공학석사학위논문

맵리듀스 클러스터에서 처리량 향상을 위한 지역성 인지 기반 분산 처리 아키텍처

Locality-Aware Distributed Processing Architecture for Improving Throughput in MapReduce Clusters

2015년 8월

서울대학교 대학원

전기컴퓨터공학부

김 경 래

맵리듀스 클러스터에서 처리량 향상을 위한 지역성 인지 기반 분산 처리 아키텍처

Locality-Aware Distributed Processing Architecture for Improving Throughput in MapReduce Clusters

지도교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함

2015년 6월

서울대학교 대학원

전기컴퓨터공학부

김 경 래

김경래의 석사 학위논문을 인준함

2015년 6월

위 원 장 김 태환 (인)

부위원장 홍 성수 (인)

위 원 심 규석 (인)

국문초록

대규모의 데이터를 처리하기 위한 클러스터의 플랫폼으로써 맵리듀스 프레임워크와 분산 파일 시스템 기반의 플랫폼이 많이 쓰이고 있다. 특별한 스토리지, 네트워크 시스템 없이 상용 하드웨어로 구축되는 맵리듀스 클러스터에서 네트워크 자원은 가장 스케일업이 어렵고, 희소한 자원이다. 그 때문에 멀티 테넌트 환경에서는 잡과 잡 사이의 네트워크 자원 경쟁을 줄여 클러스터 전체의 잡 처리량을 높이는 것이 중요하다. 네트워크 자원 경쟁이 발생하는 주된 요인은 입력 데이터의 지역성으로 인한 크로스랙 네트워크 트래픽 때문이며, 입력 데이터의 지역성을 높이고자 다양한 기법이 제안되어왔다. 이들은 지역성을 높이기 위한 데이터 분산 정책의 부재로 인해 단편적인 지역성 향상에 그친다는 한계가 있다. 따라서 본 학위 논문에서는 데이터 지역성 인지 기반 분산 처리 아키텍처를 제시한다. 파일 단위로 데이터의 지역성을 인지할 수 있는 분산 파일 시스템과 지역성 인지 기반의 잡 스케줄링 정책을 통해 잡 전체의 데이터 지역성을 높임으로써 완화된 네트워크 자원 경쟁을 통해 잡 처리량을 높인다. 실험 결과 제안된 시스템은 기존 시스템에 비해 맵리듀스 클러스터의 잡 처리량을 17% 이상 향상시킬 수 있었다.

주요어 : 맵리듀스, 분산 처리, 지역성, 분산 파일 시스템
학 번 : 2013-23101

목 차

표 목 차

그 림 목 차

제 1 장 서론

상용 컴퓨터들의 네트워크로 이루어진 클러스터에서 대규모의 데이터를 분산 처리하기 위한 소프트웨어 플랫폼으로써 맵리듀스 프레임워크와 분산 파일 시스템 기반의 플랫폼이 많이 쓰이고 있다. 맵리듀스 프레임워크는 쉬운 프로그래밍 모델과 더불어 입력 데이터의 자동화된 병렬 처리 기능을 제공하며, 분산 파일 시스템은 높은 비용을 감수하지 않고도, 분산 저장된 데이터의 내결함성 있는 접근을 제공한다. 그로인해 현재는 두 서브 시스템으로 이루어진 분산 처리 플랫폼이 데이터 집약적인 응용을 수행하는 환경에서 필수적인 소프트웨어 플랫폼으로 자리 잡았다.

이러한 소프트웨어 플랫폼을 기반으로 동작하는 맵리듀스 클러스터는 클러스터 운용의 비용 효율성과 클러스터의 높은 활용도를 위해 멀티 테넌트 환경인 경우가 많다. 즉, 복수의 사용자로부터 분산 처리 응용들이 실행되며, 이들 분산 처리 응용은 잡(job)으로 매핑된다. 맵리듀스 클러스터에서는 생성된 복수의 잡을 동시에 처리한다.

하지만 멀티 테넌트 환경에서는 복수의 잡들이 클러스터의 컴퓨팅 자원을 공유하기 때문에, 잡과 잡사이에 자원 경쟁이 불가피하다. 일반적으로 상용 컴퓨팅 자원으로 구성되는 맵리듀스 클러스터는 전통적인 2티어 이상의 네트워크 구조로 구성되어 있으며, 이 중에서도 스위치를 비롯한 네트워크 자원은 스케일업(scale-up)이 어렵고 희소한 자원이다[1, 4]. 노드별 대역폭이 적은 aggregation 이상의 네트워크 스위치의 한정적인 대역폭 아래에서 이러한 네트워크 자원 경쟁은 클러스터 전체의 잡 처리량을 감소시키는 주된 원인이 되어왔다[3, 4].

맵리듀스의 잡 수행과정에서 발생하는 네트워크 트래픽은 컨트롤 메시지를 제외하면 입력 데이터의 지역성에서 발생한다. 태스크가 입력 데이터와 다른 네트워크 위치에서 수행될 경우, 입력 데이터를 태스크가 할당된 네트워크 위치로 전송하는 과정에서 네트워크 트래픽이 발생하게

된다. 특히 태스크와 입력 데이터가 서로 다른 랙에 위치할 경우가 많아질수록, 크로스 랙 네트워크 트래픽이 많아질수록 aggregation 스위치의 제한된 대역폭으로 인해 클러스터의 잡 처리량이 감소하게 된다.

입력 데이터의 지역성을 높여 네트워크 트래픽을 감소시킴으로써 잡 처리량을 향상시키기 위한 연구가 진행되어왔다. 이들 연구는 맵 태스크 입력의 데이터 지역성을 높이기 위한 연구와 리듀스 태스크 입력의 데이터 지역성을 높이기 위한 연구로 나뉘어져 진행되어 왔다. 전자의 경우, Zaharia[3]는 맵 태스크의 스케줄링 결정 시간을 지연하여 맵 태스크를 입력과 동일한 위치에서 수행될 확률을 높여 잡 처리량을 높이기 위한 스케줄링 정책을 제시하였고, Isard[6]는 분산된 파일을 읽을 때, 입력 데이터의 지역성을 높이기 위해 잡의 공정도에 따라 수행 중인 다른 잡의 태스크를 종료시켰다. 후자에 해당하는 연구로써, 대표적인 맵리듀스 프레임워크인 하둡 맵리듀스의 잡 스케줄러는 맵 태스크의 수행 중간에 리듀스 태스크를 할당하여, 먼저 수행을 마친 맵 태스크의 결과값을 리듀스 태스크들에게 전달함으로써 동시에 한꺼번에 전달하면서 야기되는 네트워크 과부하를 줄이고자 하였다. Ahmed[4]는 맵 태스크와 리듀스 태스크를 최소한의 랙에 할당하는 스케줄링 정책을 제시하여, 리듀스 태스크의 데이터 지역성을 높였다.

하지만 이러한 연구들의 한계는 맵 태스크와 리듀스 태스크의 입력 데이터의 지역성을 동시에 고려하지 못하고, 필요에 따라 다른 하나의 지역성을 희생하거나 보장하지 못한다는 점이다. 이는 맵 태스크의 입력 데이터는 파일이 분산된 위치에 의존하지만, 리듀스 태스크의 입력 데이터는 맵 태스크가 할당된 위치에 의존하기 때문에, 기존의 분산 파일 시스템과 맵리듀스 프레임워크의 잡 스케줄러만으로는 동시의 두 개의 데이터 지역성을 향상시키는 것에 한계가 있기 때문이다.

따라서 본 학위 논문에서는 데이터 지역성 인지 기반 분산 처리 아키텍처를 제시한다. 이 분산 처리 아키텍처는 맵 태스크와 리듀스 태스크의 데이터 지역성을 동시에 높임으로써 멀티 테넌트 환경에서 맵리듀스 클러스터의 잡 처리량을 높이는 것이 목적이다.

이를 위해 파일을 적은 수의 랙에 분포시키기 위한 쓰기 정책, 파일마다의 랙 선호도에 대한 메타데이터 기록이 목적인 파일 지역성 인지 분산 파일 시스템과 입력 파일의 랙 선호도를 인지하여, 맵과 리듀스 태스크들을 적은 수의 랙에 분포시킬 수 있는 지역성 인지 기반 잡 스케줄러를 구현하였으며, 실험결과 적용 전에 비해 잡 처리량이 17% 이상 향상되었음을 검증하였다.

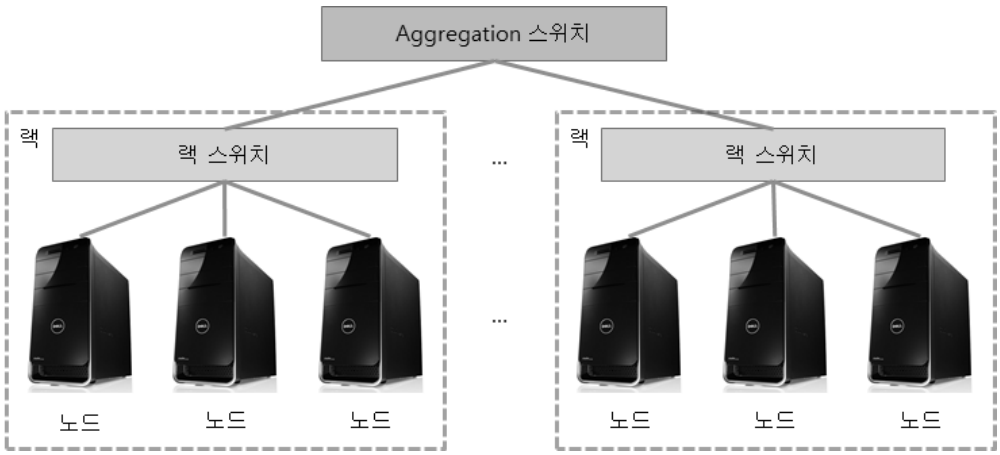
본 논문의 나머지는 다음과 같이 구성된다. 2장에서는 맵리듀스 클러스터의 네트워크 구조와 분산 파일 시스템의 간략한 소개와 데이터 관리 방법, 그리고 맵리듀스 프레임워크에서 잡이 수행되는 과정을 살펴본다. 3장에서는 맵리듀스 클러스터에서 네트워크 자원 경쟁을 발생시킬 수 있는 데이터 지역성 문제를 정의하고 이를 극복하기 위한 방법을 개괄한다. 4장에서는 제안된 해결책과 구현에 대해 상세히 설명한다. 마지막으로 5장에서는 본 연구의 검증과 의의, 그리고 향후 발전 방향에 대해 논의한다.

제 2 장 배경

제 1 절 맵리듀스 클러스터 아키텍처

1.1 네트워크 토폴로지

맵리듀스 클러스터는 노드와 랙으로 구성되는 [그림 1]과 같은 전통적인 2계층 이상의 네트워크 계층구조를 가지고 있다. 개별 물리머신인 노드들을 기본단위로 하여, 노드들이 네트워크 스위치로 연결되어 랙(rack)을 이루며, 랙들이 aggregation 스위치를 통해 연결된 네트워크 토폴로지를 이루고 있다.



[그림 1] 맵리듀스 클러스터의 네트워크 토폴로지

1.2 소프트웨어 아키텍처

일반적으로 맵리듀스 클러스터에서는 SAN(Storage Area Network)와 같은 특별한 스토리지 구성을 소프트웨어로 대체하기 위해서 분산 파일

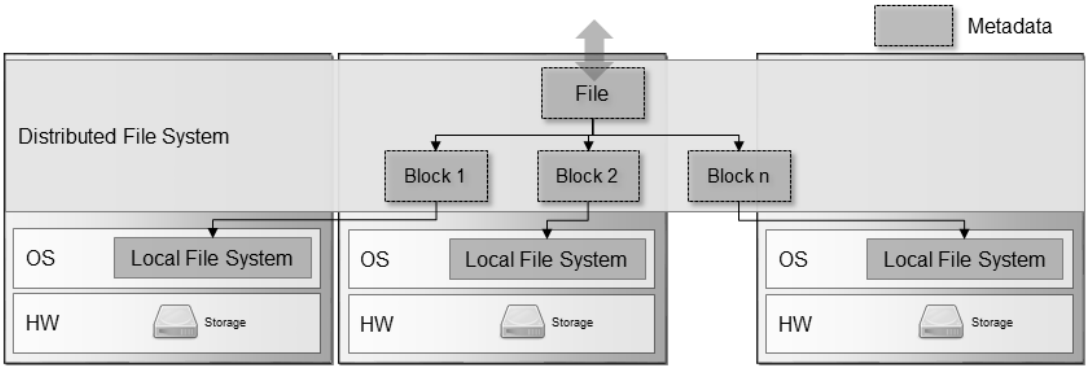
시스템을 통해 대규모 데이터의 입출력을 수행한다. 맵리듀스 프레임워크는 맵리듀스 프로그래밍 모델로 작성된 프로그램을 클러스터 내의 노드들에 태스크로 나누어 병렬처리한다. 이 때 맵리듀스 프레임워크가 처리하기 위한 입력 데이터 접근은 분산 파일 시스템을 통해서 이루어진다.

제 2 절 분산 파일 시스템

분산 파일 시스템은 각 노드의 로컬 파일 시스템들을 추상화하여 단일 노드의 스토리지 용량을 초과하는 대규모 데이터셋의 입출력을 여러 노드에 나누어 병렬 수행한다. 또한 동시에 데이터의 복제를 통해 내결함성 있는 데이터 접근을 보장하며, 또한 노드의 확장에 따른 파일 시스템의 확장성을 지원한다[2, 7].

2.1 소프트웨어 아키텍처

분산 파일 시스템은 하나의 마스터와 여러 개의 슬레이브로 구성된다. 마스터와 슬레이브는 각 노드별로 동작하는 프로세스이다. 마스터에서는 분산 파일 시스템 전체 파일 시스템의 메타데이터를 관리하고, 슬레이브는 로컬 파일 시스템을 통해 실제 파일 입출력을 수행한다. 마스터와 슬레이브 사이의 하트비트(Heartbeat)이라는 메시지를 통해서 마스터는 주기적으로 파일 시스템의 메타데이터를 갱신하고, 슬레이브는 마스터로부터 전달받은 파일 입출력 명령을 수행한다.



[그림 2] 분산 파일 시스템 아키텍처

2.2 파일 관리

분산 파일 시스템의 파일은 여러 개의 고정된 크기를 가진 블락들로 구성되어 있다. 분산 파일 시스템의 블락이란 로컬 파일 시스템에서 입출력을 수행하는 단위 기준인 파일에 해당한다. 분산 파일 시스템은 파일을 쓸 때, 파일을 여러 블락으로 나누고, 이 블락들을 여러 노드에 분산하여 저장한다. 반대로 분산 파일 시스템이 파일을 읽을 때에는 파일을 이루는 블락들을 순서대로 접근한다.

2.3 블락 쓰기 정책

파일 입출력의 내결함성을 보장하고, 파일의 높은 이용가능성을 위해 파일의 각 블락들은 임의의 노드에 분산 저장되며, 더불어 개별 블락들을 서로 다른 노드에 여러 번 복제한다. 시스템에 의해 정해진 복제 횟수에 따라 동일한 블락을 여러 노드에 저장하기때문에, 분산 파일 시스템 동작 중에 하드웨어 문제가 발생하여도 다른 노드로부터 복구하거나 재복사하여 전체 파일의 데이터를 유지한다.

제 3 절 맵리듀스 프레임워크

맵리듀스 프레임워크는 맵리듀스 프로그래밍 모델로 작성된 프로그램을 맵리듀스 클러스터 내에서 병렬 처리하기 위한 프레임워크이다. 맵리듀스 프로그래밍 모델로 작성된 프로그램은 프레임워크에 의해서 잡으로 매핑되며, 잡은 다시 프레임워크 내의 잡 스케줄러에 의해서 각 노드에 맵과 리듀스 태스크들로 할당되어진다.

3.1 맵리듀스 잡

맵리듀스 프레임워크에서 잡은 맵리듀스 프로그램이며, 프레임워크에 의해서 잡은 맵 태스크들과 리듀스 태스크들의 집합이며, 잡의 처리는 맵, 셔플, 리듀스의 3단계로 나누어져 처리된다. [그림 2]은 하나의 잡이 처리되는 과정을 나타낸다.

3.1.1 맵

맵 태스크는 대용량의 파일 혹은 입력데이터를 분산 저장된 단위, 즉 분산 파일 시스템의 하나의 블락을 읽어 사용자에 의해 정의된 전처리를 수행하는 태스크이다. 따라서 하나의 맵리듀스 잡의 맵 태스크의 수는 입력 파일을 구성하는 블락의 수와 같다. 입력 블락이 맵 태스크가 할당된 노드에 존재하지 않을 경우에는 입력 블락이 저장되어 있는 노드로부터 네트워크를 통해 블락을 복제한다. 맵 단계에서는 맵 태스크가 블락을 읽고 정해진 전처리 과정을 수행하여 중간 결과값을 생성한다.

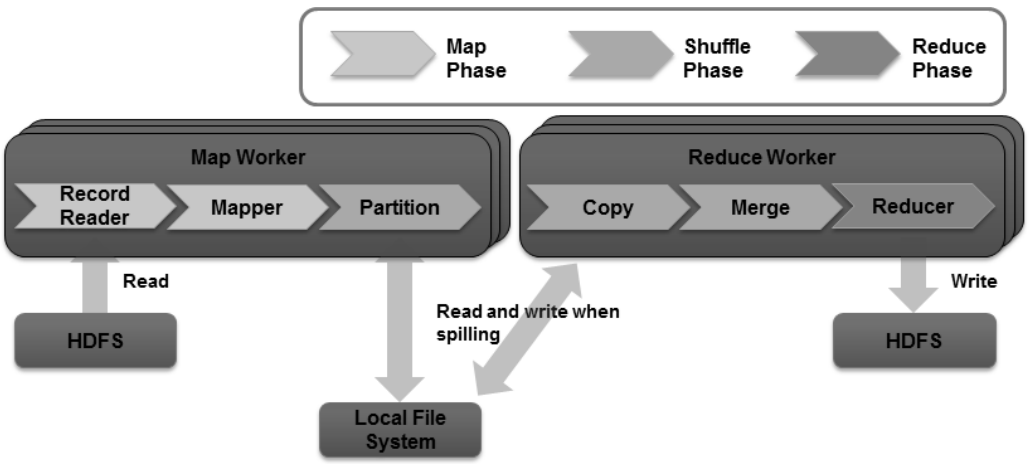
3.1.2 셔플

셔플은 맵 태스크의 결과값을 리듀스 태스크의 입력값으로 전달하는

단계이다. 리듀스 태스크는 모든 맵 태스크로부터 맵 태스크의 결과값을 선택적으로 전송받는다. 따라서 셔플이 발생할 경우 모든 맵과 모든 리듀스 태스크 사이의 통신이 발생하게 된다. 통계적으로 Yahoo, Facebook와 같은 대규모의 데이터센터의 잡의 32%는 셔플이 없는 잡이며, 나머지 68%의 잡은 셔플이 발생하는 잡이다[5].

3.1.3 리듀스

리듀스 태스크는 네트워크를 통해 모든 맵 태스크의 결과를 선택적으로 전송 받는다. 하나의 잡에 의해 생성되는 리듀스 태스크의 수는 시스템에 설정된 고정값이기 때문에, 이 값을 키로하는 해쉬테이블이나 사용자가 설정한 방법에 의해 맵 태스크의 결과값의 일부를 전송 받는다. 리듀스 단계에서는 맵 태스크의 결과값을 병합하고, 사용자에 의해 정의된 후처리과정을 수행하여 최종 결과값을 출력한다.

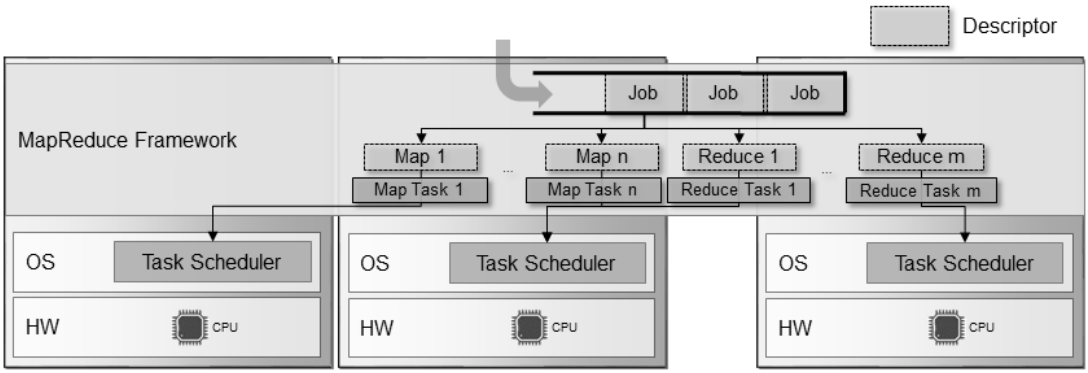


[그림 3] 맵리듀스 잡의 처리 과정

3.2 소프트웨어 아키텍처

분산 파일 시스템과 마찬가지로 하나의 마스터 프로세스와 여러 개의

슬레이브 프로세스들로 구성된다. 마스터에서는 사용자들에 의해 입력된 잡들의 진행정도를 모니터링하고 잡의 스케줄링을 수행한다. 슬레이브는 각 노드에서 마스터의 잡 스케줄링의 결과로 할당된 태스크를 실행한다. 마찬가지로 마스터와 슬레이브 사이의 하트비트(Heartbeat)이라는 메시지를 통해서 마스터는 주기적으로 잡의 진행정도를 모니터링하고 각 노드에서 수행할 태스크를 결정한다. 슬레이브는 마스터로부터 할당받은 태스크를 실행시킨다.



[그림 4] 맵리듀스 프레임워크의 소프트웨어 아키텍처

3.3 공정 스케줄러

맵리듀스 클러스터는 운용의 비용 효율성과 클러스터의 높은 활용도를 위해 복수의 잡이 동시에 실행된다. 잡이 실행되고 있다는 것은 잡을 구성하는 맵과 리듀스 태스크들이 노드에 할당되어 수행되고 있다는 것을 의미한다. 멀티테넌트 환경에서는 복수의 사용자가 잡을 생성하기 때문에, 사용자 간의 잡 수행의 공정성을 보장하기 위해 공정 스케줄링 기법이 많이 적용된다[3, 4, 6]. 잡 스케줄링은 2단계로 나뉘어져서 진행된다. (1) 공정도에 따라 태스크를 실행시킬 잡을 선택하고, (2) 맵리듀스 슬레이브를 통해 맵 혹은 리듀스 태스크를 할당한다.

3.1 잡 선택

태스크를 실행시킬 잡을 선택하는 기준인 공정도는 잡이 수행 중인 태스크 수의 비로 계산하여 이 중 가장 비율이 낮은 잡을 선택한다. 작업 보존적인 스케줄러이기때문에, 새로운 잡이 추가되면 공정도를 맞추기 위해 새로 추가된 잡을 선택하여 태스크를 스케줄링하기 위해 기존의 잡의 태스크의 수행을 중지시키거나 수행 중인 태스크의 수행 종료를 기다린다.

3.2 태스크 스케줄링

태스크 스케줄링은 잡에서 수행되지 않거나, 수행이 지체되어 재시작해야하는 맵과 리듀스 태스크 중에 실행할 태스크를 정한다. 셔플과 리듀스 단계는 맵 단계가 종료되어야 진행되기 때문에, 태스크 스케줄링 시 잡의 모든 맵 태스크를 스케줄링에 앞서 리듀스 태스크의 스케줄링이 일어나지는 않는다. 맵리듀스 클러스터의 각 노드는 각각 고정된 수치의 최대 실행가능한 맵 태스크 수와 리듀스 태스크 수를 갖는다. 이 값들을 슬랏(slot)이라고하며, 태스크 스케줄링은 슬랏이 차지 않는 노드에서 실행할 태스크를 정한다.

맵 태스크를 스케줄링할 때는 재시작해야하는 태스크를 최우선으로 스케줄링하며, 실행가능한 노드에서 입력데이터를 읽는 거리에 따라 로컬, 랙 로컬 순으로 태스크를 스케줄링하고, 그마저도 없다면 랜덤한 맵 태스크를 스케줄링한다. 리듀스 태스크의 경우는 가능한 노드 중에서 임의의 위치에 스케줄링한다.

제 3 장 관련 연구

제 1 절 데이터 지역성

맵리듀스의 잡 수행과정에서 컨트롤 메시지를 제외하면 발생하는 네트워크 트래픽은 입력 데이터의 지역성에서 발생한다. 지역성이란 태스크와 태스크의 입력데이터 사이의 네트워크 상의 거리를 의미하며, 태스크가 입력 데이터와 다른 네트워크 위치에서 수행될 경우 입력 데이터를 태스크가 할당된 네트워크 위치로 전송하는 과정에서 네트워크 트래픽이 발생하게 된다. 특히 태스크와 입력 데이터가 서로 다른 랙에 위치할 경우가 많아질수록, aggregation 스위치의 제한된 대역폭으로 인해 클러스터의 잡 처리량이 감소하게 된다.

입력 데이터의 지역성을 높여 네트워크 트래픽을 감소시킴으로써 잡 처리량을 향상시키기 위한 연구가 진행되어왔다. 이들 연구는 맵 태스크의 데이터 지역성을 높이기 위한 연구와 리듀스 태스크의 데이터 지역성을 높이기 위한 연구로 나뉘어져 진행되어 왔다.

제 2 절 맵 태스크의 데이터 지역성 관련 연구

맵 태스크의 데이터 지역성은 맵 태스크와 맵 태스크의 입력 블락 위치에 따라 지역, 랙 지역과 크로스 랙의 3가지 레벨로 나뉜다. 맵리듀스 프레임워크에서 기본적으로 지역 데이터를 처리할 수 있는 맵 태스크를 우선으로 스케줄링하지만, 하나의 태스크가 지역 데이터를 처리할 수 있는 노드의 경우의 수는 최대 복제 횟수가 같으며, 멀티 테넌트 환경에서는 잡 사이의 슬랏 경쟁으로 인해 가능성은 더욱 낮아 랙 로컬, 심하게는 크로스 랙의 데이터를 활용하게 되기도한다. 이러한 문제를 해결하

기 위해, Zaharia[3]는 맵 태스크의 스케줄링 결정 시간을 지연하여 대기 중인 태스크의 지역 데이터가 위치한 노드에 스케줄링이 가능해지길 기다리는 스케줄링 정책을 제안하였다. 맵 태스크가 지역 데이터 혹은 랙 지역 데이터를 처리할 확률을 높여서 스케줄링 결정 지연으로 인한 잡 처리량을 보완하여 역으로 향상시킬 수 있음을 보였다. Isard[6]는 맵리듀스 프레임워크가 아닌 다른 프레임워크에서 맵 단계와 동일하게 분산된 파일을 읽을 때, 입력 데이터의 지역성을 높이기 위해 잡의 공정도에 따라 대기 중인 태스크의 지역 데이터가 위치한 노드에서 수행 중인 다른 잡의 태스크를 종료시키는 정책을 제안하였다.

제 3 절 리듀스 태스크의 데이터 지역성 관련 연구

리듀스 태스크의 데이터 지역성은 리듀스 태스크와 복수의 맵 태스크 결과값의 네트워치 거리의 합에 의해서 결정된다. 맵 태스크의 결과값은 맵 태스크가 수행된 노드에 임시 저장되기 때문에, 리듀스 태스크와 모든 맵 태스크의 네트워크 거리에 따라 전부 지역 데이터, 전부 랙 지역 데이터와 그 외의 경우로 나눌 수 있다. 일반적으로 하나의 잡에서 실행되는 맵 태스크는 단일 노드의 맵 슬랏보다 많기 때문에, 리듀스 태스크의 입력이 전부 지역 데이터인 경우는 드물다. 리듀스 태스크와 다른 랙에 위치한 맵 태스크가 통신이 많아질 경우, aggregation 스위치의 제한된 대역폭으로 인해 클러스터의 잡 처리량이 감소하게 된다. 셔플 단계에서는 모든 맵 태스크가 동시에 리듀스 태스크로 데이터를 전송하기 때문에 다른 랙에 위치할 경우 네트워크 과부하가 더욱 심해진다.

가장 대중적으로 활용되는 맵리듀스 프레임워크인 하둡 맵리듀스의 잡 스케줄러에는 맵 단계와 셔플 단계를 오버랩(overlap)하여, 먼저 수행을 마친 맵 태스크의 결과값을 리듀스 태스크들에게 먼저 전달할 수 있는 인터페이스가 추가되었다. 셔플 단계에서 데이터를 동시에 한꺼번에 전달하면서 생기는 네트워크 과부하를 줄여 잡 처리량과 수행시간을 단

축시켰다. Ahmed[4]는 리듀스 태스크의 데이터 지역성을 높이기 위해서 일부 맵 태스크의 데이터 지역성을 포기하는 대신, 맵 태스크와 리듀스 태스크를 최소한의 랙에 할당하는 스케줄링 정책을 제시하여 셔플 비중이 높은 잡을 수행하는 클러스터의 처리량을 향상시켰다.

제 4 장 문제 정의

제 1 절 기존 연구의 한계점

기존의 연구들은 크로스 랙 레벨의 데이터 지역성을 줄임으로써 크로스 랙 네트워크 트래픽을 줄이고, 잡과 잡 사이의 네트워크 자원 경쟁을 완화하여 맵리듀스의 클러스터의 잡 처리량을 높이려고 하였다. 하지만 이들 연구들의 한계는 맵 태스크와 리듀스 태스크의 입력 데이터의 지역성을 동시에 고려하지 못하고, 다른 한 쪽의 데이터 지역성을 보장하지 못하거나 필요에 따라 희생한다는 점이다. 이는 맵 태스크의 입력 데이터는 파일이 분산된 위치에 의존하지만, 리듀스 태스크의 입력 데이터는 맵 태스크가 할당된 위치에 의존하기 때문에, 기존의 분산 파일 시스템과 맵리듀스 프레임워크의 잡 스케줄러만으로는 동시의 두 개의 데이터 지역성을 향상시키는 것에 한계가 있다.

제 2 절 문제 정의와 해결책 개관

맵 태스크의 데이터 지역성은 맵 태스크와 입력 블락의 네트워크 위치에 의존한다. 리듀스 태스크의 데이터 지역성을 높이기 위해서는 맵 태스크들이 분포한 랙의 수를 줄여야한다. 따라서 하나의 잡을 이루는 맵 태스크들와 리듀스 태스크들의 데이터 지역성을 동시에 높이기 위해서는 하나의 파일을 구성하는 블락들이 분산된 랙의 수를 줄여야한다.

하지만 기존의 맵리듀스 기반의 분산 처리 플랫폼에서는 맵과 리듀스 태스크들의 데이터 지역성을 동시에 향상시킬 수 없다. 하지만 기존의 맵리듀스 플랫폼에서는 블락들을 임의의 네트워크 위치에 저장한다. 블

락들이 임의의 위치에 저장되기 때문에, 맵 태스크의 지역성을 높이고자 할 경우에는 맵 태스크의 위치는 랜덤 분포가 되고, 적은 수의 랙에 맵 태스크를 모을 수 없거나 모으는 것이 제한된다. 반대로 리듀스 태스크의 지역성을 높이고자 맵 태스크들을 적은 수의 랙에 스케줄링하면 맵 태스크들이 입력 블락과 다른 위치에 실행될 확률이 높아져서 크로스랙 네트워크 트래픽의 감소를 보장할 수 없다.

따라서 본 논문에서는 멀티테넌트 환경에서 동일한 잡의 맵 태스크의 지역성과 리듀스 태스크의 지역성을 동시에 높일 수 있는 분산 처리 아키텍처를 제시한다. 구체적으로 본 논문의 아키텍처는 로컬 맵 태스크들을 적은 수의 랙에 스케줄링한다. 이를 위해서 파일 단위로 네트워크 위치의 상관관계를 형성하여 적은 수의 랙에 파일의 블락들을 위치시키기 위한 블락 배치 정책과 대부분의 블락들이 위치한 랙 정보를 바탕으로 태스크를 스케줄링하는 잡 스케줄러를 제시한다.

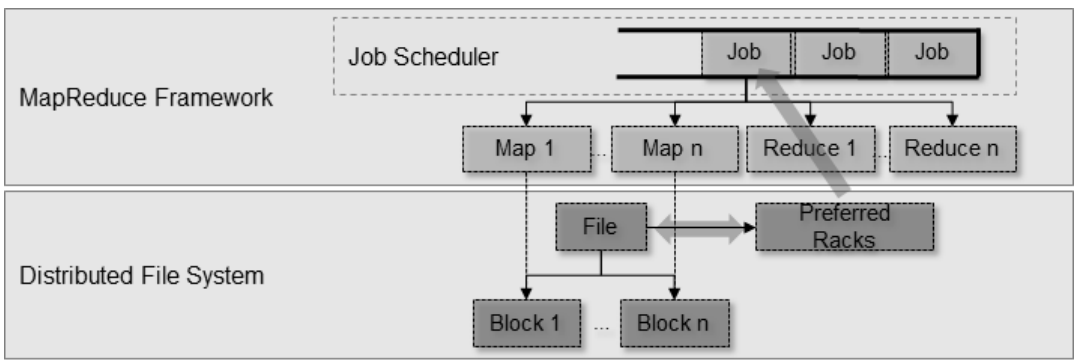
제 5 장 지역성 인지 분산 처리 아키텍처

본 장에서는 앞 장에서 다룬 문제를 해결하여 맵리듀스 클러스터의 잡 처리량을 향상시키기 위한 지역성 인지 기반 분산 처리 아키텍처에 대해 구체적으로 설명한다.

제 1 절 고려 사항들

본 절에서는 제안된 기법을 구현하기 위해서 고려해야할 요소들을 다룬다. 지역성 인지 분산 처리 아키텍처의 목적은 파일 단위로 블락들을 적은 수의 랙에 모을 수 있고, 입력 파일의 랙 선호도를 인지하여 잡의 맵과 리듀스 태스크들을 해당 랙에 우선적으로 스케줄링하는 것이다. 따라서 분산 파일 시스템의 쓰기 정책은 파일을 적은 수의 랙에 블락들을 분산시킬 수 있어야하며, 파일 시스템은 파일마다 블락들이 가장 많이 저장된 랙을 랙 선호도라는 메타데이터로 관리해야한다. 또한 잡 스케줄러는 잡의 태스크를 스케줄링할 때, 입력 파일의 랙 선호도를 분산 파일 시스템으로부터 읽어 해당하는 랙에 로컬로 수행가능한 맵 태스크를 우선하여 배치하고, 마찬가지로 리듀스 태스크를 배치할 수 있어야한다.

이를 위해 새로이 설계된 분산 처리 아키텍처는 [그림 5]과 같다.



[그림 5] 지역성 인지 분산 처리 아키텍처

하지만 파일을 기존보다 적은 수의 랙에 배치하는 것은 기존의 임의의 위치에 블락들을 배치하는 것에 비해 파일이 분산된 노드의 수가 적어진다는 점을 내포한다. 이 때 발생할 수 있는 트레이드오프(trade-off)를 최소화하여 네트워크 자원 경쟁을 완화하기 위해서 추가적으로 고려해야할 내용은 다음과 같다. (1) 파일을 구성하는 블록들을 적은 수의 랙에 위치시키면서도, 랙 손실에 대해 분산 파일 시스템의 내결함성을 보장해야한다. 또한 (2) 파일을 구성하는 블록들이 분산된 노드의 수가 적음에도 데이터 지역성을 기준으로 최적의 위치에 태스크를 할당할 수 있도록 잡과 잡 사이의 스케줄링 경쟁을 줄여야한다.

제 2 절 파일 지역성 인지 분산 파일 시스템

분산 데이터 관리 시스템은 파일을 적은 수의 랙에 분포시키기 위한 쓰기 정책, 파일마다의 랙 선호도에 대한 메타데이터 기록이 목적이다. 또한 클러스터 내의 랙에 하드웨어 실패가 발생하더라도 기존과 동일하게 분산 파일 시스템의 내결함성을 유지하는 쓰기 정책을 제공할 수 있어야한다. 또한 잡 사이의 스케줄링 경쟁으로 인해 선호되는 랙에 태스크를 할당할 수 없는 일이 발생하지 않도록 적절한 수의 노드에 블락들을 분산시켜야한다.

따라서 수정된 블락 쓰기 정책은 파일을 적은 수의 랙에 분포시키기 위해 첫 번째 블락을 임의의 위치에 기록을 하고, 그 위치를 기준으로 이후의 블락들을 쓸 노드를 정한다. 하나의 노드는 동시에 정해진 수만큼의 맵 태스크를 실행할 수 있기 때문에, 각 노드는 최대 맵 태스크 수 이상의 블락을 동시에 읽을 수 없다. 즉, 노드마다 정해진 최대 맵 태스크 수 이상의 블락들을 하나의 노드에 작성하는 것은 잡 사이의 스케줄링 경쟁이 심할 때, 한 번에 읽을 수 있는 입력 파일의 블락 수를 감소하는 결과를 낳는다. 따라서 랙 단위로 최대 맵 태스크 수를 기준으로 이를 초과할 경우 분산 저장할 랙을 추가하게 된다. 구체적인 쓰기 정책의 의사 결정 과정은 [그림 6]과 같다.

Algorithm: BLOCKALLOCATION

BLOCKALLOCATION$(block)$

1: $G \leftarrow \emptyset$, count $\leftarrow 0$
2: **if** $(is\_first\_block(block))$ **then**
3: $Choose\_local\_node(block)$
4: **for** $i \leftarrow 1$ **upto** $replication\_factor - 1$
5: $r \leftarrow Choose\_remote\_rack(block)$
6: $G \leftarrow G \cup r$
7: $Choose\_a\_random\_node\_in(G, block)$
8: $count \leftarrow count + 1$
9: **end for**
10: **else**
11: **for** $i \leftarrow 1$ **upto** $replication\_factor - 1$
12: **if** $(count > maximum\ map\ slots\ in\ G)$ **then**
13: $r' \leftarrow Choose\_another\_rack(G)$
14: $G \leftarrow G \cup r'$
15: **end if**
16: $Choose\_a\_random\_node\_in(G, block)$
17: $count \leftarrow count + 1$
18: **end for**
19: **end if**
20: $r_f \leftarrow most\_frequently\_accessed\_rack(G)$
21: **return** r_f

[그림 6] 블락 쓰기 정책 알고리즘

각 블락의 마지막 복제본은 기존에 랙을 제외한 다른 노드에 저장한다. 이는 하나의 복제본을 클러스터 내의 노드들에 최대한 고르게 분포시켜서 앞에서 다룬 파일 시스템의 내결함성을 유지하기 위함이다. 이는 추가적으로 잡과 잡 사이의 경쟁을 줄이기 위해서인데, 이는 다음 절에서 설명할 것이다.

모든 블락들이 분산 저장되면 가장 많은 블락을 작성한 랙을 파일의 랙 선호도로써 마스터의 메타데이터로 관리한다. 잡 스케줄러가 태스크를 할당할 때, 파일의 랙 선호도를 참조하여 이를 기준으로 맵과 리듀스

태스크를 스케줄링하게 된다.

제 3 절 지역성 인지 기반 잡 스케줄러

지역성 인지 태스크 스케줄링 정책은 잡의 입력 파일의 랙 선호도를 인지하여, 맵과 리듀스 태스크들을 적은 수의 랙에 분포시켜서 클러스터의 잡 처리량을 높이는 것이 목표이다. 잡과 잡 사이의 스케줄링 경쟁을 줄여 최적의 위치에 태스크를 할당할 수 있어야한다. 이를 위한 잡의 스케줄링 과정은 [그림 7]과 같다.

Algorithm: TASKASSIGNMENT

```
TASKASSIGNMENT(node)
 1:  T ← ∅
 2:  j ← select_a_job_with_fairness()
 3:  if (is_preferred_rack(node, j)) then
 4:      if (is_shuffle_job(j)) then
 5:          T ← T ∪ assign_local_maps(j)
 6:          T ← T ∪ assign_too_delayed_maps(j)
 7:          T ← T ∪ assign_reduces(j)
 8:      else
 9:          delay_assign_maps(j)
10:  else
11:      if (is_no_shuffle_job(j)) then
12:          T ← T ∪ assign_local_maps(j)
13:      end if
14:  end if
15:  return T
```

[그림 7] 잡 스케줄러의 태스크 스케줄링 정책

맵과 리듀스 태스크를 스케줄링하기에 앞서 선호되는 랙의 노드인지 확인하는 작업을 제외한 태스크 스케줄링 알고리즘은 기존의 공정 스케줄러에서 제공하는 알고리즘을 활용한다. 또한 선호하지 않는 랙이기 때

문에 현재의 태스크 스케줄링을 잠시 보류하는 알고리즘은 Zaharia[3]에 의해 제시된 알고리즘을 따른다.

잡과 잡 사이의 스케줄링 경쟁은 각 노드에서 최대 실행가능한 태스크의 수가 정해져 있기 때문에 발생한다. 셔플을 발생시키지 않는 잡의 경우, 맵 태스크의 입력 데이터의 지역성만 고려하면 된다. 이런 잡의 태스크들을 파일의 랙 선호도에 해당되지 않는 다른 랙의 저장된 블락들을 우선하여 맵 태스크를 할당한다. 선호되는 랙의 스케줄링 경쟁을 낮게하여, 동일한 블락들을 사용하거나 동일한 위치를 선호하는 셔플을 발생시키는 잡들을 수행할 때 태스크들의 데이터 지역성을 높일 수 있다.

제 6 장 실험 및 검증

이번 장에서는 제안된 지역성 인지 분산 처리 아키텍처를 구현하고, 그 유효성을 평가한다. 이를 위해서 5장에서 소개한 아키텍처를 하둡 분산 파일 시스템과 하둡 맵리듀스 프레임워크에 적용하고 대상 시스템에 대상 시나리오를 수행하여 성능 향상 정도를 측정한다. 마지막으로 결과에 대해 토의하고 앞으로의 개선 방향에 대해 논의한다.

제 1 절 실험 환경

지역성 인지 분산 처리 아키텍처를 구현하고 해당 아키텍처가 적용된 맵리듀스 클러스터의 잡 처리량을 평가하기 위해 아래 [표 1]와 같은 하드웨어, 소프트웨어 실험 환경을 구성하였다.

[표 1] 실험 환경

CPU	Intel Core2 Duo E7500 @2.93GHz
메모리 크기	6GB
NIC	1Gbit Ethernet
스위치	1Gbit Ethernet
랙당 노드 수 / 전체 랙 수	3 / 5
하둡 버전	1.2.1
블락 복제 횟수	3
노드당 최대 맵 태스크 수	3
노드당 최대 리듀스 태스크 수	2

맵리듀스 클러스터의 2계층 네트워크 토폴로지를 구현하기 위해서 리눅스의 tc 도구를 사용하여 랙과 랙 사이의 대역폭을 조절하였다. aggregation 스위치의 대역폭을 노드당 50Mbps로하여 랙과 랙 사이의 대역폭을 150Mbps로 조절하였다[1, 8].

또한 각 노드의 최대 맵과 리듀스 태스크의 수는 태스크의 메모리 사용량의 합이 노드의 전체 메모리 크기를 넘지 않도록 하여, 페이지 폴트로 인한 성능저하 요인을 배제하였다[9, 10].

제 2 절 실험 응용

Zaharia[3]의 스케줄링 기법은 하둡의 공정 스케줄러에 적용되어 오픈소스로 제공된다. 따라서 해당 기법과 본 논문의 기법이 적용된 하둡 맵리듀스 클러스터의 잡 처리량과 평균 잡 수행시간을 측정하기 위해 아래 [표 2], [표 3]와 같은 응용을 설정하였다.

멀티 테넌트 환경을 구성하기 위해 복수의 잡들이 동시에 수행되어 클러스터 자원을 공유하도록 하였다. 데이터센터의 약 70%에 해당하는 잡들이 24개 이하의 맵 태스크를 생성하기 때문에[3], 대다수에 해당하는 이들 잡을 응용으로 선정하였다. 단, 실제 데이터센터에서 활용되는 노드 수를 실험으로 재현하긴 어렵기 때문에 동시에 실행되는 응용의 수를 선형적으로 조절하였다.

실제 데이터센터의 잡의 33%는 셔플이 없는 잡으로 보고되기 때문에[5], 실험은 동일한 입력 파일을 공유하는 셔플이 있는 잡과 없는 잡이 동일한 입력 파일을 공유하는 환경에서 진행되었다. 다양한 워크로드가 수행되는 데이터센터의 수행환경을 재현하고자 추가적으로 셔플이 존재하는 잡들만 수행되는 환경에서 성능을 검증하였다.

[표 2] 셔플 없는 잡을 포함한(50%) 실험 응용 1

동일한 입력 파일을 처리하는 응용들(잡)의 종류	Terasort (셔플 존재), Teravalidate (셔플 없음)
동시에 실행된 응용의 수	6
입력 파일의 종류	3 (Teragen을 통해 생성)
각 입력 파일의 크기	1.5GB
각 입력 파일의 블락 수	24
블락 크기	64MB

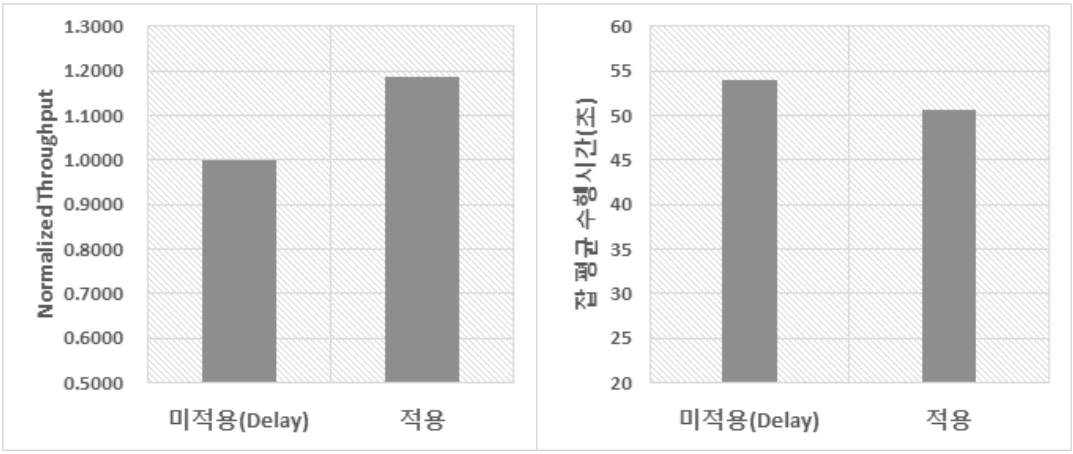
[표 3] 셔플이 존재하는 잡으로만 구성된 실험 응용 2

입력 파일을 처리하는 응용들(잡)의 종류	Terasort (셔플 존재)
동시에 실행된 응용의 수	6
입력 파일의 종류	6 (Teragen을 통해 생성)
각 입력 파일의 크기	1.5GB
각 입력 파일의 블락 수	24
블락 크기	64MB

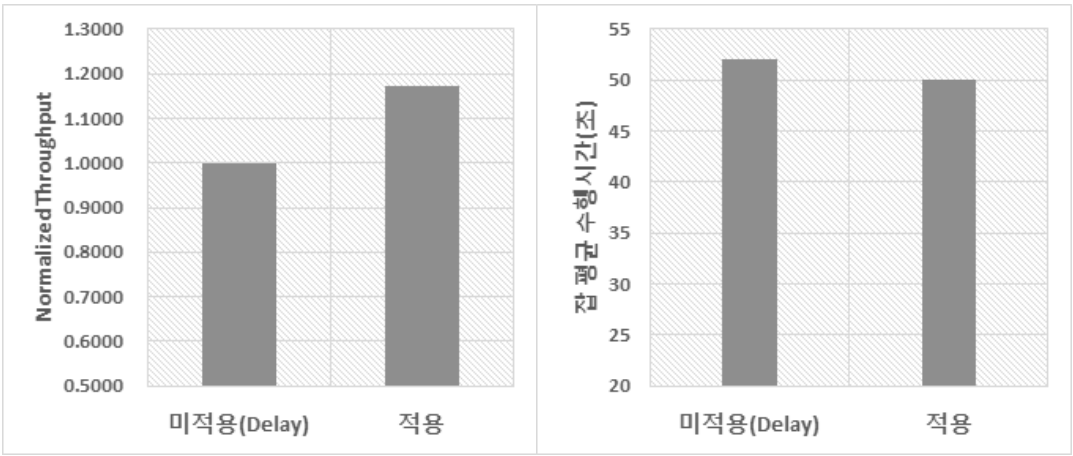
제 3 절 실험 결과

Zaharia[3]의 하둡의 공정 스케줄러와 그림 8, 그림 9은 대상 응용을 수행할 때 대상 시스템에 적용하기 이전과 이후에 얻은 실험 결과를 나타낸다. 제안된 기법이 적용된 맵리듀스 클러스터의 잡 처리량은 셔플이 없는 잡과 있는 잡이 클러스터 자원을 공유하는 실험 응용 1의 경우에는 적용 전에 비하여 18% 향상되었으며, 평균 잡 수행시간은 6% 향상되었다. 또한 모든 잡이 셔플을 발생시키더라도 잡 처리량에서는 17%, 평균

잡 수행시간의 경우는 4%가 적용된 기법을 통해 개선되었다.



[그림 8] 실험 응용 1의 실험결과



[그림 9] 실험 응용 2의 실험결과

제 7 장 결론

본 논문은 맵리듀스 클러스터에서 네트워크 자원 경쟁을 완화하여 클러스터의 잡 처리량을 향상시키기 위해 지역성 인지 분산 처리 아키텍처를 제시하였다. 이를 위해 맵리듀스 프레임워크와 분산 파일 시스템의 동작과 기본 정책을 분석하였다. 태스크와 입력데이터를 네트워크 거리를 통해 데이터의 지역성을 정의하였다. 이를 토대로 문제가 정의되고, 해결책이 제안되었다. 해결책은 잡 스케줄러의 테스크 스케줄링과 분산 파일 시스템의 쓰기 정책을 수정하여 기존의 연구에서 보이지 못했던 맵 태스크들과 리듀스 태스크의 데이터 지역성을 동시에 높여 클러스터의 잡 처리량이 향상되었다. 제안된 해결책은 실험결과 17% 이상 향상된 잡 처리량을 보였으며, 평균 잡 수행시간은 기존 대비 3% 이상 줄어듦을 확인할 수 있었다.

참 고 문 헌

[1] Jeffrey Dean, and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters.” USENIX Symposium on Operating Systems Design and Implementation, 2004

[2] S. Ghemawat, H. Gobioff, and S. Leung. "The Google File System." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.

[3] M. Zaharia, et al. “Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling.” European Conference on Computer Systems. ACM, 2010.

[4] F. Ahmad, et al. “ShuffleWatcher: Shuffle-Aware Scheduling in Multi-tenant MapReduce Clusters.” USENIX Annual Technical Conference, USENIX Association, 2014.

[5] M. Chowdhury, et al. “Managing Data Transfers in Computer Clusters with Orchestra.” ACM SIGCOMM Computer Communication Review 41.4: 98-109, 2011.

[6] M. Isard, et al. “Quincy: Fair Scheduling for Distributed Computing Clusters." ACM SIGOPS Symposium on Operating Systems Principles. ACM, 2009.

[7] Dharuba Borthakur. "HDFS Architecture Guide." Hadoop Apache Project, 2008.

[8] P. Costa et al. “Camdoop: Exploiting In-Network Aggregation for Big Data Applications.” NSDI, 2012.

[9] T. White. “Hadoop: the Definitive Guide 3rd Edition.” O’Reilly, 2011.

[10] S. Chakravarthi. “Tuning Hadoop for Performance.” Hadoop Summit, 2010.

Abstract

Locality-Aware Distributed Processing Architecture for Improving Throughput in MapReduce Clusters

Kyungrae Kim

Electrical and Computer Engineering

The Graduate School

Seoul National University

MapReduce clusters are now de facto standard cluster computing platform for processing large-scale data. Network resources are most scarce resources in MapReduce clusters, which are deployed only with commodity hardware except special storage and network systems. Therefore, especially in multi-tenant systems, it is important to achieve higher job throughput by relieving the contention on network resources between jobs. The contentions on network resources mainly come from the locality of input data, and many previous researches suggests solutions to improve data locality.

However, there researches ended up with having incomplete achievements on data locality due to the inexistence of data distributing policy. To achieve higher data locality, I suggest locality-aware distributed processing architecture in this dissertation. In the suggested architecture, the distributed file system is aware of file-level data locality, and the job scheduler schedules map and reduce tasks based on the file-level data locality. With two subsystems, the suggested architecture can relieve the network contentions and improve the job throughput. As a results of the experiments, the suggested architecture proved that it can have 18% higher job throughput.

keywords : MapReduce, Distributed Computing, Data Locality, Distributed Systems
Student Number : 2013-23101