신기술해설

하둡 프로그래밍을 위한 순환 처리 기술

최동훈·최윤수·이원구·이민호·윤화묵·정한민 (한국과학기술정보연구원)

목 차 »

하둡(Hadoop)은 맵리듀스(MapReduce)를 공개 소스로 구현한 것으로 맵리듀스 프레임워크와 하둡 분산 파일 시스템으로 구성된다. 하둡은 하나의 맵 단계와 하나의 리듀스 단계를 지원하는 단순한 병렬 프로그래밍 모형으로, 순환 처리 메커니즘이 없다. 그러나 과학 어플리케이션이나 데이터 마이닝에서 많이 사용하고 있는 기계 학습이나 진화적 알고리즘은 데이터 분할에 의한 병렬 처리 기법을 사용할 수 있으면서 순환 처리를 포함하고 있기 때문에, 하둡이 순환 처리를 지원해야 한다는 데에 많은 연구자들이 동의하고 있다. 이 밖에 순환 처리를 요구하는 어플리케이션으로 웹 그래프 분석, 클러스터링, 신경망 분석, 소셜 네트워크 분석, 바이오 네트워크 분석 등 매우 다양하다.

하둡의 순환 처리를 가능하게 하기 위한 접근 방법으로 두 가지가 있다. 하나는 하둡 실행 시스템을 확장 개선하는 방법으로, 실행 시스템 내부에서 순환 처리를 직접 지원하는 것이다. 다른 방법은 하둡의 실행 시스템 내부를 수정하지 않고 하둡 외부에서 순환 처리 드라이버를 사용자에게 제공한다. 즉, 사용자가 순환 처리 드라이버를 사용하여 다수의 맵리듀스 작업을 생성하고 이들의 실행을 통제하는 방식으로 순환 프로그램을 구현하는 것이다. 전자의 대표적인 예가 하룹(HaLoop)이고, 후자의 대표적인 예로 트위스터(Twister)가 있다. 하룹은 순환 처리의 성능 향상을 기대할 수 있으나, 하둡의 장점으로 부각되었던 확장성(scalability)과 결함 내성(fault tolerance)을 저해하는 문제가 있다. 트위스터는 순환 처리의 성능에 문제가 있을 수 있으나 충분한 인-메모리(in-memory)를 전제로 하면 성능 문제를 완화할 수 있으며, 많은 데이터를 처리하는 과학 응용에 특화되어 있는 것이 장점이다. 본고에서는 공개 소스로 알려진 트위스터[1]와 하룹[2]을 소개한다.

1. 트위스터

1.1 트위스터의 구조

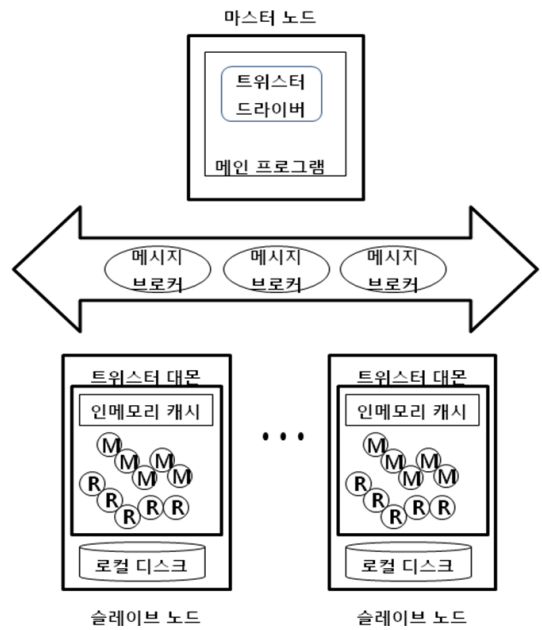
트위스터는 하둡을 기반으로 개발된 것이 아니라 인디아나 대학교에서 자체 개발한 맵리듀스

런타임 시스템이다. 트위스터는 순환 맵리듀스를 계산하기 위해 최적화된 분산 인-메모리 맵리듀스 실행 시스템으로, 슬레이브 노드의 로컬 디스크로부터 데이터를 읽어서 슬레이브 노드의 분산 메모리에서 임시 데이터를 처리한다. 노드 간의 모든 통신과 데이터 이동은 메시지 큐(message queue)와 같은 publish/subscriber 메시징 기반 구조를 통해서 이루어진다.

트위스터는 맵리듀스 계산 전체를 추진시키는 트위스터 드라이버, 모든 슬레이브 노드에서 동작하는 트위스터 대몬(daemon), 그리고 메시지 큐 기반의 브로커 네트워크, 이렇게 세 가지 주요 요소로 구성되어 있다. 실행 시스템을 초기화하는 동안에, 트위스터는 각 슬레이브 노드에 대몬 프로세스를 개시시키고, 대몬 프로세스는 명령어와 데이터를 받기 위해 브로커 네트워크에 연결을 설정한다. 대몬은 그에 할당된 맵과 리듀스 타스크의 관리, 맵과 리듀스 타스크를 실행하기 위한 슬레이브 노드 풀의 유지, 상태 알림, 통제 이벤트에 대한 대응 등의 역할을 수행한다. 트위스터 드라이버는 사용자에게 API를 제공하며 트위스터 API를 통제 명령어와 입력 데이터 메시지로 변환하여 브로커 네트워크를 통해 슬레이브 노드에서 동작하는 대몬에게 전송한다.

트위스터는 publish/subscribe 메시징 기반 구조를 사용하여 통제 이벤트의 전송, 트위스터 드라이버와 트위스터 대몬 간의 데이터 전송, 맵(Map)과 리듀스(Reduce) 타스크 간의 데이터 전송, 리듀스 타스크의 출력 데이터의 트위스터 드라이버로 전송 등의 통신 수요를 만족시킨다. (그림 1)은 트위스터 실행 시스템의 구조를 나타낸 것이다.

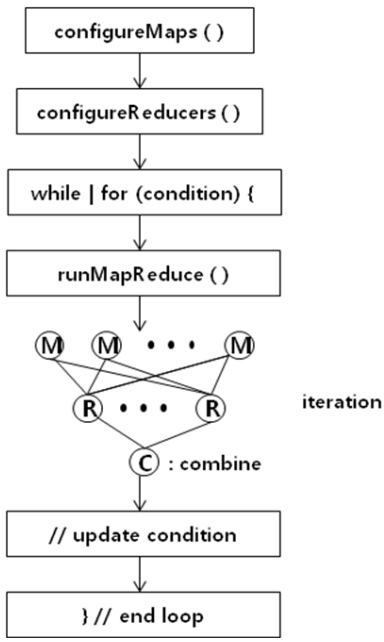
트위스터의 데이터 입출력은 하둡과 달리 하둡 분산 파일 시스템을 지원하지 않는다. 슬레이브 노드의 로컬 디스크로부터 읽고 쓰거나, 브로커 네트워크로부터 읽고 쓴다. 반면에 성능을 높이기 위해 트위스터는 임시 데이터를 슬레이브 노드의 분산 메모리에서 처리한다.



(그림 1) 트위스터의 구조

1.2 트위스터의 순환 맵리듀스 프로그래밍 모형

과학 분야에서 순환 구조의 어플리케이션이 생성하는 데이터를 살펴 보면, 입력 데이터에 대한 반복 계산을 수행하면서 순환 단계마다 변하는 부분과 순환 단계를 거쳐도 불변하는 부분으로 구분된다. 일반적으로 불변 데이터는 모든 순환 단계에서 사용되면서도 계산을 거쳐도 고정된 값을 갖는 반면에, 변하는 데이터는 각 순환 단계마다 새로 계산된 값을 가지며 다음 순환 단계에서 순환 구조를 통제하기 위한 조건을 구하는데 사용된다. 트위스터는 맵리듀스 타스크가 이러한 두 가지 유형의 데이터를 효율적으로 적재하기 위해, 맵과 리듀스 타스크를 위한 불변 데이터 설정(configure) 단계를 제공한다. (그림 2)는 트위스터



(그림 2) 트위스터 프로그래밍 모형

가 지원하는 순환 맵리듀스 프로그래밍 모형을 나타낸 것이다.

트위스터가 제공하는 API는 다음과 같다.

- configureMaps: Map 타스크를 위해 불변 데이터를 읽어 들여 로컬 디스크에 분할한다.
- configureReduce: Reduce 타스크를 위해 불변 데이터를 읽어 들여 로컬 디스크에 분할한다.
- runMapReduce
- runMapReduceBCast: 단일 값을 모든 Map 타스크에 전송한다
- map
- reduce
- combine

1.3 타스크 스케줄링

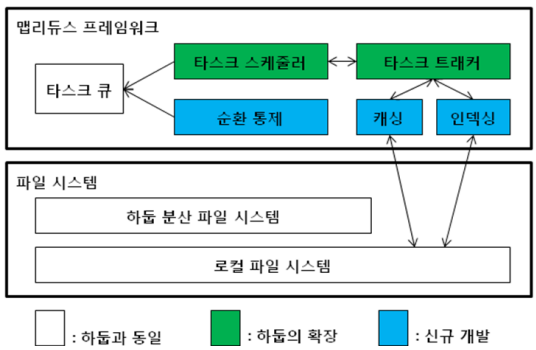
트위스터의 맵리듀스 타스크는 각 데이터의 위치가 고정되어 있을 때만 효율적이기 때문에, 트위스터의 스케줄러는 고정형 타스크 스케줄링을 지원하도록 개발되어 있다. 그러나 슬레이브 노드의 결함 발생 시, 다른 노드 클러스터에서 계산을 다시 스케줄링한다. 이러한 고정형 스케줄러는 입력 데이터의 분포가 고르지 않을 때 자원 이용률이 떨어지지만, 맵 타스크에 입력 데이터를 임의로 할당하여 이러한 효과를 최소화할 수 있다.

2. 하 룹

순환 처리 드라이버를 사용하여 수동으로 순환 프로그램을 구현하는 것은 크게 두 가지 문제점이 있다. 첫째 문제는 각 순환 단계를 거치면서 많은 데이터가 변경되지 않더라도 순환 단계마다 이들 데이터를 다시 적재하고 다시 처리해야 한다는 것이다. 이것은 IO 대역폭, 네트워크 대역폭, CPU 자원 등의 낭비를 초래한다. 둘째 문제는 순환을 종료하려면 언제 어플리케이션의 출력 데이터가 부동점(fix point)에 이르는지 탐지해야 하는데, 이 조건의 탐지는 각 순환 단계마다 별도의 맵리듀스 작업을 요구한다는 것이다. 이것은 역시 별도의 작업의 스케줄링, 디스크로부터 별도의 데이터 읽기, 네트워크 간의 데이터 이동 등에 따르는 추가 비용을 초래한다. 하룹은 이러한 순환 구조를 효율적으로 처리하기 위해 워싱턴대학교에서 개발되었다. 하룹은 전 순환 단계에서 변경되지 않은 데이터를 맵리듀스 클러스터에 캐쉬(cache)하여 후 순환 단계에서 재사용한다. 또한 리듀스 단계의 출력 데이터를 맵리듀스 클러스터에 캐쉬하여, 별도의 맵리듀스 작업을 실행하지 않고서도 부동점 계산을 더욱 효율적으로 수행한다.

2.1 하룹의 구조

하룹은 공개 소스 맵리듀스인 하둡을 확장하여



(그림 3) 하룹의 구조

개발한 것으로, 하둡의 분산 컴퓨팅 모형과 아키텍처를 그대로 이어 받고 있다. (그림 3)은 하룹의 구조를 나타내고 있다.

각 입출력 데이터의 저장은 HDFS에 의존하고, 클러스터는 하나의 마스터 노드와 다수의 슬레이브 노드로 구성된다. 사용자가 제출한 작업에 대해 마스터 노드는 슬레이브 노드에서 실행할 다수의 병렬 타스크를 스케줄링한다. 각 슬레이브 노드에는 타스크 트래커가 있다. 타스크 트래커는 마스터 노드와 통신하면서 각 타스크의 실행을 관리한다. 순환 처리를 위해, 하룹은 새로운 API를 사용자에게 제공하여 순환 맵리듀스 프로그램의 표현을 단순화한다. 하룹의 마스터 노드는 순환 통제 모듈을 포함하여, 사용자가 명시한 종료 조건이 만족될 때까지 순환체(loop body)를 구성하는 새로운 맵리듀스 단계를 지속적으로 개시시킨다. 하룹은 순환 어플리케이션을 위한 새로운 타스크 스케줄러를 사용하여, 어플리케이션 간의 데이터 국지성을 높인다. 하룹은 슬레이브 노드에 어플리케이션 데이터를 캐싱 및 인덱싱하여 다음 순환 단계에서 재사용될 수 있도록 한다. 하룹의 타스크 트래커는 하둡과 달리, 타스크의 실행뿐만 아니라 슬레이브 노드에 캐쉬와 인덱스를 관리하며 각 타스크의 캐쉬와 인덱스 접근을 로컬 파일 시스템으로 방향을 돌려(redirect) 놓는다.

2.2 하룹의 프로그래밍 모형

하룹 프로그램은 순환체, 종료 조건, 순환 불변 데이터에 대한 명세로 구성된다. 이 중에서 종료 조건과 순환 불변 데이터는 선택 사항이다. 순환체는 하나 이상의 맵리듀스 쌍을 말한다.

여러 단계의 맵리듀스로 구성된 하룹의 순환체를 명세화하기 위한 API는 다음과 같다.

- Map: 입력 <key, value> 튜플을 임시의 <in-key, in-value> 튜플로 변환한다.
- Reduce: 동일한 in-key를 공유하는 임시 튜플을 처리하여 <out-key, out-value> 튜플을 생성한다. In-key와 연관된 불변 데이터의 캐싱을 위한 파라미터가 인터페이스에 추가된다.
- AddMap, AddReduce: 하나 이상의 맵리듀스 단계로 구성된 순환체를 포현한다. AddMap과 AddReduce는 Map, Reduce 함수에 단계의 순서를 나타내는 정수를 연관시킨다.

하룹은 종료를 판단하기 위해 전단계와 현단계의 계산 결과가 같은지 검사한다. 프로그래머는 다음의 메소드를 사용하여 부동점 종료 조건을 명시한다.

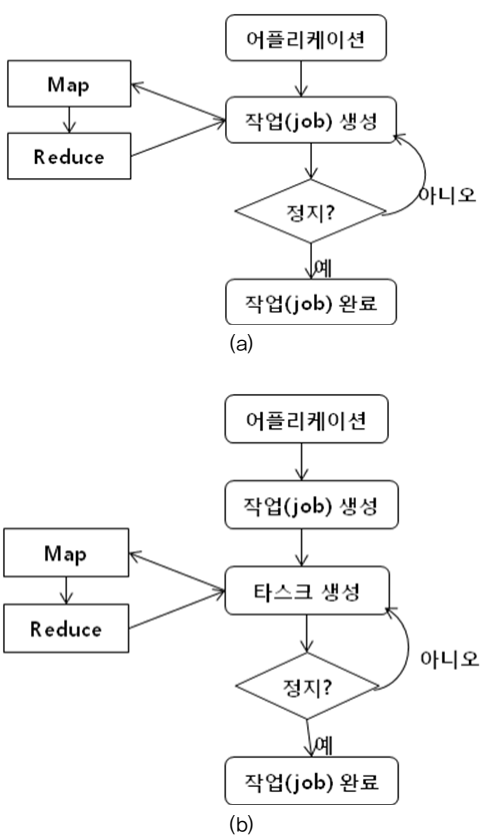
- SetFixedPointThreshold: 현재 순환과 다음 순환 간의 거리에 한계를 설정한다. 한계를 넘으면 부동점에 도달하지 않았다는 것이고, 따라서 계산은 지속적으로 반복 수행된다.
- ResultDistance: 이 함수는 동일한 out-key를 공유하는 두 out-value 집합 간의 거리를 계산한다. 여기서 두 out-value 집합은 동일한 out-key를 갖는 전단계의 출력과 현단계의 출력 데이터 집합을 말한다. 현단계와 전단계의 출력의 차이는 모든 out-key에 대한

ResultDistance를 합하여 계산된다.

- SetMaxNumOfIterations: 최대 순환 횟수를 명시한다. 이 횟수를 넘어 가면 결과의 거리에 상관없이 순환 처리를 종료한다. 이것은 for 순환을 구현하는 데 사용될 수 있다.

사용자는 입력 데이터를 명시하고 통제하기 위해 다음 API를 사용한다

- SetIterationInput: 입력 파일이 순환마다 다를 수 있기 때문에 이를 위해 입력 데이터 소스를 특정 순환에 연관시킨다.
- AddStepInput: 추가적인 입력 데이터 소스를 순환체의 임시 맵리듀스 쌍에 연관시킨다. 전단계 맵리듀스의 출력은 다음 맵리듀스의 입력 데이터가 된다.
- AddInvariantTable: 순환 불변 데이터가 되는 입력 테이블(HDFS 파일)을 명시한다. 작업을 실행하는 동안 하룹은 이 테이블을 클러스터 노드에 캐싱한다.



(그림 4) (a) 하둡을 사용하는 경우, (b) 하룹을 사용하는 경우

동일한 기능의 어플리케이션을 하룹의 API를 사용하여 실행하는 경우와 하둡을 사용하여 실행하는 경우를 비교하면, (그림 4)와 같이 나타낼 수 있다.

2.3 하룹의 순환 인식 타스크 스케줄링

하룹 스케줄러의 목표는 다른 순환 단계에서 발생하지만 동일한 데이터를 접근하는 맵과 리듀스 타스크를 동일한 물리적 머신에 할당하는 것이다. 이렇게 하면 순환 간에 더욱 쉽게 데이터를 캐싱하고 재사용할 수 있다. 이를 위해 하룹 스케줄러는 각 물리적 노드에서 맵 타스크와 리듀스 타스크가 처리한 데이터 파티션을 추적하고, 이 정보에 따라 순환 간의 국지성을 고려하여 다음 단계의 타스크를 스케줄링한다.

3. 요약 및 활용 현황

하둡은 병렬 처리를 위한 수단으로 널리 사용되고 있다. 하둡은 순환을 직접 지원하고 있지 않기 때문에, 순환을 요구하는 프로그램의 개발에 어려움을 내포하고 있다. 본고에서는 이러한 문제점을 해결하기 위한 방법으로 하룹과 트위스터를 소개하였다. 하룹은 아직 프로토타입 수준으로, 활성화되지는 않았지만 향후 결함 포용성이 제고된다면 일반적인 어플리케이션에서 널리 활용될

〈표 1〉 트위스터와 하둡의 비교

	트위스터	하둡
하둡과의 호환성	불가능 (맵리듀스 프레임워크 자체 개발)	가능 (하둡의 확장 개선)
순환 처리 방식	외부 순환 처리 드라이버 제공	실행 시스템 내부에 순환 처리 구현
순환에 따른 작업 생성	순환마다 별도의 작업 생성	순환마다 별도의 작업 대신에 타스크 생성
작업 및 타스크 간의 불변 데이터 공유 방법	안메모리 사용하여 저장 및 공유	캐쉬, 인덱싱 사용하여 공유
결함 내성 지원	지원하지 않음	지원하지 않음 (기술적으로 하둡을 계승하고 있으므로 향후 지원 가능)

것이다. 반면, 트위스터는 살사(SALSA)[3]라는 프로젝트를 통해 메모리를 많이 사용하는 데이터 집중적인 과학 어플리케이션의 병렬화에 사용되었고, 이들 어플리케이션은 대규모 데이터 분석을 원하는 고성능 컴퓨팅 계산 과학자에게 널리 사용될 것이다.

참 고 문 헌

[1] J. Ekanayake, et. al., “Twister: A Runtime for Iterative MapReduce,” HPDC 2010

[2] Y. Bu, B. Howe, M. Balazinska and M. Ernst, “HaLoop: Efficient Iterative Data Processing on Large Clusters,” VLDB 2010

[3] http://salsahpc.indiana.edu/