으로 하지만, 동일 연산을 반복적으로 행해야 할 경우도 존재한다. 이 경우 인덱싱을 통해 추후 데이터 읽기 연산 시 많은 I/O를 절약할 수가 있다. Hadoop^{++}은 UDF를 이용하여 HDFS에 저장된 데이터에 대한 인덱싱을 Hadoop 프레임워크의 변경 없이 지원한다. Hadoop^{++}은 Trojan 인덱스라 명명된 일종의 커버링 인덱스를 데이터 블록에 같이 삽입시킨다. HAIL은 Hadoop^{++}에서의 긴 인덱스 생성시간을 단축시키기 위해 데이터 적재 시 데이터 복제에 있어 유휴한 CPU 사이클을 이용하여 각 복제 블록에 대하여 인덱스를 생성하도록 고안되었다. 이 인덱스는 각 복제 블록에 대하여 서로 다른 순서로 정렬되는 클러스터드 인덱스(clustered index)로, 질의 처리 시 질의에 가장 적합한 클러스터드 인덱스를 선택하는 기능도 같이 제공한다.

다. 중복의 제거/데이터 및 연산의 공유

MapReduce의 이용에 있어 많은 연산들은 서로 유사하거나 같을 수 있다. 이러한 연산들을 각기 다른 MapReduce작업으로 구현하는 경우 동일하거나 유사한 데이터에 대하여 반복적으로 같은 연산을 수행하게 되는데, 이는 자원의 낭비 및 처리 시간의 지연을 야기한다. 이러한 문제를 해결하기 위하여 여러 기법들 또한 고안되었다. 이를 종류 별로 나열해 보면, 1) 반복 연산에 대한 효과적 처리 2) 다중 질의 최적화(multi-query optimization)를 통한 여러 질의들 간의 데이터 및 처리 공유 3) 실체화된 뷰(materialized view) 기법을 이용한 처리 결과의 재활용으로 구분해 볼 수 있다. 이 중 1) 반복 연산에 대한 효과적 처리는 4.3절에서 소개한 바와 같이 데이터 처리 흐름을 변경하는 방식을 이용하고 있으며, 이 절에서 2), 3)의 내용에 대해 주로 설명하도록 한다.

MRShare는 MapReduce 작업들 간의 하부 태스크들 간의 공유를 통한 성능 개선을 목표로 고안된 시스템이다. MapReduce 작업 수행 시 질의들은 미리 준비될 수 있는 특징을 고려하여, MRShare는 등록되는 MapReduce 작업들 간에 중복되는 공통 패턴들을 동적 프로그래밍 기법을 통해 질의 처리 이전에 미리 그룹화하고 각 그룹에 대해서 한 MapReduce 작업만을 수행하는 방법으로 중복 처리를 방지한다. MRShare에서는 3가지 형태의 공유 형태를 제안하였는데, 데이터 입력 스캔의 공유와 Map 함수 공유, 그리고 중간 결과의 공유가 가능함을 보였다. 또한 입력 스캔의 공유는 언제나 좋은 성능을 보이지는 못한다는 것 또한 밝혔는데. 이는 MapReduce의 정렬-병합에 기반한 그룹화에 기인한 것이다. 예를 들어 |D| 크기를 가지는 입력 데이터에 대하



(그림 2) ReStore의 시스템 구조

여 n개의 작업을 수행하는 경우, 입력 스캔의 공유 전에는 |D| · n만큼의 읽기 I/O가 요구되고, 대신에 n · (|D|log|D|)만큼의 정렬 연산이 소비된다. 반면, 입력 스캔 공유 시에는 |D|만큼의 읽기 I/O가 소비되는 반면, 전체 중간 결과의 크기는 최대 n · |D| log (n · |D|)으로 오히려 커질 수 있음을 보였다.

ReStore는 실체화된 뷰(materialized view) 기법에 기반한 저장소를 포함하도록 Pig 시스템을 개량한 것이다((그림 2)참조). ReStore에서는 한번 MapReduce 작업을 수행한 후, 이를 다음 MapReduce 작업을 처리하는데 있어 그 작업을 실제 처리하지 않거나 또는 일부만 처리해도 되도록 이전 처리 결과를 해당 MapReduce 작업의 물리적 실행계획과 같이 저장한다. ReStore는 MRShare와 다르게 서로 다른 시점에 작성되어 입력된 개별 질의에 대하여 작업 결과를 공유할 수 있도록 허용한다. 이에 반해, MRShare는 병행 처리되어야 할 일련의 작업들에 대하여 공유의 최적화를 목표로 한다.

5. 스케줄링과 균등 부하

MapReduce는 블록 단위의 런타임 스케줄링을 이용한다. Hadoop 스케줄러는 단순히 각 태스크의 진행 정도와 평균 진행 정도를 비교하는 휴리스틱한 방법을 이용하여 스케줄링 한다. 하지만, 이 방법은 각 노드가 각기 다른 컴퓨팅 파워를 갖는 경우에는 적합하지 않다. 이 문제를 해결하기 위하여 LATE(Longest Approximate Time to End) 스케줄링 기법이 고안되었다. 이 방법에서는 단순한 진행 정도가 아닌 진행 속도를 판단하여 결정한다.

효과적인 스케줄링 기법은 각 노드에 할당되는 작업 부하를 균등하게 한다. 특히 MapReduce에서는 각 단계에서 모든 태스크가 종료되어야만 다음 단계로 전이될 수 있으므로, 이의 지원이 보다 중요하다. 이를 지원하는 방법은 크게 1) 샘플링에 기초한 선행(preprocessing)과 2) 데이터의 재분할(repartitioning)이다. 샘플링에 기초한 선수행 방법은 입력 데이터의 일부만을 먼저 처리해 보고, 이를 기초로 데이터의 태스크 할당을 조정함으로써 작업 부하의 균등을 꾀한다. 데이터 재분할 기법은 Skew Tune 등에서 적용한 기법으로 각 태스크가 할당 받은 데이터를 처리하는데 있어, 데이터를 재분할하는 방식으로 실행 중에 작업 부하의 균등화를 꾀하는 기법이다. 이 기법에서는 먼저 임의 태스크가 할당된 데이터를 다 처리하여 유휴해진 경우 스케줄러에게 이 사실을 알리고, 스케줄러는 다른 태스크의 진행 속도를 점검하여 이들이 아직 처리하지 못한 데이터를 휴리스틱하게 유휴 태스크에 할당하는 방식으로 균등화를 꾀한다.

6. 관계형 연산의 지원

셀렉션이나 프로젝션과 같은 관계형 연산자들은 Map 단에서 쉽게 구현이 가능하다. 하지만 조인 연산의 경우 단항 연산자가 아닐뿐더러 응용의 성능에 지대한 영향을 미치므로 이에 대한 많은 연구가 진행되어 왔다.

가. 이항 조인 연산자

MapReduce에서 조인은 실제 조인 연산이 수행되는 위치에 따라 Map 단에서의 조인과 Reduce 단에서의 조인으로 나뉜다. Map 단에서의 조인은 크게 Map-

merge 조인과 Broadcast 조인 기법이 있고, Reduce 단 조인은 Repartition 조인이 대표적이다.

Map-merge 조인은 가장 단순한 Map 단의 조인으로 두 입력 릴레이션들이 먼저 조인 키 값에 의해 모두 정렬되어 있을 때 이들을 읽으면서 바로 Map단에서 조인을 수행하는 방법이다. 이 방법은 데이터가 미리 정렬되어 있어야 하는 제약을 가지고 있다. 반면 Broadcast 조인 기법은 한 쪽 릴레이션의 크기가 현저히 작은 경우에 유용한 방법으로 크기가 작은 릴레이션은 수행 전에 미리 각 매퍼마다 복사 할당 한다. 그리고 각 매퍼는 이들을 가지고 메모리 상의 해시테이블을 구축한 다음 입력 데이터들에 대하여 해당 해시테이블을 조회하는 방법으로 조인을 수행한다.

Repartition 조인은 MapReduce에서 조인 연산 구현 시 가장 널리 쓰이는 기법 중 하나로 각 매퍼는 해당 튜플이 어느 릴레이션에 속하는지에 대한 태깅을 수행한다. 이 후 키 값에 의해 정렬, 그룹화된 튜플들이 리듀서에 모이면 리듀서는 태그 값으로 튜플들을 구분하고 키 값을 기존으로 조인을 수행한다. 이 방법은 DBMS에서의 해시 조인과 유사하며, 동등 조인 연산의 구현이 있어 효과적이다. θ -조인의 경우에는 조인에 참여할 두 릴레이션의 부분 집합들을 리듀서들에 어떻게 균등 분배할 것인가가 문제가 되는데, 1-Bucket-Theta 알고리즘 등이 이에 대한 해결책으로 제시되었다.

나. 다방향 조인

다방향 조인(multi-way join)은 조인 연산에 참여하는 릴레이션이 2개가 넘는 조인 연산이다. OLAP 환경에서의 스타스키마 조인, XML에서의 가지 패턴 조인 등이 대표적인데 이의 연산은 크게 1) 이항조인 연산(binary join operator)들을 조합, 이들을 여러 번 수행하여 얻는 방법과 2) 다방향 조인 연산 자체를 고안하여 지원하는 방법으로 나뉜다. 먼저 이항 조인 연산자를 조합하여 하는 경우에는 각 이항 조인 연산자들이 하나 또는 복수 개의 MapReduce 작업들로 구현되므로, 전체 조인 연산은 여러 MapReduce 작업들이 체인 형태로 연결되는 모습을 갖는다. 이 때 전체 조인 연산의 비용이 최소가 되도록 하기 위해서는, 조인 적용 순서를 효과적으로 결정하면서 이와 동시에 이를 구현하는데 요구되는 MapReduce 작업의 수가 최소가 되도록 해야 한다. 이에 대한 논의는 Wu 등의 연구[10]에서 다루어지고 있다.

다방향 조인 연산자를 하나의 연산자로 구현하는 경우에는 MapReduce 작업에서 셔플링 과정의 변형이 요구된다. 왜냐하면 각 리듀서가 조인을 수행할 릴레이션들을 서로 분할하여 가져야 하는데, 이때 릴레이션이 3개 이상이면 일부 릴레이션의 중복이 없이는 분할 할당이 곤란하기 때문이다. 기본적으로 MapReduce 작업에서 셔플링은 매퍼에서 리듀서로의 일대일 할당이므로 복사를 통한 중복 할당을 위해서는 이를 수정해야 한다. 이 때, 리듀서들에 복사, 할당되는 데이터의 양을 최소로 하면서, 리듀서들이 서로 균등하게 조인 연산을 할당하는 것이 중요하다. Afrati등[11]은 이에 대한 해결책으로 라그랑지승수법(Lagrangianmultiplier)에 기반한 비선형계획법으로 최적화를 수행하였다.

V. 최근의 대용량 데이터 솔루션

MapReduce의 출현은 I/O 효율성을 목표로 해왔던 기존 DBMS와 이를 중심으로 한 데이터 솔루션에 많은 영향을 끼쳐왔다. DBMS 진영에서는 비정형 데이터 분석에 있어 MapReduce의 장점을 포함시키고자 SQL/MapReduce와 같이 UDF를 MapReduce 작업으로 구현할 수 있도록 하는 방안을 제시하였다. 또한 MapReduce의 하위 저장소를 열-기반 DBMS로 대체함으로써 DBMS가 가지는 질의 처리 성능의 이점과 MapReduce의 확장성과 내고장성을 통합시키려는 시도도 있다. 대표적인 시스템으로는 HadoopDB와 Osprey 등이 존재한

다. 최근에는 실시간 데이터 처리의 필요성이 보다 대두됨에 따라, MapReduce의 배치 형식의 데이터 분석에서 기존 스트림 처리 시스템의 분산화를 통한 대용량 데이터의 실시간 처리와 인메모리 기반으로 임의 질의 처리가 가능한 데이터 분석 시스템으로 그 연구 방향이 변하고 있다. 분산 스트림 처리 시스템은 기존 스트림데이터 처리에서 이용되던 DSMS들을 모아 임의로 토폴로지를 구성하고 이를 통한 데이터 처리의 병렬화를 수행한다. 대표적인 분산 스트림 처리 시스템으로는 Storm, Apache Kafka, Yahoo의 S4 등이 존재한다.임의 질의 처리가 가능한 데이터 분석 시스템으로는 Cloudera의 Impala와 Apache 인큐베이터 프로젝트인 Spark과 Tajo등이 존재한다. Spark의 경우는 독자적인 분산 메모리 구조에 기반하는 인메모리 데이터 분석을 지원하는 시스템이고, Impala와 Tajo는 Hadoop의 HDFS 위에 MapReduce 대신 임의 질의를 지원하는 DW 시스템의 모습을 취하고 있다. 주목할 점은 오픈소스 진영의 이러한 시도는 HDFS라는 분산 파일 시스템에 기반하는 DW 시스템으로 그 기능이 기존 열-기반 DBMS와 거의 동일하다는 점이다.

VI. 결론

MapReduce는 I/O효율성을 희생하여 우수한 확장성과 내고장성을 지원한다. 반면에, DBMS는 초기 OLTP를 목적으로 하여 임의 질의의 지원과 파이프라인 병렬성에 기초한 I/O 효율성 등이 장점이나 분산 구조로의 확장성 등에 있어서는 고가의 라이선스 비용 등으로 인해 강점을 가지지 못하고 있다. MapReduce와 Hadoop에 기반한 데이터 웨어하우스 시스템들은 OLAP 질의 처리를 목적으로 기존 OLTP 시장을 잠식하지는 못할 것이라 여겨지지만 OLAP과 비정형 데이터 분석 부분에서는 Sybase와 같은 기존 DBMS들과 경쟁을 할 것이라 여겨진다. 더불어 더욱더 많고 복잡한 포맷들을 지원하는데 있어 유연한 특징을 지원하는 MapReduce 기반 시스템들은 그 위치를 더 공고히 할 것으로 예측된다. 다만, 오픈소스 정책에 기반한 이런 시스템들은 운영에 있어 코드의 미성숙 또는 내재화된 기술력의 부재로 인해 드는 비용은 추가적인 부담이 될 여지가 있다.

용어해설

Hadoop Apache 프로젝트로 관리, 유지되는 오픈 소스 SW로 Google의 MapReduce와 GFS를 Java로 구현하였음. Hadoop은 HDFS와 Hadoop MapReduce로 구성됨. MapReduce와 GFS는 Google이 논문만 공개했을 뿐 코드는 공개하지 않아왔으며, 대신 Hadoop이 널리 이용되어 왔음. 본고의 모든 설명은 Hadoop을 기반으로 작성됨. 다른 언어로 구현된 Map Reduce버전으로는 Skynet(Ruby), FileMap(Perl), Disco (Python과 Erlang) 등도 존재하나, Hadoop이 월등히 많이 선택되고 있음.

약어정리

BSP	Bulk Synchronization Processing
DAG	Directed Acyclic Graph
DBMS	Data Base Management System
HDFS	Hadoop Distributed File System
HTTPS	Hyper Text Transfer Protocol Secure
LATE	Longest Approximate Time to End
OLAP	On-Line Analytical Processing
OLTP	On-Line Transactional Processing
SQL	Structured Query Language
UDF	User Defined Function

참고문헌

[1] K.-H.Lee et al., “Parallel data processing with MapReduce: a survey,” SIGMOD Record, vol. 40, no. 4, 2011, pp. 11-20.

[2] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of ACM, vol. 51, no. 1, 2008, pp. 107-113.

[3] Apache projects, Hadoop, http://Hadoop.apache.org, 2009.

[4] M. Stonebraker et al., “One size fits all? Part2: Benchmarking results”, *Proc.CIDR*, 2007.

[5] A. Pavlo etal., "A comparison of approaches to large-scale data analysis" *Proc. ACM SIGMOD*, 2009.

[6] E. Anderson and J. Tucek "Efficiency matters!," *ACM SIGOPS Operating. Syst. Review*,vol. 44, no. 1, 2010, pp. 40-50.

[7] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, "Column-oriented database systems," *Proc. VLDB*, 2009, pp. 1664-1665.

[8] C. Doulkeridis et al., "A Survey of Large-Scale Analytical Query Processing in MapReduce," *The VLDB Journal (to appear)*.

[9] Feng Li et al., "Distributed Data Management using MapReduce," *ACM Computing Surveys (to appear)*.

[10] S. Wu et al., "Query optimization for massively parallel data processing,"*Proc. 2nd ACM Symposium on Cloud Comput.*, 2011.

[11] F. N. Afrati and J. D. Ullman "Optimizing joins in a map-reduce environment," *Proc. 13th EDBT*, 2010.