공학석사 학위논문

SSD 기반 하둡 맵리듀스 시스템에서 셔플 단계 최적화를 통한 작업 처리속도 향상 메커니즘

Improving Job Processing Speed through Shuffle Phase Optimization for SSD-based Hadoop MapReduce System

2015 년 8 월

서울대학교 융합과학기술대학원

융합과학부 지능형융합시스템전공

고 광 옥

SSD 기반 하둡 맵리듀스 시스템에서 셔플 단계 최적화를 통한 작업 처리속도 향상 메커니즘

Improving Job Processing Speed
through Shuffle Phase Optimization
for SSD-based Hadoop MapReduce System

지도 교수 홍 성 수

이 논문을 공학석사 학위논문으로 제출함
2015 년 6 월

서울대학교 융합과학기술대학원
융합과학부 지능형융합시스템전공
고 광 옥

고광옥의 공학석사 학위논문을 인준함
2015 년 6 월

위 원 장 박 재 흥 (인)
부위원장 홍 성 수 (인)
위 원 안 정 호 (인)

초 록

맵리듀스는 클라우드 데이터센터에서 대용량 데이터 처리를 위해 널리 사용되는 분산 처리 프로그래밍 모델이다. 맵리듀스는 맵, 셔플, 리듀스의 3단계로 구성된다. 하둡 맵리듀스는 맵리듀스 프로그래밍 모델을 구현한 프레임워크 중 가장 많이 쓰이는 것 중 하나이다.

현재 하둡 맵리듀스의 셔플 단계는 동일 데이터의 중복된 읽기/쓰기로 대량의 I/O를 발생시키며, 네트워크 전송에 의한 긴 지연을 발생시킨다.

이 문제를 해결하기 위하여 본 논문에서는 SSD 기반 하둡 맵리듀스 시스템에서 데이터 주소 기반의 셔플 메커니즘을 제안한다. 데이터 주소 기반의 셔플 메커니즘은 (1) 데이터 주소 기반 정렬 방법, (2) 데이터 주소 기반 병합 방법과 (3) 맵 출력 데이터 선 전송 방법으로 구성된다. 이는 임의 읽기/쓰기 속도가 빠른 SSD의 특징을 활용하여 대량의 중간 데이터 전체를 정렬하는 대신 작은 크기의 데이터 주소정보만을 정렬하고, 맵 태스크에서 리듀스 태스크로의 데이터 전송을 맵 출력 파일이 아닌 스필 파일과 주소정보 파일로 함으로써 네트워크 전송 시작을 앞당길 수 있는 메커니즘이다.

이를 활용하여 (1) 로컬 저장장치에 대한 읽기/쓰기 횟수와 데이터 양을 줄이고, (2) 네트워크 전송을 위한 지연 시간을 줄여 하둡 맵리듀스 셔플 단계의 수행시간을 단축하였다.

데이터 주소 기반의 셔플 메커니즘을 하둡 1.2.1에 구현하고 실험하였다. 실험결과 데이터 주소 기반의 셔플 메커니즘은 Terasort 벤치마크와 Wordcount 벤치마크의 평균 실행시간이 각각 8%와 1% 감소시킴을 보였다.

주요어 : Hadoop; MapRedue; SSD; Shuffle Mechanism; Distributed Computing

학 번 : 2013-23758

목 차

표 목차

그림 목차

제 1 장 서 론

정보통신 기술의 급격한 발달과 함께 도래한 디지털 시대에는 스마트 기기, 인터넷, 미디어 등 다양한 곳으로부터 무수히 많은 데이터가 생성되고 있다. 더욱이 최근 부각되고 있는 사물인터넷 환경에서는 네트워크에 연결된 기기들의 폭발적인 증가로 인하여 생성되는 데이터의 양이 기하급수적으로 증가할 것으로 예상된다.

이렇게 다양한 곳으로부터 무수히 많이 생성되는 대용량 데이터는 클라우드 데이터센터에서 통하여 처리가 된다. 클라우드 데이터센터에서는 이러한 대용량 데이터를 빠르게 처리하기 위하여 데이터를 여러 서버에서 나누어 병렬적으로 처리하는 분산 처리 시스템을 이용한다.

맵리듀스(MapReduce)는 대용량 데이터의 분산 처리를 위해 구글에서 정의한 대표적인 분산 처리 프로그래밍 모델이며, 하둡(Hadoop)은 맵리듀스 프로그래밍 모델을 구현한 프레임워크 중 가장 많이 쓰이는 것 중 하나이다[1, 2]. 하둡은 아파치에서 개발하고 배포하는 공개 소프트웨어로써 사용하기 쉽고 확장이 용이하고 안정성이 높으며 무료인 장점으로 인하여 널리 사용되고 있다. 하둡은 분산 파일시스템인 하둡 분산 파일시스템(HDFS, Hadoop Distributed File System)과 분산 처리 엔진인 하둡 맵리듀스(Hadoop MapReduce)로 구성되어 있다.

하둡 분산 파일시스템은 데이터를 여러 노드에 분산하여 저장하게 해주는 파일시스템이다[3]. 분산 저장을 병렬적으로

수행함으로써 데이터를 읽고 쓰는 속도를 높이고, 데이터의 복사본(Replica)을 여러 개 저장함으로써 데이터의 안정성을 높여준다.

하둡 맵리듀스는 맵리듀스 프로그래밍 모델에 따라 여러 노드에서 데이터를 분산 처리한다. 이 처리는 맵(Map), 셔플(Shuffle), 리듀스(Reduce)의 3단계로 수행된다. 맵 단계에서는 사용자가 정의한 맵 알고리즘을 이용하여 입력 데이터를 여러 노드에서 병렬적으로 처리한다. 셔플 단계에서는 맵 단계에서 생성된 중간 데이터(Intermediate Data)를 정렬하고 리듀스 단계를 담당하는 여러 노드에 분배한다. 리듀스 단계에서는 할당 받은 중간 데이터를 사용자가 정의한 리듀스 알고리즘을 사용하여 처리한 후 최종 결과를 생성한다. 데이터를 처리하는 과정에서 맵 단계와 리듀스 단계는 사용자가 정의한 알고리즘에 따라 다양한 동작을 수행하는 반면 셔플 단계에서는 하둡 맵리듀스에서 정의한 일관된 동작을 수행한다.

셔플 단계에서 이루어지는 동작 중에는 중간 데이터의 정렬이 있다. 이 정렬의 목적 중 하나는 데이터를 저장장치 내에 순차적으로 위치시킴으로써 리듀스 단계의 사용자 알고리즘이 수행될 때 성능을 향상시키고자 하는 것이다. 이는 디스크의 기계적인 회전으로 인하여 임의(Random) 읽기/쓰기 속도가 느린 하드 디스크 드라이브(HDD, Hard Disk Drive)를 고려한 설계이다. 그러나 이로 인하여 대량의 데이터에 대한 읽기/쓰기가 여러 번 발생한다. 또한 맵 단계의 정렬이 완료된 후 중간 데이터를 리듀스

태스크로 보내기 때문에 네트워크 전송에 의한 지연 시간이 길어진다.

그런데 최근 클라우드 데이터센터에서 많이 사용되기 시작한 SSD(Solid State Drive)는 기계적인 동작이 필요 없는 플래시 메모리(Flash Memory)로 구성되어 있어 임의 읽기/쓰기 속도가 순차(Sequential) 읽기/쓰기 속도에 비해 크게 감소되지 않는다. 따라서 하둡 시스템의 저장장치로 SSD를 사용시 중간 데이터를 순차적으로 저장할 필요가 없다.

본 논문에서는 위와 같은 SSD의 특성을 활용하여 셔플 단계의 수행시간을 단축시키는 방법을 제안한다. 제안하는 메커니즘은 대량의 중간 데이터 전체를 정렬하는 대신 상대적으로 훨씬 작은 크기의 데이터 주소 정보만을 정렬한다. 이는 (1) 데이터 주소정보 기반 정렬 방법, (2) 데이터 주소정보 기반 병합 방법, (3) 맵 출력 데이터 선 전송 방법으로 구성된다.

제안된 메커니즘을 하둡 맵리듀스 1.2.1에 구현하고 우분투 14.04가 설치된 5대의 서버로 구성된 환경에서 실험을 수행하였다. 실험 결과 제안된 메커니즘을 적용한 하둡 맵리듀스의 평균 실행시간이 최대 8%만큼 감소했다.

본 논문의 나머지 부분은 다음과 같이 구성되어 있다.

2장에서는 본 연구와 관련된 연구로써 하둡 맵리듀스의 성능을 개선하기 위한 연구들과 SSD 기반 하둡 시스템을 기반으로 하는 하둡 연구들을 소개한다.

3장에서는 본 연구의 배경 지식으로써 맵리듀스 프로그래밍

모델, 하둡 맵리듀스 및 SSD의 특성에 대하여 설명한다.

4장에서는 시스템 모델로써 SSD 기반의 하둡 맵리듀스 시스템과 하둡 맵리듀스의 셔플 단계에 대하여 설명한다.

5장에서는 문제 정의로써 기존 하둡 맵리듀스의 셔플 단계의 문제점인 동일 데이터의 중복 저장 문제와 네트워크 전송의 지연 문제에 대하여 살펴본다.

6장에서는 해결 방안으로써 기존 하둡 맵리듀스의 셔플 단계의 문제점들을 해결할 수 있는 데이터 주소 기반 셔플 메커니즘에 대하여 설명한다.

7장에서는 실험 및 평가로써 제안된 데이터 주소 기반 셔플 메커니즘을 평가하기 위한 실험 방법과 결과에 대하여 설명한다.

8장에서는 결론으로써 본 연구의 내용을 정리하고, 향후 연구방향에 대하여 살펴본다.

제 2 장 관련 연구

이번 장에서는 본 연구와 관련된 연구로써 하둡 맵리듀스의 성능을 개선하기 위한 연구들과 SSD 기반 하둡 시스템을 기반으로 하는 하둡 연구들을 소개한다.

2.1 하둡 맵리듀스 성능 개선 연구

하둡 맵리듀스의 성능을 개선을 위한 연구들로는 다음과 같은 것들이 있다.

Yandong Wang외 4인은 하둡 맵리듀스의 셔플 단계와 리듀스 단계의 성능을 개선하기 위하여 네트워크를 이용한 병합 메커니즘에 대한 연구를 하였다[10]. 이를 위하여 맵 태스크의 맵 출력 파일들을 리듀스 태스크로 복사한 후 병합하지 않고, RDMA (Remote Direct Memory Access)가 지원되는 네트워크를 사용하여 병합한 결과를 리듀스 태스크에 저장함으로써 저장장치에 대한 I/O를 줄였다. 또한 이를 사용하여 리듀스의 병합과 리듀스 함수 수행이 병렬적으로 처리되게 함으로써 셔플 단계와 리듀스 단계의 처리속도를 개선하였다.

Abhishek Verma외 4인은 맵리듀스 셔플 단계에서 사용자의 맵 알고리즘에 의한 결과인 <키, 값> 쌍들을 정렬하지 않는 메커니즘에 대한 연구를 하였다[13]. 이를 위해 맵 태스크에서 <키, 값> 쌍들을 정렬하지 않고 리듀스 태스크로 전송한다. 따라서

사용자의 리듀스 알고리즘은 정렬되어 있지 않은 키를 순차적으로 처리하게 된다. 정렬된 결과가 필요 없는 응용의 경우에는 전체 실행시간이 감소하여 성능이 개선되지만, 많은 응용의 경우 정렬된 결과가 필요하고, 이 경우 성능 개선이 거의 되지 않거나 오히려 나빠지는 경우도 있다.

Jingui Li외 3인은 맵 태스크의 출력 데이터를 리듀스 태스크로 전송하기 위해 각 노드에 셔플 서비스 컴포넌트를 만들었다[14]. 즉, 하나의 노드 내에서 리듀스 태스크 별로 가져오던 맵 출력 데이터를 하나의 셔플 서비스가 가져오게 되었다. 이로 인하여 데이터 전송 요청 수가 줄어 이를 처리하기 위한 비용이 줄게 되었으며, 맵 출력 데이터를 임의로 읽는 빈도를 줄여 읽기 성능을 향상시킴으로써 셔플 성능을 개선하였다.

Yanfei Guo외 2인은 셔플 단계에서 데이터 전송을 관리하는 별도의 셔플 서비스를 만들었다[15]. 이 셔플 서비스를 이용하여 맵 단계에게 출력 데이터가 생성되면 리듀스 태스크가 시작하기 전에 리듀스 태스크가 수행될 노드로 미리 전송하도록 하였다. 이를 통하여 셔플에서 네트워크에 의한 지연시간을 감소시킴으로써 셔플 성능을 개선하였다.

2.2 SSD 기반 하둡 시스템 연구

SSD 기반 하둡 시스템을 기반으로 하는 하둡 연구들로는 다음과 같은 것들이 있다.

Sangwhan Moon외 2인은 SSD를 사용하여 효율적인 하둡 시스템을 구성하는 방법을 연구하였다[19]. 기존 HDD에 비하여 고성능인 SSD의 빠른 읽기/쓰기 속도를 활용하기 위하여 충분한 대역폭의 네트워크를 사용할 것을 제안하였다. 또한 SSD를 중간 데이터 저장을 위한 저장장치로 이용하고 HDD를 하둡 분산 파일시스템의 데이터 저장을 위한 저장장치로 사용하는 것이 가장 높은 가격 대비 성능을 나타내므로, SSD를 중간 데이터 저장을 위한 저장장치로 사용할 것을 제안하였다.

Karthik Kambatla외 1인은 SSD를 사용하는 하둡 시스템에서 워크로드에 따른 성능 향상과 cost-per-performance에 대해 연구하였다[20]. 하둡 시스템에서 SSD를 사용하는 것이 HDD 대비 성능이 최대 70%까지 좋아지지만 cost-per-performance는 2.5배 높은 것을 확인하였다. 따라서 성능과 가격을 고려하여 SSD를 적용할 것을 제안하였다. 또한 하둡 맵리듀스의 워크로드의 특성에 따라 SSD를 사용한 하둡 시스템의 성능이 다름을 확인하였으며, 하둡 시스템의 워크로드의 특성을 고려하여 SSD를 적용할 것을 제안하였다.

Sungyong Ahn외 3인은 SSD를 적용한 가상화 환경에서 하둡 시스템의 성능에 대하여 연구하였다[21]. 가상화 환경에서 하둡 시스템은 CPU의 부하보다 I/O의 부하에 의하여 성능이 감소됨을 확인하였고, 그 이유는 가상화에 의하여 I/O 패턴이 단편화되고 무작위화되기 때문으로 분석하였다. 이로 인하여 SSD를 사용하는 하둡 시스템이 HDD를 사용하는 하둡 시스템에서보다 가상화에

의한 부하가 작음을 확인하였다. 또한 SSD를 사용하는 하둡 시스템은 VM 개수가 증가하여도 좋은 성능을 유지함을 확인하였다.

제 3 장 배 경

이번 장에서는 본 논문에서 제안하는 메커니즘의 이해를 돕기 위한 배경 지식으로써 맵리듀스 프로그래밍 모델, 하둡 맵리듀스 및 SSD의 특성에 대해 자세히 설명한다.

3.1 맵리듀스 프로그래밍 모델

맵리듀스는 대용량 데이터의 분산 처리를 위하여 구글에서 정의한 대표적인 분산 처리 프로그래밍 모델이다. 이 프로그래밍 모델은 대용량 데이터를 고성능 서버가 아닌 다수의 범용 서버나 PC를 이용하여 병렬 처리할 수 있도록 설계되었다. 이를 위해 입출력 데이터는 맵리듀스 시스템을 구성하는 노드 들에 분산되어 저장되며, 데이터 처리 도중 발생하는 중간 데이터는 각 노드의 로컬 저장장치에 임시 저장된다. 맵리듀스 프로그래밍 모델을 이용하여 데이터를 처리하는 응용 프로그램을 작성하는 개발자는 분산 처리 메커니즘에 대한 상세한 지식이 없어도 프로그래밍 모델의 템플릿에 맞게 맵 함수와 리듀스 함수만 작성하면 맵리듀스의 분산 처리 메커니즘에 의해 맵리듀스 시스템을 구성하는 노드들에 분산되어 처리된다.

맵리듀스 프로그래밍 모델은 그림 1과 같이 맵, 셔플, 리듀스의 3단계로 나누어 데이터를 처리한다. 맵 단계에서는 입력 데이터를 읽어와 사용자가 정의한 맵 함수의 알고리즘에 의해 처리한 후

맵리듀스의 중간 데이터를 출력한다. 이 중간 데이터는 키, 값의 쌍인 <K, V> 쌍의 리스트로 구성되어 있다. 맵 단계는 여러 노드에서 나누어 분산하여 수행된다. 셔플 단계에서는 중간 데이터인 <K, V> 쌍의 리스트를 K를 기준으로 정렬하고 리듀스 단계를 수행하는 노드들에게 분배한다. 리듀스 단계에서는 여러 노드에서 전달받은 <K, V> 쌍의 리스트들을 병합한 후, 사용자가 정의한 리듀스 함수의 알고리즘에 의해 처리를 하여 최종 출력 데이터를 생성한다.

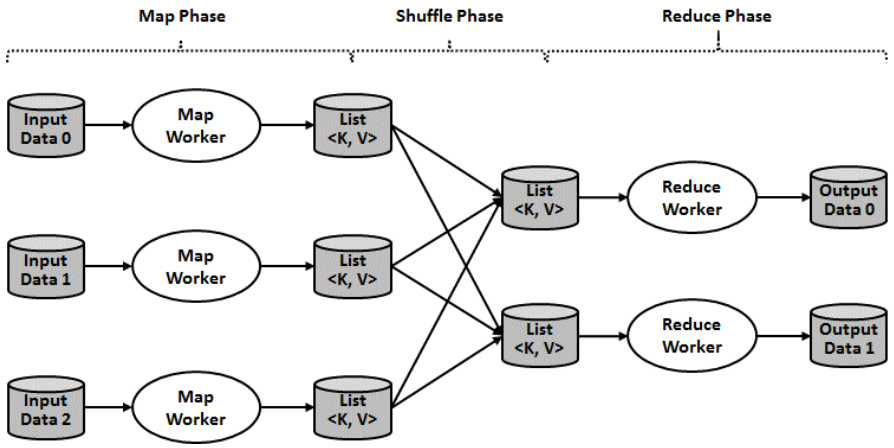

그림 1 맵리듀스 프로그래밍 모델

3.2 하둡 맵리듀스

하둡 맵리듀스는 대용량 데이터를 빠르고 효율적으로 처리하기 위한 분산처리 엔진이다. 하둡 맵리듀스는 그림 2와 같이 1개의 마스터 노드(Master node)와 다수의 슬래이브 노드(Slave node)들로 구성되어 있다. 사용자는 데이터 처리를 잡(Job)이라는 단위로 마스터 노드에 요청한다. 마스터 노드는 하나의 잡을 여러 개의 태스크로 분할하고 이를 슬래이브 노드들에 할당하여 수행되도록 한다. 태스크의 종류는 두 가지로 맵 태스크(Map Task)와 리듀스 태스크(Reduce Task)이다. 맵 태스크의 개수는 잡의 입력 데이터의 블록(Block)의 개수이다. 즉 맵 태스크 당 하나의 블록을 처리한다. 리듀스 태스크의 개수는 사용자의 설정에 의해 결정된다. 슬래이브 노드들은 할당 받은 태스크들을 수행하며 진행상황을 주기적으로 마스터 노드에 보고한다.

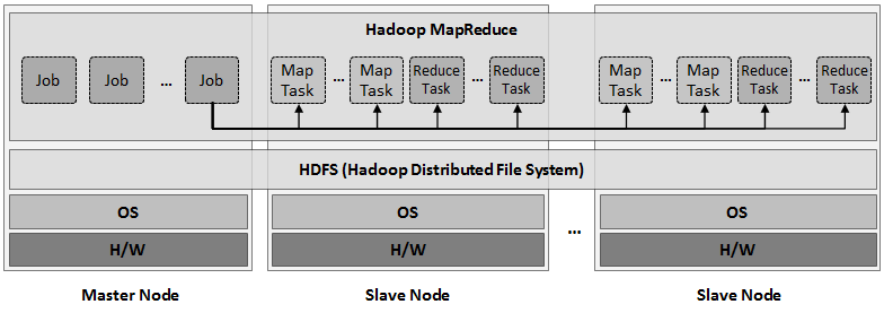

그림 2 하둡 맵리듀스의 구성

하둡 맵리듀스의 데이터 처리는 그림 3과 같이 맵 태스크와 리듀스 태스크를 순서대로 거치면서 수행된다. 이 데이터 처리는 맵, 셔플, 리듀스의 3단계로 수행된다. 맵 단계에서는 잡의 입력데이터를 하둡 분산 파일시스템으로부터 읽어와 사용자가 정의한 맵 함수를 수행하고 그 결과로 중간 데이터를 생성한다. 셔플 단계는 맵 태스크에서의 수행과 리듀스 태스크에서의 수행으로 구분되는데, 맵 태스크에서는 해당 맵 태스크의 중간 데이터를 키 값을 기준으로 정렬하여 맵 출력 파일(Map Output File)을 생성한다. 이 맵 출력 파일은 각 리듀스 태스크에서 나누어 처리하도록 데이터 영역이 분할되어 있다. 리듀스 태스크에서는 모든 맵 태스크의 맵 출력 파일들로부터 해당 리듀스 태스크가 처리할 데이터를 네트워크를 통하여 복사한 후 하나의 파일로

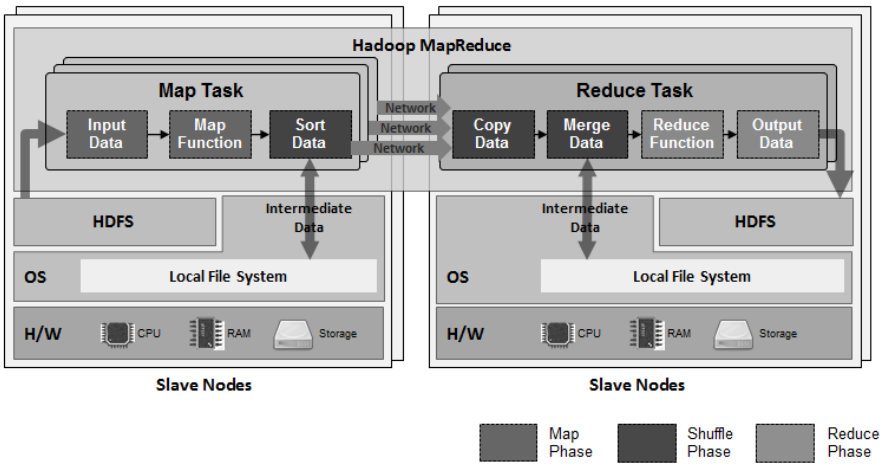

그림 3 하둡 맵리듀스의 데이터 처리

병합한다. 이 때 키 값을 기준으로 정렬된 통합 파일을 생성하기 위하여 정렬된 순서를 유지하며 병합한다. 리듀스 단계에서는 정렬된 중간 데이터를 입력으로 받아 사용자가 정의한 리듀스 함수를 수행한 후 최종 출력 데이터를 생성한다. 정렬된 중간 데이터에서 동일 키에 대한 처리를 수행하고, 최종 출력 데이터는 하둡 분산 파일시스템을 이용하여 저장한다.

3.3 SSD (Solid State Drive) 특성

SSD는 반도체를 이용하여 데이터를 저장하는 저장장치로써 데이터가 저장되는 플래시 메모리(Flash Memory)와 플래시 메모리를 제어하고 호스트(Host) 컴퓨터와 인터페이스를 담당하는 컨트롤러(Controller)로 구성된다.

SSD는 내부적으로 다수의 플래시 메모리를 사용하여 데이터의 읽기/쓰기를 병렬적으로 수행하여 하드디스크 드라이브에 비하여 높은 성능을 발휘한다. 더욱이 하드디스크 드라이브와 같이 물리적인 디스크를 사용하지 않으므로 연속되지 않은 주소의 데이터를 처리할 때 디스크의 회전이 없다. 따라서 임의 읽기/쓰기 속도가 순차 읽기/쓰기 속도에 비하여 급격히 나빠지지 않는다.

표 1은 fio 벤치마크 툴을 이용하여 SSD와 하드디스크 드라이브의 순차 읽기/쓰기 속도와 임의 읽기/쓰기 속도를 측정한 것이다[5]. 측정 결과 하드디스크 드라이브의 임의 읽기/쓰기 속도는 순차 읽기/쓰기 대비 약 1/80인 반면, SSD의 임의

읽기/쓰기 속도는 순차 읽기/쓰기 대비 약 2/3 정도로 성능이 많이 감소되지 않음을 알 수 있다.

표 1 SSD와 HDD 읽기/쓰기 속도

(단위: MB/s)

		SSD	HDD
Sequential	Read	459.8	173.6
	Write	435.1	172.2
Random	Read	359.5	2.3
	Write	301.4	2.0

이러한 SSD의 뛰어난 성능에도 불구하고 하드디스크 대비 비싼 가격으로 인하여 그 사용이 제한적이었다. 그러나 플래시 메모리의 집적도 향상으로 인하여 플래시 메모리의 가격이 꾸준히 하락하여 SSD의 가격 또한 하락하였다. 이로 인하여 데이터센터에서도 SSD의 사용이 꾸준히 증가하고 있다.

제 4 장 시스템 모델

이번 장에서는 본 연구의 시스템 모델로써 SSD 기반의 하둡 맵리듀스 시스템과 하둡 맵리듀스의 셔플 단계에 대하여 자세히 설명한다.

4.1 SSD 기반의 하둡 시스템

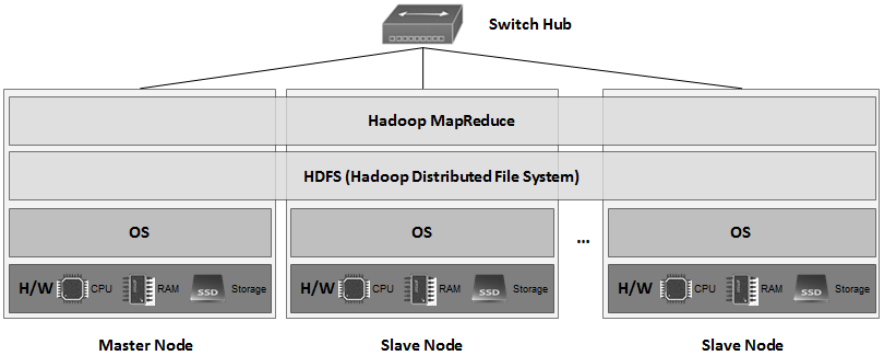

그림 4 SSD 기반의 하둡 시스템

SSD 기반의 하둡 시스템은 그림 4와 같이 마스터 노드, 슬래이브 노드, 스위치 허브로 구성된다. 하나의 SSD 기반의 하둡 시스템 내에는 1개의 마스터 노드가 있으며, 슬래이브 노드는 여러 개있다. 스위치 허브는 시스템의 구성에 따라 여러 개 있을 수 있다. 마스터 노드와 슬래이브 노드는 각각 독립적인 CPU, 메모리, 저장장치를 가지고 있다. 특히 저장장치로는 SSD를 사용한다.

또한 이러한 H/W를 기반으로 독립적인 운영체제가 설치되어 있다. 운영체제의 로컬 파일시스템을 이용하여 SSD에 읽기/쓰기를 수행할 수 있다. 반면 하둡 프레임워크를 구성하는 하둡 분산 파일시스템과 하둡 맵리듀스는 모든 노드들을 통합하여 설치된다. 이를 통하여 노드들간의 태스크 분배, 데이터 전송 등의 연동이 가능하다.

4.2 하둡 맵리듀스의 셔플 단계

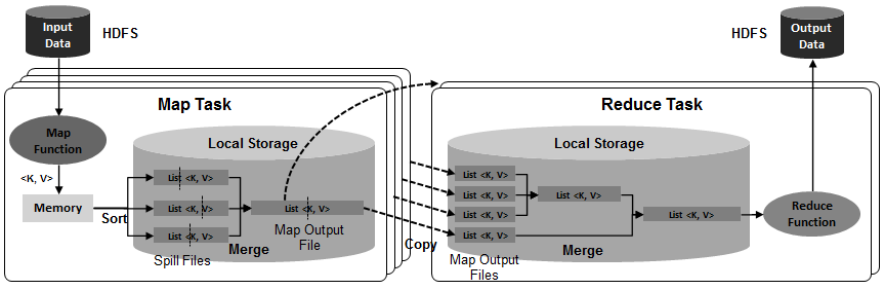

그림 5 하둡 맵리듀스의 셔플 단계

하둡 맵리듀스의 셔플 단계는 그림 5와 같다. 맵 태스크의 맵 함수는 입력 데이터를 처리하여 <K, V> 쌍의 형태로 출력 한다. 맵 태스크는 맵 함수의 출력인 <K, V> 쌍들을 일차적으로 저장하기 위한 메모리 버퍼를 유지한다. <K, V> 쌍들은 미리 설정된 임계 값에 도달할 때까지 메모리 버퍼에 저장되며, 임계 값에 도달 시 저장된 <K, V> 쌍들을 메모리 버퍼 내에서 정렬한 후 맵 태스크가 수행되는 노드의 저장장치에 파일형태로 저장을 한다. 정렬은 K를

기준으로 이루어지며, 먼저 해당 <K, V> 쌍이 처리될 리듀스 태스크를 정하기 위한 정렬과 동일 리듀스 태스크에서 처리되는 <K, V> 쌍들 내에서 정렬하는 두 단계로 이루어진다. 이렇게 두 단계에 의해 정렬 완료된 <K, V> 쌍들이 저장되는 파일은 스필(Spill) 파일이라고 한다.

정렬이 진행됨과 동시에 맵 알고리즘에 의해 생성되는 <K, V> 쌍을 메모리 버퍼의 나머지 부분에 계속 저장한다. 메모리 버퍼가 가득 차는 경우, 스필 파일 생성이 완료 때까지 맵 알고리즘의 수행이 지연된다. 메모리 버퍼의 <K, V> 쌍들의 크기가 임계 값에 도달할 때마다 메모리 버퍼내의 정렬과 저장장치에 저장이 이루어지며 여러 개의 스필 파일이 생성될 수 있다. 이렇게 생성된 스필 파일들은 맵 알고리즘의 <K, V> 쌍의 출력이 완료된 후 하나의 최종 맵 출력 파일로 병합이 된다. 스필 파일들을 병합할 때 처리될 리듀스 태스크에 의한 정렬을 유지하면서 동일 리듀스 태스크에서 처리되는 <K, V> 쌍들을 하나의 정렬된 순서로 병합한다. 따라서 동일 리듀스를 위해 맵 알고리즘이 생성한 <K, V> 쌍들은 K를 기준으로 정렬되어 순차적으로 저장되어 있다.

스필 파일들의 병합이 완료되면 리듀스 태스크는 맵 태스크의 맵 출력 파일로부터 해당 리듀스 태스크가 처리할 <K, V> 쌍들을 네트워크를 통해 가져와 리듀스 태스크가 수행되는 노드의 저장장치에 저장한다. 리듀스 태스크는 다른 모든 맵 태스크의 맵 출력 파일을 가져와 저장하며, 저장된 맵 출력 파일들을 하나의 파일로 병합한다. 맵 출력 파일들의 최소 K들간의 비교 연산으로

인한 부하를 줄이기 위하여 한번에 병합할 수 있는 파일의 개수가 제한된다. 따라서 맵 출력 파일의 개수에 따라 여러 번에 걸쳐 병합이 수행된다. 병합 시 K를 기준으로 하는 정렬을 유지한다. 따라서 최종 병합 파일은 K를 기준으로 정렬되어 있다. 이렇게 전체 정렬 및 병합이 완료된 결과를 리듀스 함수를 이용하여 처리하여 최종 출력 데이터를 생성한다.

제 5 장 문제 정의

이번 장에서는 하둡 맵리듀스에서 셔플 단계를 분석하고 기존 하둡 맵리듀스에서 데이터 처리를 지연시키는 문제점인 동일 데이터의 중복 읽기/쓰기 문제와 네트워크 전송의 지연 문제에 대하여 설명한다.

5.1 동일 데이터의 중복 읽기/쓰기 문제

먼저 하둡 맵리듀스의 셔플 단계 중 맵 태스크를 살펴보자. 그림 6과 같이 맵 함수에서 생성되는 <K, V> 쌍들의 전체 정렬을 위해 임시로 스필 파일을 만든 후 이를 병합한다. 이로 인하여 각 <K, V> 쌍들은 저장장치에 스필 파일로 쓰기 1회, 스필 파일에서 읽기 1회, 병합 파일로 쓰기 1회를 수행한다. 따라서 총 쓰기

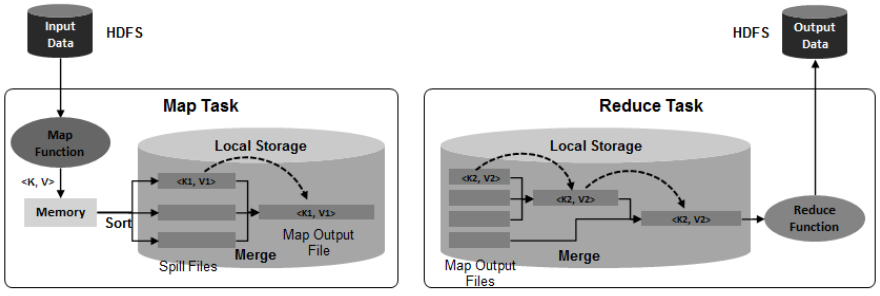

그림 6 동일 데이터의 중복 읽기/쓰기

2회와 읽기 1회가 발생한다.

또한 리듀스 태스크에서는 많은 맵 출력 파일들을 병합해야 한다. 이 때, K들간의 비교 연산으로 인한 부하를 줄이기 위하여 여러 단계에 걸쳐 병합이 수행된다. 병합이 수행될 때마다 <K, V> 쌍들의 읽기 1회와 쓰기 1회가 추가된다.

저장장치로의 I/O는 메모리의 접근보다 수백 배 이상 오래 걸리는 동작으로 동일한 <K, V>의 중복적인 읽기/쓰기는 하둡 맵리듀스의 셔플 단계의 주요 지연 원인 중 하나이다.

5.2 네트워크 전송의 지연 문제

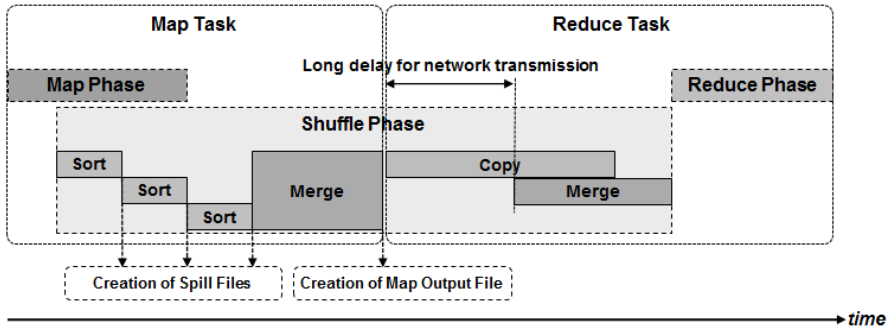

그림 7 네트워크 전송 지연

셔플 단계에서 맵 태스크의 맵 출력 파일들은 정렬된 형태로 리듀스 태스크에 전송된다. 이를 위하여 그림 7에서와 같이 맵 태스크에서 스필 파일의 병합이 완료된 후에 전송이 시작된다. 이로 인하여 네트워크를 통한 데이터 전송이 완료될 때까지 맵 단계

완료와 리듀스 단계 시작 사이에 긴 대기시간이 발생한다. 이러한 맵 태스크와 리듀스 태스크 사이의 긴 네트워크 전송 지연은 하둡 맵리듀스의 셔플 단계에서의 주요 지연 원인 중 하나이다.

제 6 장 데이터 주소 기반 셔플 메커니즘

이번 장에서는 기존 하둡 맵리듀스 셔플 단계의 지연 원인을 SSD 기반의 하둡 맵리듀스 시스템에서 개선하기 위하여 데이터 주소 기반 셔플 메커니즘을 제안한다. 데이터 주소 기반 셔플 메커니즘은 (1) 데이터 주소정보 기반 정렬 방법, (2) 데이터 주소정보 기반 병합 방법, (3) 맵 출력 데이터 선 전송 방법으로 구성된다.

6.1 데이터 주소 기반 정렬

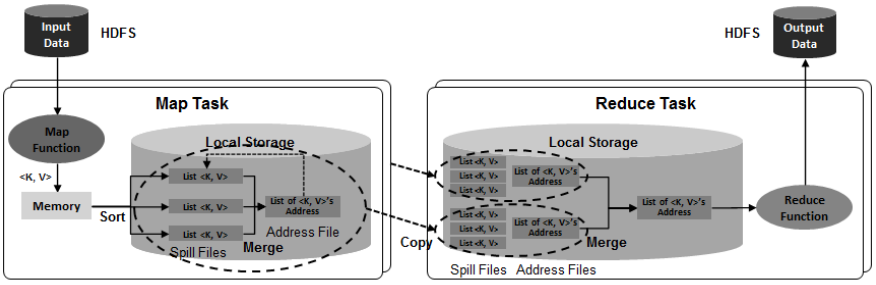

그림 8 데이터 주소 기반 셔플

앞서 설명했듯이 SSD 기반 시스템에서는 리듀스 태스크에 전송되는 <K, V> 쌍들이 반드시 순차적으로 저장되어 있을 필요는 없다. 따라서 대량의 <K, V> 쌍들을 정렬하는 대신 그림 8과 같이 <K, V> 쌍에 대한 주소정보를 생성하여 이를 정렬한다. <K, V> 쌍에 대한 주소정보는 스필 파일의 번호와 스필 파일에서의 오프셋

위치를 가리킨다. 이 주소정보는 추가적인 메모리 버퍼나 저장장치에 저장된다. 그러나 주소정보만 가지고 있으므로 그 크기는 작다. 리듀스 태스크에 정렬 정보를 전달하기 위해 이 주소정보가 전송된다. 기존에 정렬된 형태로 전송되던 <K, V> 쌍들은 맵 태스크에서 생성된 스필 형태로 전달한다. 리듀스 태스크는 <K, V> 쌍들에 대한 주소정보를 참조하여 스필 파일들에서 정렬된 순서로 <K, V> 쌍들을 읽을 수 있다.

6.2 데이터 주소 기반 병합

기존 하둡 맵리듀스의 셔플 메커니즘에서는 맵 태스크의 병합 과정에서 스필 파일들의 <K, V> 쌍을 하나의 병합된 파일로 생성하였다. 그러나 제안하는 메커니즘에서는 각 스필 파일들의 <K, V> 쌍에 대한 주소정보만 정렬하여 해당 맵 태스크의 출력 데이터를 정렬하게 된다. 따라서 기존에 쓰기 2회, 읽기 1회씩 발생하던 <K, V> 쌍들을 I/O 횟수가, K는 쓰기 1회, 읽기 1회로, 값은 쓰기 1회로 각각 줄어들게 된다. 주소정보를 파일로 저장하면 이에 대한 추가적인 쓰기가 필요하지만 주소정보는 <K, V> 쌍들에 비해 상대적으로 크기가 작고, 크기가 큰 <K, V> 쌍들의 I/O가 K는 쓰기 1회, 값은 읽기/쓰기 1회씩 감소하므로 저장장치에 대한 I/O 횟수와 크기가 감소하여 이로 인한 셔플단계의 지연을 줄일 수 있다.

또한 리듀스 태스크의 병합 과정에서 기존 하둡 맵리듀스의

셔플 메커니즘에서는 <K, V> 쌍당 K와 V에 대한 읽기와 쓰기가 각각 1회씩 발생하지만, 제안하는 데이터 주소 기반의 셔플 메커니즘에서는 위치정보와 K에 대한 읽기 1회와 위치정보 쓰기 1회가 발생한다. 그리고 기존 하둡 맵리듀스의 셔플 메커니즘에서 저장장치에 대한 접근 방식은 순차 읽기/쓰기와 유사한 반면, 제안하는 데이터 주소 기반의 셔플 메커니즘에서 저장장치에 대한 접근 방식은 임의 읽기/쓰기와 유사하다.

따라서 기존 하둡 맵리듀스의 셔플 메커니즘에서 <K, V> 쌍 1개에 대한 병합 시간은 다음과 같이 표현된다.

$$\frac{(S_{Key} + S_{Key})}{T_{SeqR}} + \frac{(S_{Key} + S_{Key})}{T_{SeqW}}$$

또한 제안하는 데이터 주소 기반의 셔플 메커니즘에서 <K, V> 쌍 1개에 대한 병합 시간은 다음과 같이 표현된다.

$$\frac{(S_{Addr} + S_{Key})}{T_{RanR}} + \frac{S_{Addr}}{T_{RanW}}$$

사용된 표기에 대한 요약은 표 2와 같다.

앞서 설명한 바와 같이 SSD의 임의 읽기/쓰기 속도는 순차 읽기/쓰기 속도 대비 약 2/3 정도로 큰 차이가 없으므로 제안하는 데이터 주소 기반 셔플 메커니즘의 병합 시간이 기존 메커니즘에 비해 빠르다. 그러나 하드디스크 드라이브는 임의 읽기/쓰기 속도가

순차 읽기/쓰기 속도 대비 약 1/80으로 상당히 느리므로 제안하는 메커니즘의 병합 시간이 기존 메커니즘에 비해 오히려 느려질 수 있다.

표 2 표기 요약

Notation	Description
S_{Key}	The size of key
S_{Value}	The size of value
S_{Addr}	The size of address
T_{SeqR}	The throughput of sequential read
T_{SeqW}	The throughput of sequential write
T_{RanR}	The throughput of random read
T_{RanW}	The throughput of random write

6.3 맵 출력 데이터 선 전송

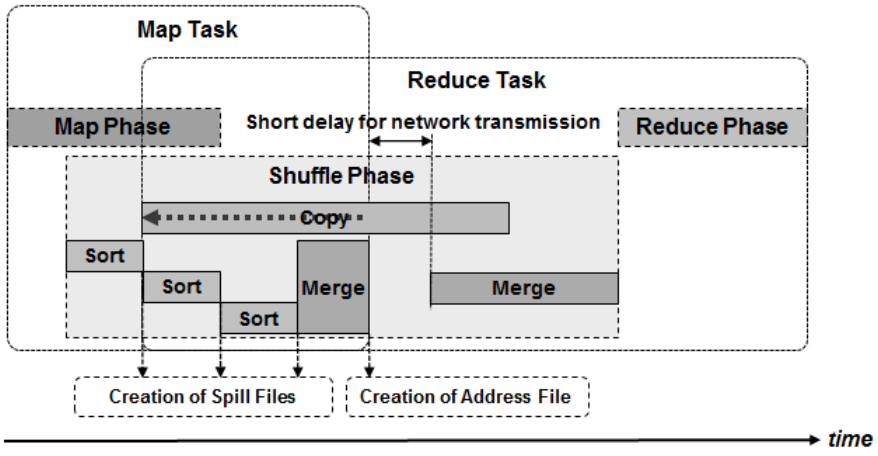

그림 9 맵 출력 데이터 선 전송

또한 기존의 셔플 메커니즘에서는 맵 태스크에서 스필 파일들의 병합이 완료된 후 병합된 맵 출력 파일을 리듀스 태스크로 전송하였다. 그러나 그림 9와 같이 제안하는 메커니즘에서는 리듀스 태스크에 <K, V> 쌍들을 스필 파일과 주소 파일 형태로 전송하고 병합된 데이터 만들지도 않으므로 병합이 완료된 후 전송을 시작할 필요가 없다. 따라서 스필이 완료될 때마다 리듀스 태스크가 이를 가져갈 수 있도록 한다. 이 스필 파일의 전송은 다음 스필 파일들의 처리 또는 주소 파일의 병합과 병렬적으로 수행될 수 있다. 따라서 맵 단계 완료 후 맵 출력 파일보다 크기가 훨씬 작은 주소 파일만 복사하면 되므로 리듀스 단계의 시작 전에 기존 대비 적은 양의 데이터만 전송하면 된다.

그러므로 맵 단계 완료와 리듀스 단계 사이의 대기시간이 감소하여 셔플단계의 지연을 줄일 수 있다.

제 7 장 실험 및 평가

이번 장에서는 본 연구에서 제안하는 데이터 주소 기반 셔플 메커니즘을 검증하기 위하여 수행한 실험에 대하여 기술한다. 먼저 실험 환경에 대하여 설명하고 그 다음 실험 결과를 제시한다.

7.1 실험 환경

제안된 데이터 주소 기반의 셔플 메커니즘을 실험하기 위한 환경은 표 3과 같다. 실험을 위한 하둡 클러스터는 총 5개의 노드로 구성하였으며, 1개의 마스터 노드와 4개의 슬래이브 노드로 사용하

표 3 실험 환경

Hardware	Cluster	5 servers
	CPU	Intel i7-4790 4 cores 3.6GHz
	Main Memory	DRAM DDR3 32GB
	Storage	SATA3.0 SSD 500GB SATA3.0 HDD 1TB
	Network	10Gbps Ethernet
Software	Kernel	Linux 3.13.0
	OS	Ubuntu 14.04
	Hadoop	Hadoop 1.2.1

였다. 저장장치는 SATA3.0 인터페이스 SSD와 HDD를 사용하였고, 네트워크 장비는 10기가비트(Gbps) 이더넷 스위치와 어댑터를 사용하였다.

소프트웨어는 리눅스 커널 3.13.0이 바탕인 우분투 14.04버전을 사용하였고, Hadoop 1.2.1을 수정하여 제안하는 메커니즘을 구현하였다.

하둡 환경변수의 설정은 표 4와 같다. 노드 당 수행 가능한 맵 태스크의 개수는 8개, 노드 당 수행 가능한 리듀스의 개수는 4개, 하둡 분산 파일시스템의 블록 크기는 128MB, Replication 인자는 3이다.

표 4 하둡 환경변수

The number of Map tasks per node	8
The number of Reduce tasks per node	4
HDFS block size	128MB
Replication factor	3

본 논문에서 제안된 데이터 주소 기반의 셔플 메커니즘을 검증하기 위하여 하둡에서 제공하는 벤치마크인 Terasort 벤치마크와 Wordcount 벤치마크를 사용하였다. 두 벤치마크는 하둡 프레임워크의 성능을 평가하기 위해 기존 연구에서 대표적으로 사용된 벤치마크들이다.

Terasort 벤치마크는 입력 데이터를 정렬하는 벤치마크로써 표 5와 같이 입력 데이터, 셔플 데이터, 출력 데이터의 크기가 동일하다. Wordcount 벤치마크는 입력 데이터에서 단어의 수를 세는 벤치마크로써 셔플 데이터와 출력 데이터의 크기가 입력 데이터에 비해서 상당히 작다[18]. 표 5의 수치는 입력 데이터 크기로 정규화되었다. 본 논문에서 제안하는 메커니즘은 Terasort 벤치마크와 같이 셔플 데이터가 많은 벤치마크에서 성능개선 효과가 크다.

표 5 벤치마크의 단계별 데이터 크기

Benchmark	Input Size	Shuffle Size	Reduce Size
Terasort	1	1	1
Wordcount	1	0.07	0.09

7.2 실험 결과 및 평가

기존 하둡 맵리듀스의 셔플 메커니즘과 본 논문에서 제안하는 데이터 주소 기반의 셔플 메커니즘을 Terasort 벤치마크에서 수행하여 실행시간을 비교한 결과는 그림 10과 같다. 데이터 주소 기반의 셔플 메커니즘을 적용 시 Terasort 벤치마크의 평균 실행시간이 8%만큼 감소하였다.

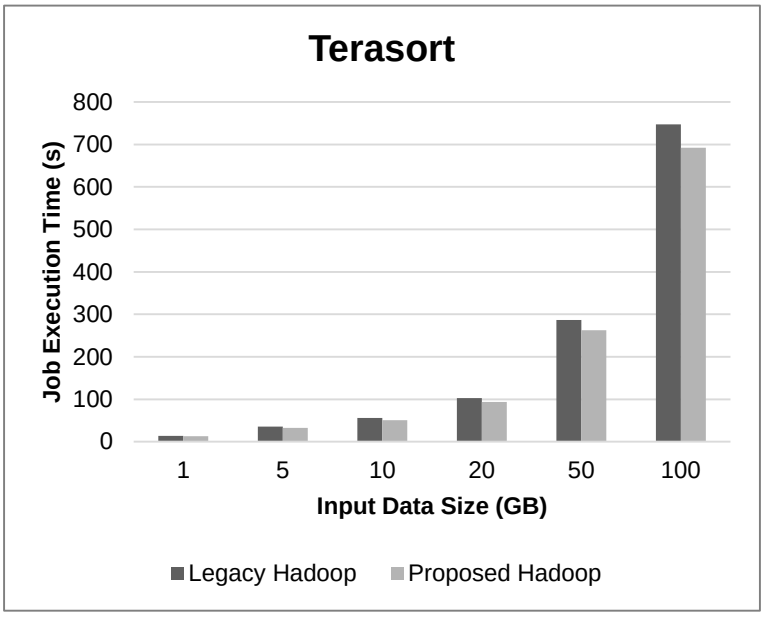

그림 10 Terasort 수행 결과

기존 하둡 맵리듀스의 셔플 메커니즘과 본 논문에서 제안하는 데이터 주소 기반의 셔플 메커니즘을 Wordcount 벤치마크에서 수행하여 실행시간을 비교한 결과는 그림 11과 같다. 데이터 주소 기반의 셔플 메커니즘을 적용 시 Terasort 벤치마크의 평균 실행시간이 1%만큼 감소하였다.

그 이유는 본 논문에서 제안하는 데이터 주소 기반의 셔플 메커니즘이 셔플 단계에서 데이터를 효율적으로 처리하는 방법이므로 입력 데이터에 비해 셔플 데이터가 작은 Wordcount 벤치마크에서는 개선 효과가 거의 없는 반면, 셔플 데이터 크기가 입력 데이터와 동일한 Terasort 벤치마크에서는 개선 효과가 큰 것을 알 수 있다.

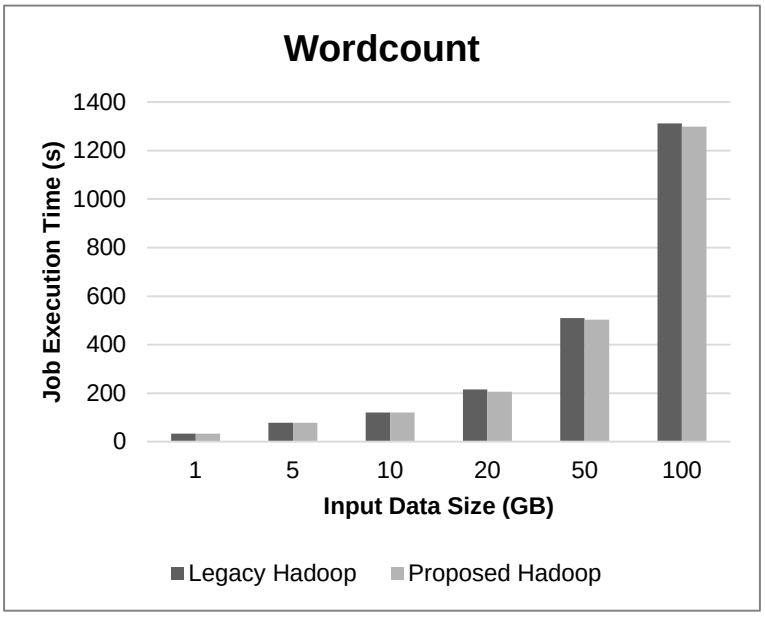

그림 11 Wordcount 수행 결과

HDD 기반 하둡 맵리듀스 시스템에서 기존 하둡 맵리듀스의 셔플 메커니즘과 본 논문에서 제안하는 데이터 주소 기반의 셔플 메커니즘을 Terasort 벤치마크에서 수행하여 실행시간을 비교한 결과는 그림 12와 같다. 데이터 주소 기반의 셔플 메커니즘을 적용 시 Terasort 벤치마크의 평균 실행시간이 4%만큼 증가하였다.

HDD 기반 하둡 맵리듀스 시스템에서 기존 하둡 맵리듀스의 셔플 메커니즘과 본 논문에서 제안하는 데이터 주소 기반의 셔플 메커니즘을 Wordcount 벤치마크에서 수행하여 실행시간을 비교한 결과는 그림 13과 같다. 데이터 주소 기반의 셔플 메커니즘을 적용 시 Wordcount 벤치마크의 평균 실행시간이 1%만큼 증가하였다.

제안하는 데이터 주소 기반의 셔플 메커니즘은 임의

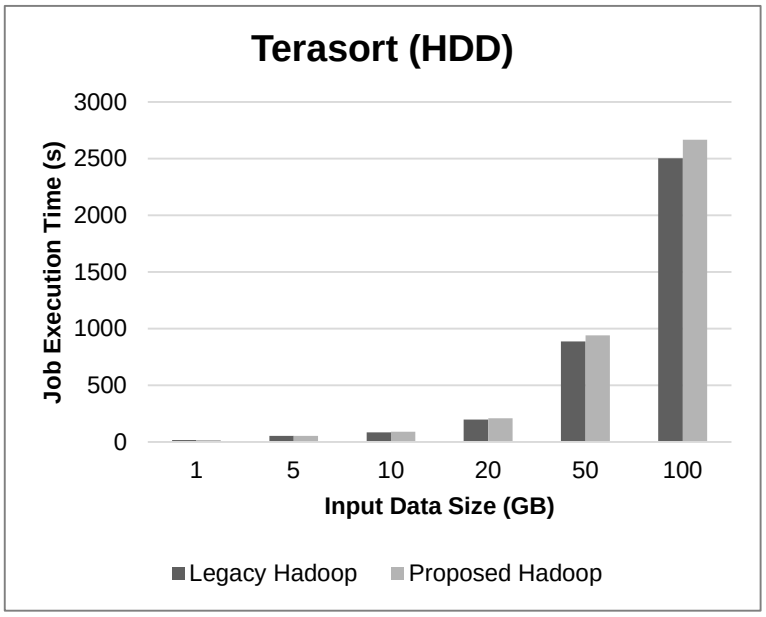

그림 12 Terasort 수행 결과 (HDD 기반 하둡 맵리듀스)

읽기/쓰기를 많이 수행하므로 저장장치로 임의 읽기/쓰기 속도가 느린 HDD를 사용시 Terasort 벤치마크의 실행시간이 오히려 증가하였음을 알 수 있다.

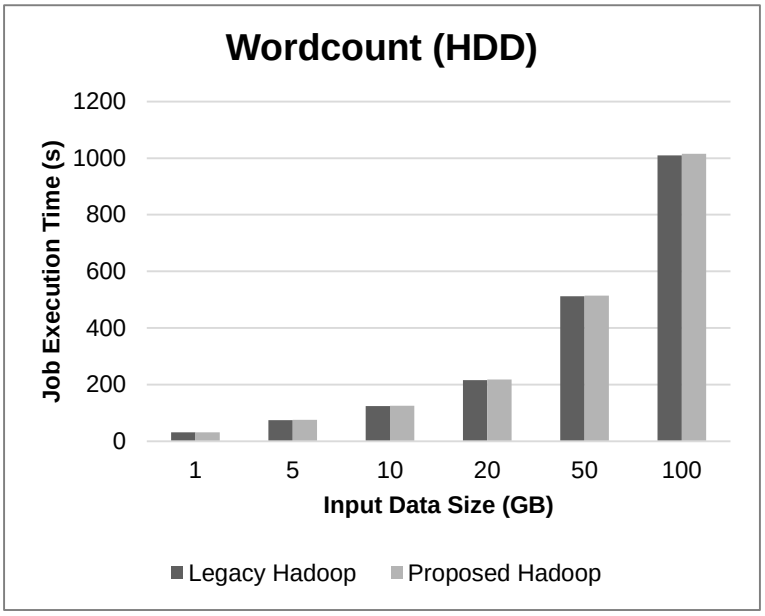

그림 13 Wordcount 수행 결과 (HDD 기반 하둡 맵리듀스)

제 8 장 결 론

본 논문을 통하여 기존 하둡 맵리듀스의 셔플 단계를 분석하고 문제점을 정의하였다. 그 문제점은 (1) 동일 데이터의 중복된 읽기/쓰기, (2) 네트워크 전송의 긴 지연시간으로 인한 하둡 맵리듀스의 셔플 단계의 수행시간 지연이다.

이 문제점을 해결하기 위하여 데이터 주소 기반의 셔플 메커니즘을 제안하였다. 데이터 주소 기반의 셔플 메커니즘은 (1) 데이터 주소 기반의 정렬 방법, (2) 데이터 주소 기반의 병합 방법, (3) 맵 출력 데이터의 선 전송 방법으로 구성된다. 이는 임의 읽기/쓰기 속도가 빠른 SSD의 특징을 활용하여 대량의 중간 데이터 전체를 정렬하는 대신 작은 크기의 데이터 주소정보만을 정렬하고, 맵 태스크에서 리듀스 태스크로의 데이터 전송을 맵 출력 파일이 아닌 스필 파일과 주소정보 파일로 함으로써 네트워크 전송 시작을 앞당길 수 있는 메커니즘이다. 이를 활용하여 (1) 로컬 저장장치에 대한 읽기/쓰기 횟수와 데이터 양을 줄이고, (2) 네트워크 전송을 위한 지연 시간을 줄여 하둡 맵리듀스 셔플 단계의 수행시간을 단축하였다.

제안하는 데이터 주소 기반의 셔플 메커니즘을 하둡 1.2.1에 구현하고 실험해 본 결과 Terasort 벤치마크와 Wordcount 벤치마크의 평균 실행시간이 각각 8%와 1% 감소하였다.

향후에는 SSD의 특성을 활용하여 맵 단계와 리듀스 단계의 성능 개선을 목표로 본 연구를 확장할 계획이다. 또한 하둡

맵리듀스 뿐만 아니라 하둡 분산 파일시스템에 대한 연구도 진행할 예정이다.

참고 문헌

[1] Jeffrey Dean, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[2] Apache Hadoop, "http://hadoop.apache.org"

[3] Dhruba Borthakur. "The hadoop distributed file system: Architecture and design." Hadoop Project Website 11.2007 (2007): 21.

[4] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google file system." ACM SIGOPS operating systems review. Vol. 37. No. 5. ACM, 2003.

[5] Fio benchmark, "http://freecode.com/projects/fio"

[6] Tom White. "Hadoop: The definitive guide." O'Reilly Media, Inc., 2012.

[7] Dawei Jiang, et al. "The performance of MapReduce: an in-depth study." Proceedings of the VLDB Endowment 3.1-2 (2010): 472-483.

[8] Kyong-Ha Lee, et al. "Parallel data processing with MapReduce: a survey." AcM sIGMoD Record 40.4 (2012): 11-20.

[9] Christos Doulkeridis, and Kjetil Nørvåg. "A survey of large-scale analytical query processing in MapReduce." The

VLDB Journal–The International Journal on Very Large Data Bases 23.3 (2014): 355–380.
[10] Yandong Wang, et al. "Hadoop acceleration through network levitated merge." Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011.
[11] Weikuan Yu, Yandong Wang, and Xinyu Que. "Design and evaluation of network-levitated merge for hadoop acceleration." Parallel and Distributed Systems, IEEE Transactions on 25.3 (2014): 602-611.
[12] Vaibhav Dhore, and Sonali R. Jagtap. "A Survey on Hierarchical Merge for Hadoop-A."
[13] Abhishek Verma, et al. "Breaking the MapReduce stage barrier." Cluster computing 16.1 (2013): 191-206.
[14] Jingui Li, et al. "Improving the Shuffle of Hadoop MapReduce." Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. Vol. 1. IEEE, 2013.
[15] Yanfei Guo, Jia Rao, and Xiaobo Zhou. "iShuffle: Improving Hadoop Performance with Shuffle-on-Write." ICAC. 2013.
[16] Diana Moise, et al. "Optimizing intermediate data management in MapReduce computations." Proceedings of the first international workshop on cloud computing platforms. ACM,

2011.
[17] Edward Mazur, et al. "Towards scalable one-pass analytics using mapreduce." Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. IEEE, 2011.
[18] Seok-Hoon Kang, et al. "A case for flash memory ssd in hadoop applications." International Journal of Control and Automation 6.1 (2013): 201-210.
[19] Sangwhan Moon, Jaehwan Lee, and Yang Suk Kee. "Introducing SSDs to the Hadoop MapReduce Framework." Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on. IEEE, 2014.
[20] Karthik Kambatla, and Yanpei Chen. "The truth about MapReduce performance on SSDs." Proceedings of the 28th USENIX conference on Large Installation System Administration. USENIX Association, 2014.
[21] Sungyong Ahn, et al. "Performance Implications of SSDs in Virtualized Hadoop Clusters." Big Data (BigData Congress), 2014 IEEE International Congress on. IEEE, 2014.
[22] Sangwhan Moon, et al. "Optimizing the Hadoop MapReduce Framework with high-performance storage devices." The Journal of Supercomputing (2015): 1-24.

Abstract

Improving Job Processing Speed through Shuffle Phase Optimization for SSD-based Hadoop MapReduce System

Gwangok Go
Graduate School of Convergence Science and Technology
Seoul National University

MapReduce is a programming model widely used for processing large data in cloud datacenter. It is composed of Map, Shuffle and Reduce phases. Hadoop MapReduce is one of the most popular framework implementing MapReduce.

During Shuffle phase, Hadoop MapReduce performs an excessive number of disk I/O operations and the transmission of large data. This accounts for about 40% of total data processing time.

In order to solve these problems, we propose a new shuffle mechanism using the characteristics of SSD. This mechanism consists of (1) data address based sorting, (2) data address based merging and (3) early data transmission before Map

phase completion.

To demonstrate the effectiveness of our approach, we have implemented this mechanism into Hadoop MapReduce 1.2.1. Our experiments show that the proposed mechanism reduces the average job execution time up to 8% compared to that of the legacy Hadoop MapReduce.

Keywords: Hadoop; MapRedue; SSD; Shuffle Mechanism; Distributed Computing

Student Number: 2013-23758