creative commons

COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

조선대학교
CHOSUN UNIVERSITY

[UCI]I804:24011-200000265550

2016年 2月

석사학위논문

하둡기반 대량의 스몰파일 처리성능 향상을 위한 효율적인 데이터 처리방법

조 선 대 학 교 대 학 원

소프트웨어융합공학과

최 철 웅

조선대학교
CHOSUN UNIVERSITY

2016년 2월

석사학위논문

하둡기반 대량의 스몰파일 처리성능 향상을 위한 효율적인 데이터 처리방법

최 철 웅

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

하둡기반 대량의 스몰파일 처리성능 향상을 위한 효율적인 데이터 처리방법

An Effective Data Process Method for Improving the Processing Performance of Massively Small Files on Hadoop Based

2016年 2月25日

조 선 대 학 교 대 학 원

소프트웨어융합공학과

최 철 웅

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

하둡기반 대량의 스몰파일 처리성능 향상을 위한 효율적인 데이터 처리방법

지도교수 김 판 구

이 논문을 공학석사학위신청 논문으로 제출함.

2015年 10月

조 선 대 학 교 대 학 원

소프트웨어융합공학과

최 철 웅

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

최철웅의 석사학위 논문을 인준함

위원장 조선대학교 교수 변 재 영 印

위 원 조선대학교 교수 최 준 호 印

위 원 조선대학교 교수 김 판 구 印

2015 년 12 월

조 선 대 학 교 대 학 원

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

목 차

ABSTRACT

Ⅰ. 서론 ····· 1
A. 연구 배경 및 목적 ····· 1
B. 연구 내용 및 구성 ····· 2

Ⅱ. 관련 연구 ····· 3
A. 하둡(Hadoop) ····· 3
1. HDFS(Hadoop Distributed File System) ····· 4
a. 네임노드(NameNode) ····· 5
b. 보조 네임노드(SecondaryNameNode) ····· 6
c. 데이터노드(DataNode) ····· 7
2. MapReduce ····· 9
a. 잡트래커(JobTracker) ····· 11
b. 태스크트래커(TaskTracker) ····· 11
3. JVM 재사용 ····· 12
4. CombineFileInputFormat ····· 14
B. 하둡에서 스몰파일이 갖는 문제점 ····· 15
1. 하둡 스몰파일 ····· 15
2. 네임노드 메모리 부족 문제 ····· 15
3. 맵리듀스 스케줄링 및 처리시간 증가 문제 ····· 16
C. 하둡 스몰파일문제 관련 기존 연구 ····· 17
1. 파일병합 방법 ····· 17
a. 하둡 아카이브(Hadoop Archive) ····· 17
2. 분산 캐시 방법 ····· 18
3. 기타 방법들 ····· 18

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

Ⅲ. 대량의 스몰파일 처리성능 향상 방법 ····· 20
A. 시스템 구성도 ····· 20
B. 대량의 스몰파일 처리성능 향상 방법 ····· 22
1. CombineFileInputFormat 적용 방법 ····· 22
2. JVM 재사용 방법 ····· 24
3. 제안한 방법 처리과정 ····· 25

Ⅳ. 실험 및 분석 ····· 27
A. 실험환경 ····· 27
B. 실험데이터세트 ····· 30
C. 실험방법 및 결과분석 ····· 32
1. 실험방법 ····· 32
2. 실험결과분석 ····· 33
a. CombineFileInputFormat 적용 방법 ····· 33
b. JVM 재사용 방법 ····· 37
c. 기존 MapReduce 처리 ····· 40
d. 제안한 방법 ····· 42
3. 처리성능 비교 및 평가 ····· 47

Ⅴ. 결론 ····· 50

참고문헌 ····· 51

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

표 목 차

[표 2-1] 태스크 수 조절 파라미터 ····· 12
[표 2-2] JVM 재사용 파라미터(mapred-site.xml 환경설정파일) ····· 12
[표 2-3] mapred.job.reuse.jvm.num.tasks 파라미터 설정값 ····· 13
[표 4-1] 하둡 의사분산모드 실험환경(서버 1대) ····· 27
[표 4-2] 하둡 완전분산모드 실험환경(서버 6대) ····· 27
[표 4-3] Linux Split 명령어 ····· 30
[표 4-4] 의사분산모드 실험 데이터 셋(5GB) ····· 31
[표 4-5] 완전분산모드 실험 데이터 셋(11GB) ····· 31
[표 4-6] 실험방법 ····· 32
[표 4-7] MapReduce 프로그램 ····· 33
[표 4-8] JVM 재사용 설정값에 따른 처리결과(의사분산모드) ····· 39
[표 4-9] JVM 재사용 설정값에 따른 처리결과(완전분산모드) ····· 39
[표 4-10] MapReduce 기존 처리시간(의사분산모드) ····· 40
[표 4-11] MapReduce 기존 처리시간(완전분산모드) ····· 40
[표 4-12] 4MB 스몰파일 처리성능 향상도 ····· 47
[표 4-13] 5MB 스몰파일 처리성능 향상도 ····· 47
[표 4-14] 6MB 스몰파일 처리성능 향상도 ····· 48
[표 4-15] 혼합 스몰파일 처리성능 향상도 ····· 48
[표 4-16] 혼합파일(빅7:스몰3) 처리성능 향상도 ····· 48
[표 4-17] 혼합파일(빅3:스몰7) 처리성능 향상도 ····· 48
[표 4-18] 평균 처리성능 향상도 ····· 49

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

그 림 목 차

[그림 2-1] Hadoop Architecture ····· 3
[그림 2-2] HDFS Architecture ····· 4
[그림 2-3] 네임노드의 메타데이터 관리 ····· 5
[그림 2-4] 네임노드와 보조 네임노드의 체크포인트 ····· 6
[그림 2-5] HDFS 블록 파일 저장 ····· 7
[그림 2-6] Replication Factor(Default Set: 3) ····· 8
[그림 2-7] MapReduce Architecture ····· 9
[그림 2-8] MapReduce 처리과정 ····· 10
[그림 2-9] MapReduce 'WordCount' 프로그램 처리과정 ····· 11
[그림 2-10] JVM 재사용 방법(설정값: 3) ····· 13
[그림 2-11] Combine File Input Format 처리과정(64MB) ····· 14
[그림 2-12] 하둡 스몰파일 ····· 15
[그림 2-13] 하둡 스케줄링 및 처리시간 증가 문제 ····· 16
[그림 2-14] 파일병합 방법 ····· 17
[그림 2-15] Hadoop Archive Layout ····· 18
[그림 3-1] 전체 시스템 구성도 ····· 20
[그림 3-2] CombineFileInputFormat을 사용하기 위한 Combine 클래스 ····· 22
[그림 3-3] JVM을 재사용하기 위한 환경설정 파일(mapred-site.xml) ····· 24
[그림 3-4] 제안방법 처리과정(CombineFileInputFormat 64MB, JVM 재사용 3) 25
[그림 4-1] 하둡 의사분산모드 구성도 ····· 28
[그림 4-2] 하둡 완전분산모드 구성도 ····· 28
[그림 4-3] 미국 규격협회에서 제공하는 미국 항공편 운항 통계 데이터 ····· 30
[그림 4-4] Combine 설정값에 따른 처리결과(의사분산모드) ····· 34
[그림 4-5] Combine 설정값에 따른 처리결과(완전분산모드) ····· 34
[그림 4-6] CombineFileInputFormat 설정값에 따른 처리시간(의사분산모드) ··35
[그림 4-7] CombineFileInputFormat 설정값에 따른 처리시간(완전분산모드) ··36
[그림 4-8] JVM 재사용 설정값에 따른 처리결과(의사분산모드) ····· 37
[그림 4-9] JVM 재사용 설정값에 따른 처리결과(완전분산모드) ····· 38

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[그림 4-10] MapReduce 기존 처리시간(완전분산모드) ·····41
[그림 4-11] 제안한방법 처리시간 비교(완전분산모드) ·····42
[그림 4-12] 제안방법 처리시간(데이터세트: 4MB) ·····43
[그림 4-13] 제안방법 처리시간(데이터세트: 5MB) ·····43
[그림 4-14] 제안방법 처리시간(데이터세트: 6MB) ·····44
[그림 4-15] 제안방법 처리시간(혼합 스몰파일 4,5,6MB) ·····44
[그림 4-16] 제안방법 처리시간(빅파일7:스몰파일3) ·····45
[그림 4-17] 제안방법 처리시간(빅파일3:스몰파일7) ·····45

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

ABSTRACT

An Effective Data Process Method for Improving the Processing Performance of Massively Small Files on Hadoop Based

Chulwoong Choi
Advisor : Prof. PanKoo Kim, Ph.D
Department of SoftWare Convergence Engineering
Graduate School of Chosun University

With the proliferation of smartphones and expansion of social networking services (SNSs), various types of data are now being generated and the big data era has arrived. Due to various types and sizes of data, and the short period of big data generation, existing systems are unable to process big data. Thus, an appropriate new system to process big data is required. Hadoop is a representative technology that stores and processes big data. It is a distributed parallel processing system consisting of the Hadoop Distributed File System (HDFS) that stores big data and MapReduce that processes and analyzes big data. Hadoop is optimized to store data in files with relatively large size and thus suffers from disadvantages of memory insufficiency of name nodes and increased scheduling and processing time for a large number of small files.

This study proposes a performance improvement method for MapReduce processing in which the CombineFileInputFormat method and the reuse method of the Java Virtual Machine (JVM) are employed together. Existing methods

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

create a mapper for every small file. Unlike these methods, the proposed method reduces the number of created mappers by processing large number of files that are combined by a single split using CombineFileInputFormat. It also improves the MapReduce processing performance by reducing the creation time of JVM by reusing a single JVM for a large number of mappers rather than creating a JVM for every mapper.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

Ⅰ. 서론

A. 연구 배경 및 목적

스토리지 전문기업인 EMC에서 발표한 '디지털 유니버스 보고서'의 연구결과를 보면 2012년 전 세계에서 생성된 디지털 데이터의 양은 2.8제타바이트(Zetabytes)에 이르며, 2020년에는 40제타바이트까지 성장할 것으로 예측했다. 40제타바이트는 2005년 디지털 데이터양의 약 300배에 해당하며, 전 세계 해변 모래알 수의 약 57배에 해당하는 숫자이다[1]. 이처럼 우리는 빅데이터가 폭증하는 시대에 살고 있다. 하둡은 빅데이터를 저장하고 처리하는 대표적인 분산처리 기술로써 크게 빅데이터를 저장하는 HDFS(Hadoop Distributed File System)와 빅데이터를 처리하는 MapReduce로 구성된다. 하둡은 기본적으로 파일의 크기가 큰 빅데이터에 최적화되어있어, 대규모 스몰파일과 같이 다수의 크기가 작은 스몰파일에는 취약한 단점을 가지고 있다. 하둡은 데이터를 저장할 때 일정한 크기의 블록(기본 64MB)으로 나누어 저장하는데 하둡에서 의미하는 스몰파일이란 기본 블록보다 크기가 작은 파일을 의미한다. 하둡에서 스몰파일로 인해 발생하는 문제점은 크게 2가지가 있다. 첫 번째 HDFS에 파일들이 블록으로 나누어져 저장될 때 파일들의 메타데이터를 네임노드에 인메모리 방식으로 저장하는데, 대량의 스몰파일을 저장하게 되면 메타데이터를 저장하는 네임노드의 메모리가 부족하여 데이터노드에 파일을 저장할 수 있는 여유 공간이 있더라도 저장하지 못하는 문제가 있다. 두 번째 MapReduce는 HDFS에 저장된 블록 하나당 각각 매퍼(Mapper)가 생성되어 처리하게 되는데 대량의 스몰파일과 같은, 예를 들어 1,000개의 파일이라면 1,000번의 매퍼가 생성되어 스케줄링 및 처리성능이 저하된다[2]. 본 논문에서는 하둡에서 제공하는 JVM 재사용 방법과 CombineFileInputFormat 방법을 함께 사용하여 하둡에서 대량의 스몰파일 처리 시 스케줄링 및 처리성능을 향상하는 방법을 제안하였다. 기존 방법들과는 달리 전처리 과정이 없고 하둡시스템 자원을 사용하지 않기 때문에 간결하고 쉬운 방법으로 하둡에서 대용량 스몰파일의 처리성능을 향상할 수 있다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

B. 연구 내용 및 구성

본 연구의 주요 내용은 하둡에서 대용량 스몰파일의 처리성능을 향상하는 방법을 제안하기 위해 다음과 같이 작성되었다.

서론에 이어 2장 관련 연구에서는 본 연구의 배경이 되는 하둡과 대용량 스몰파일 문제를 해결하는 기존연구들에 대해 살펴본다.

3장에서는 본 논문에서 제안하는 대용량 스몰파일 처리성능 향상방법에 관해 기술한다. MapReduce 처리과정속에서 JVM 재사용방법과 CombineFileInputFormat 방법을 함께 사용하여 처리성능을 향상하는 방법을 제안한다.

4장에서는 실험에 사용한 데이터 셋과 실험환경에 관해 설명하고, 다양한 실험을 통한 비교측정 및 성능을 평가한다.

마지막으로 5장에서는 결론과 향후 연구에 관해 서술하며 마무리한다.

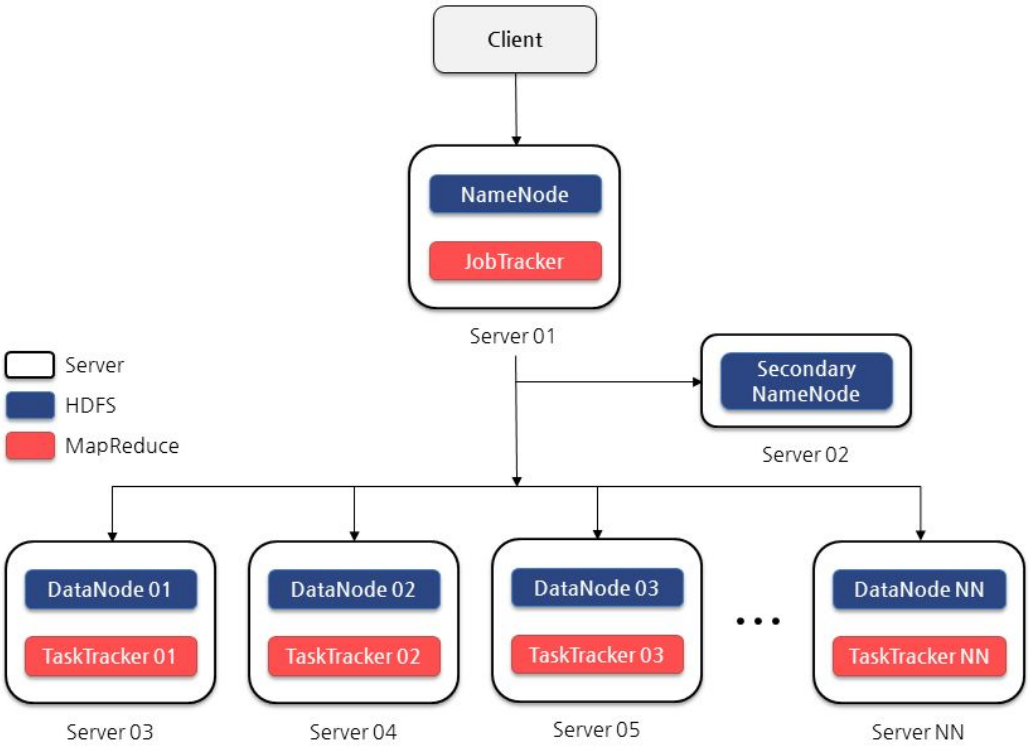
Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

Ⅱ. 관련 연구

A. 하둡(Hadoop)

하둡의 시작은 더그 커팅(Doug Cutting)의 오픈소스 검색엔진 너치(Nutch) 프로젝트의 확장성을 위해 개발되었다. 구글 연구소(Google Lab.)에서 2003년과 2004년에 내놓은 'The Google File system', 'MapReduce: Simplified Data Processing on Large Cluster'라는 2편의 논문을 바탕으로 아이디어를 얻어 만들기 시작하였다. 너치 프로젝트의 하위프로젝트로 개발되던 중 독립프로젝트로 나왔으며 2008년에는 아파치 최상위 프로젝트로 승격되었다. 하둡은 아파치 재단에서 관리하는 오픈소스로써 Apache License 출처만 적는다면 누구나 무료로 사용이 가능하다[3].



[그림 2-1] Hadoop Architecture

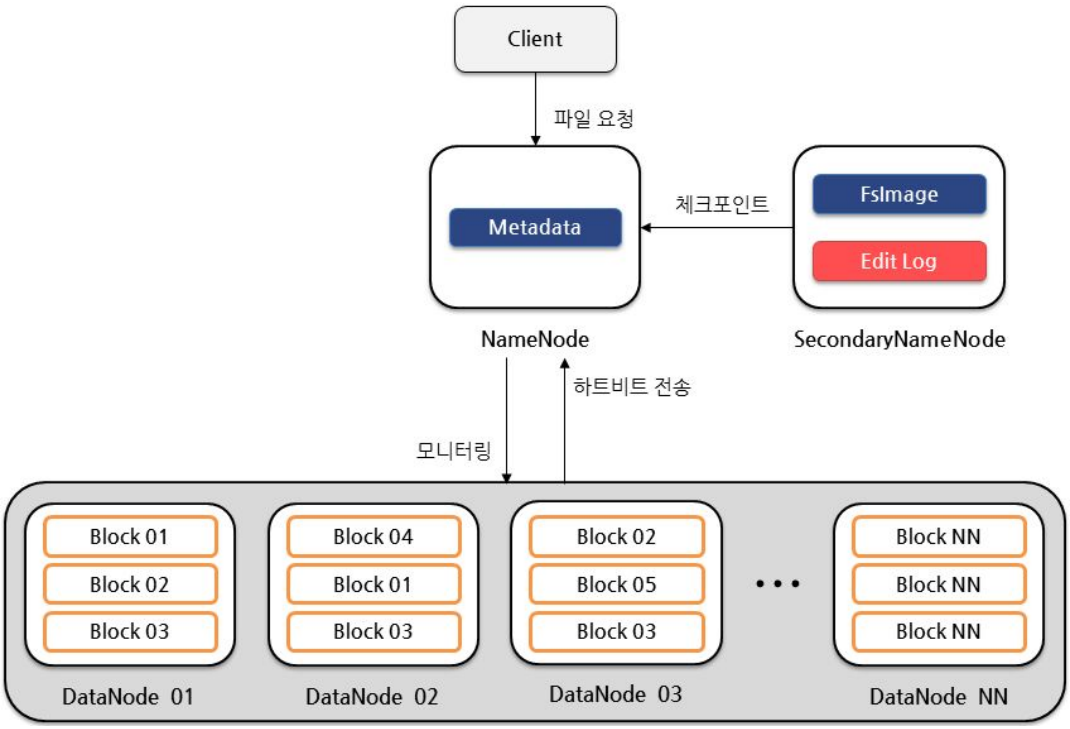
Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

하둡은 스케일 아웃방식의 분산병렬처리 시스템으로 크게 빅데이터를 저장하는 HDFS(Hadoop Distributed File System) 분산파일시스템과 빅데이터를 처리하는 MapReduce 분산처리시스템으로 구성되어 있다[3][17][19]. 마스터(Master)/슬레이브(Slave) 구조로 작동되며 전체적인 아키텍처는 [그림 2-1]과 같다. 마스터 서버는 네임노드(NameNode), 보조 네임노드(Secondary NameNode) 및 잡트래커(JobTracker)로 구성되며, 하둡 아키텍처에 각 1대만 실행된다[16]. 슬레이브 서버는 데이터노드(DataNode)와 태스크트래커(TaskTracker)로 구성되며, 서버를 계속 추가하여 사용할 수 있다[4].

1. HDFS(Hadoop Distributed File System)

HDFS는 하둡의 창시자 더그 커팅이 구글의 논문 'The Google File System'을 바탕으로 만든 분산 파일 시스템이다. 네임노드와 보조 네임노드 그리고 데이터노드로 구성되어 있으며 전체적인 아키텍처는 [그림 2-2]와 같다[5].



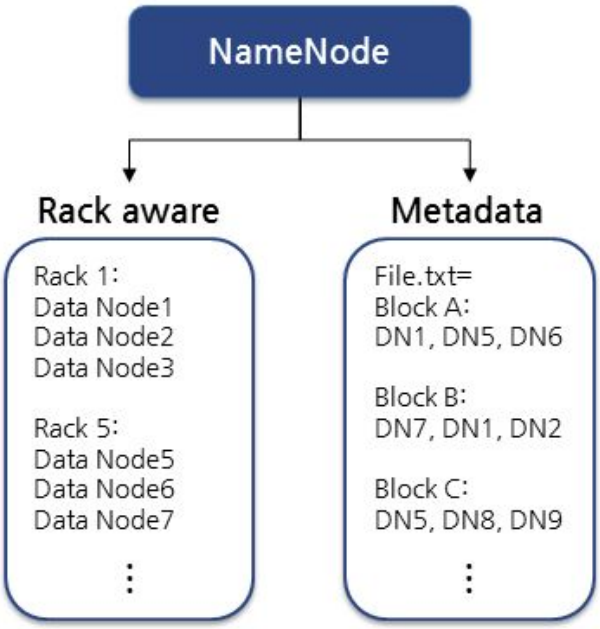
[그림 2-2] HDFS Architecture

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

a. 네임노드(NameNode)

마스터 서버인 네임노드는 실제 데이터가 저장되는 데이터노드의 메타데이터를 [그림 2-3]과 같이 관리하며 모든 사용자는 저장된 파일을 요청할 때 네임노드를 거쳐야만 한다. Rack aware에는 Rack 정보와 서버의 위치가 담겨있고, Metadata는 실제 HDFS에 저장되는 파일들의 블록정보와 블록이 저장된 데이터노드의 위치정보를 담고 있다[3]. 메타데이터는 네임노드에 인 메모리(In-memory)형식으로 저장되며, 메타데이터 하나당 약 150Byte 크기의 메모리가 필요하다[4].



[그림 2-3] 네임노드의 메타데이터 관리

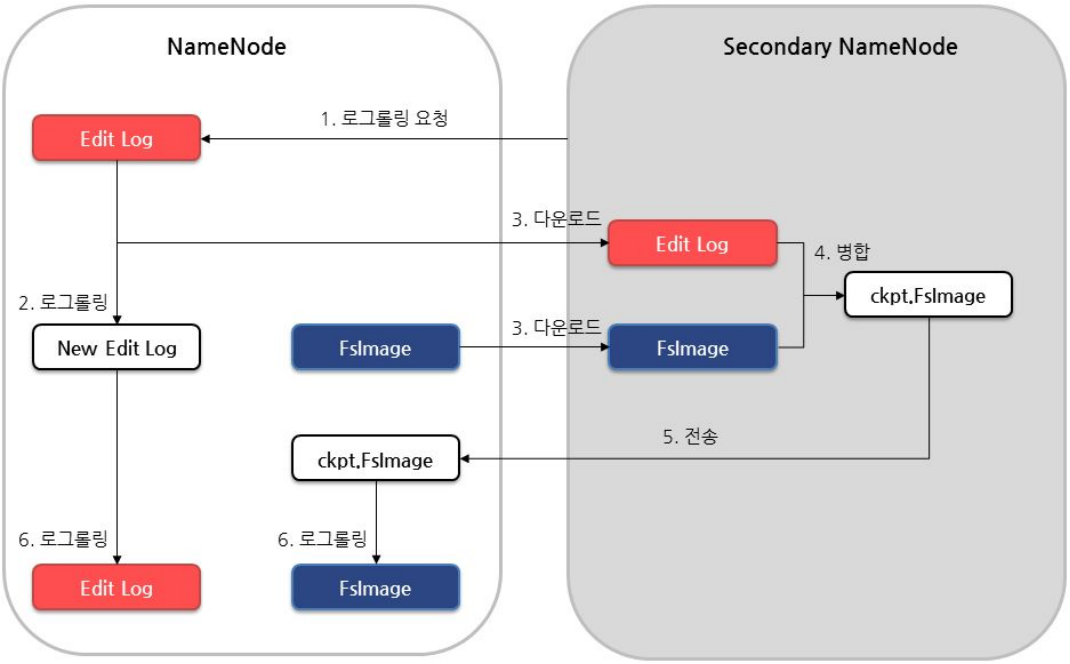
네임노드는 슬레이브 서버인 데이터노드들을 모니터링하며 감독한다. 모든 데이터노드들은 3초에 한 번씩 네임노드로 데이터노드의 현재 상태와 블록목록을 담고 있는 하트비트(Heartbeat) 메시지를 전송한다. 데이터노드로부터 하트비트가 일정 시간 동안 전송되지 않으면 장애가 발생한 것으로 판단하고 장애가 발생한 데이터노드에 데이터를 모두 다른 데이터노드로 복제한다. 또한, 복제본 수(기본: 3)가 같지 않은 블록이 발견되면 추가로 블록을 복제하거나 삭제한다[5].

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

b. 보조 네임노드(SecondaryNameNode)

네임노드는 파일들의 메타데이터를 인 메모리 방식으로 저장하는데, 네임노드를 재부팅 하거나, 오류가 발생하게 되면 메타데이터를 저장하고 있는 메모리가 손실되어 HDFS에 심각한 장애가 생길 수 있다. 보조 네임노드는 위와 같은 상황을 대비하여 에디트 로그(Edit Log)와 파일시스템이미지(FsImage) 2가지를 사용하여 네임노드에게 1시간마다 주기적으로 체크포인트(CheckPoint)를 요청한다. 에디트 로그는 용량 제한 없이 HDFS의 메타데이터의 모든 변화를 기록하는 로그 파일이고, 파일시스템이미지는 네임스페이스와 블록 매핑 정보 등 네임노드의 현재 상태를 저장한 파일이다. 에디트 로그와 파일시스템이미지 모두 네임노드의 로컬 파일시스템에 파일로 저장한다[3][5].



[그림 2-4] 네임노드와 보조 네임노드의 체크포인트

네임노드와 보조 네임노드의 체크포인트 단계는 [그림 2-4]와 같이 6단계로 진행된다[5]. 첫 번째 단계는 보조 네임노드가 네임노드에게 로그롤링을 요청하는 단계이다. 로그롤링이란 현재 로그 파일의 이름을 수정하고, 기존 이름으로 새 로그파일을 만드는 과정을 의미한다. 두 번째 단계는 로그롤링을 요청받은 네임노드

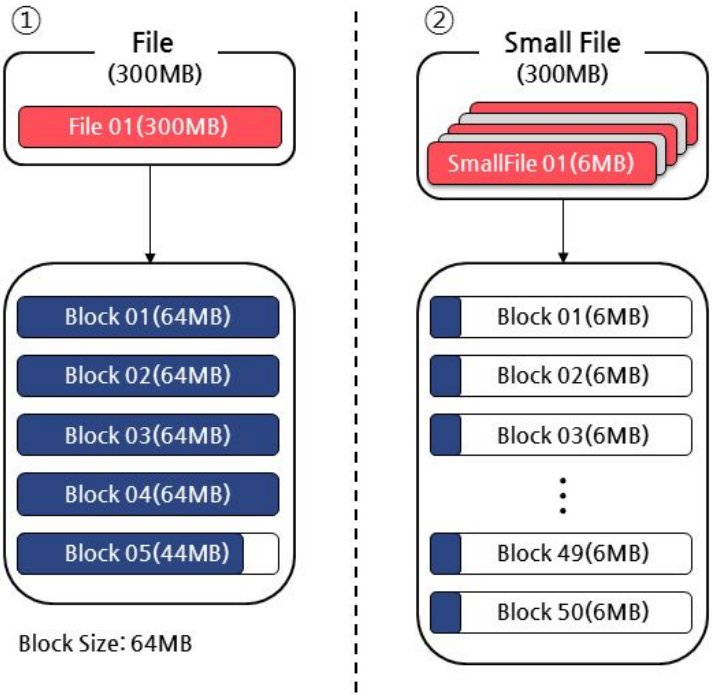
Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

가 새로운 에디트 로그를 생성하여 현시점부터 HDFS의 모든 수정 기록들을 저장하게 한다. 세 번째 단계는 보조 네임노드가 네임노드에 있는 기존 에디트 로그와 파일시스템이미지를 다운받는다. 4번째 단계는 보조 네임노드가 다운받은 파일 시스템이미지에 그동안 HDFS의 모든 수정기록이 담긴 에디트 로그를 더하여 가장 최신의 파일시스템이미지인 체크포인트 파일시스템이미지를 생성한다. 다섯 번째 단계에서는 생성된 체크포인트 파일시스템이미지를 네임노드로 전송하고, 6번째 단계에서는 새로운 에디트 로그와 보조 네임노드로부터 받은 체크포인트 파일시스템이미지를 네임노드가 각각 에디트 로그와 파일시스템이미지로 로그롤링 한다[3][5].

c. 데이터노드(DataNode)

파일들이 실제로 저장되는 데이터노드는 파일을 저장할 때 일정한 크기의 블록으로 나누어 저장한다. 기본 블록 크기는 64MB이지만 실제 환경에서는 주로 128MB를 많이 사용한다[4].



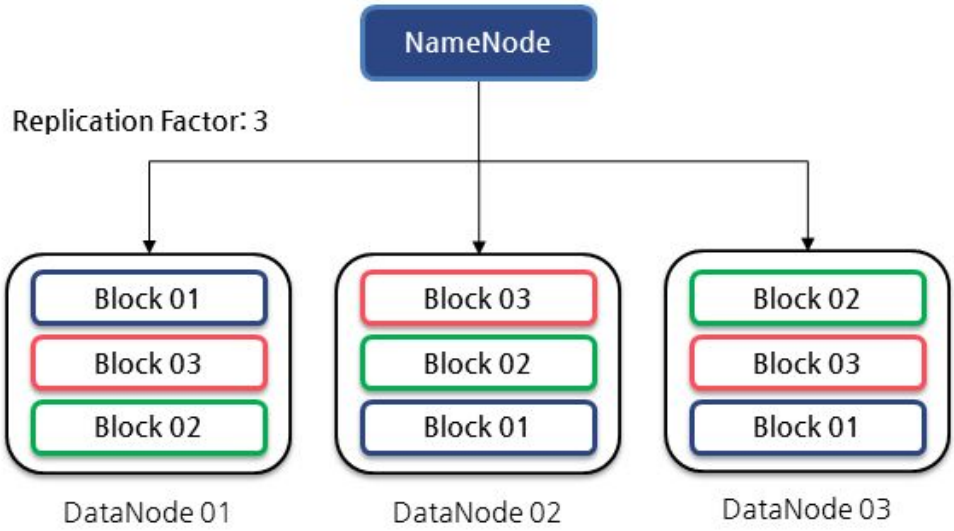
[그림 2-5] HDFS 블록 파일 저장

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

블록 크기가 다른 파일 시스템에 비해 큰 이유는 크게 2가지가 있다. 첫 번째는 네임노드의 메타데이터 크기를 감소하기 위함이다. 블록 크기가 클수록 네임노드가 관리하는 메타데이터 수가 감소하여 네임노드의 부하를 줄일 수 있다. 두 번째는 디스크 탐색시간을 줄이는 데 목적이 있다. 하둡은 블록 크기를 크게 하여 데이터의 위치를 찾는 시크 타임(seek time)을 감소시켜 탐색시간을 줄인다[5].

[그림 2-5]는 데이터노드에 파일 저장 시 기본파일과 대량의 스몰파일이 블록으로 나누어져 저장되는 과정이다. 1번은 기존파일을 저장할 때 블록으로 나누어져 저장되는 방법으로 총 300MB의 크기의 파일이 기본 블록 크기인 64MB로 나누어져 총 5개의 블록에 저장되었고 마지막 블록에는 44MB만 저장되었다. 2번은 기본 블록 크기보다 작은 6MB의 스몰파일들이 각각 64MB의 블록에 저장되어 총 50개의 블록이 생성되었다. 똑같은 크기의 300MB 파일이지만 스몰파일 경우 기존 파일보다 10배의 블록이 더 생성되었으며, 이는 네임노드의 메타데이터 수 증가로 이어져 네임노드의 메모리가 부족해지는 문제점이 발생한다[3][4][5].



[그림 2-6] Replication Factor(Default Set: 3)

데이터노드에 블록으로 나누어져 저장된 파일들은 Replication Factor에 의해 2개의 복제본을 생성하여 같은 파일 3개가 HDFS에 저장된다. [그림 2-6]은 Replication Factor 기본값 3에 의해 3개의 같은 파일이 데이터노드에 저장되는

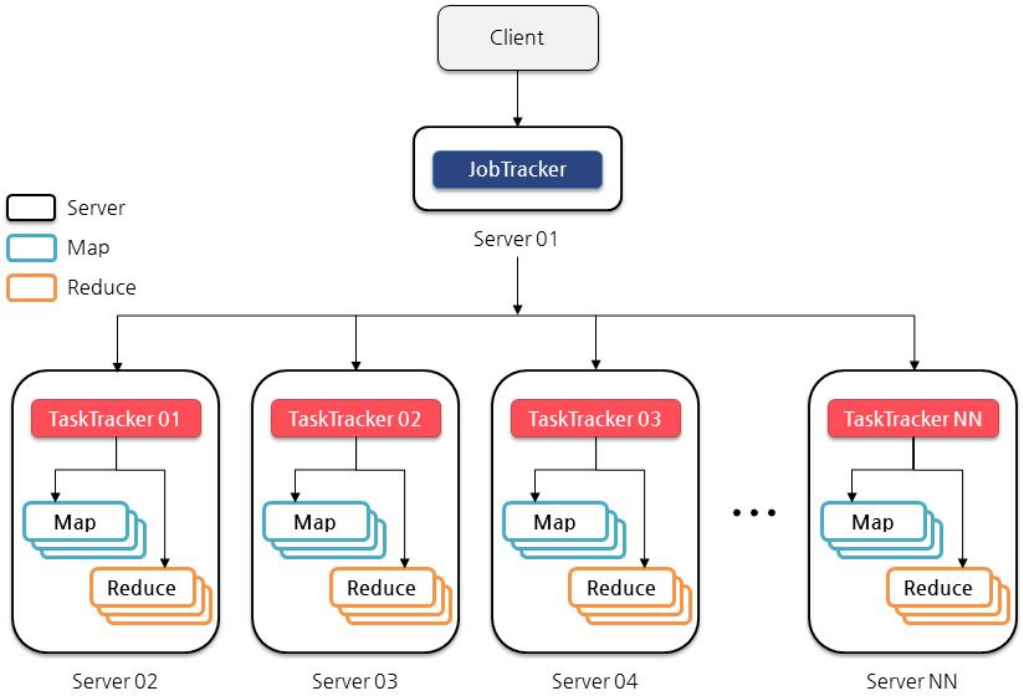
Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

그림이다[3]. 같은 파일을 3개나 저장하는 이유는 특정 서버나 Rack에 오류가 생겼더라도 쉽게 복구하기 위함이다. 하둡은 스케일 아웃방식을 사용하기 때문에 고가의 서버보다는 중저가의 서버를 여러 대 사용하여 고가의 장비보다는 오류에 취약하다. 복제본이 저장된 서버 3개가 동시에 장애가 날 확률은 극히 드물어서 하둡에서는 복제본을 만들어 각기 다른 서버에 저장함으로써 파일을 안전하게 보관한다. 또한, 네임노드에서 데이터노드를 모니터링 함으로써 중간에 장애가 발생하여 Replication Factor가 3이 되지 않는 파일들은 즉시 다른 데이터노드로 복제하여 항상 복제본 수를 3으로 유지한다[3][4][5].

2. MapReduce(맵리듀스)

MapReduce는 더그 커팅이 구글의 논문 'MapReduce: Simplified Data Pro-cessing on Large Cluster'를 바탕으로 만든 분산 처리 시스템이다. 잡트래커와 태스크트래커로 구성되어 있으며 전체적인 아키텍처는 [그림 2-7]과 같다[3][5].

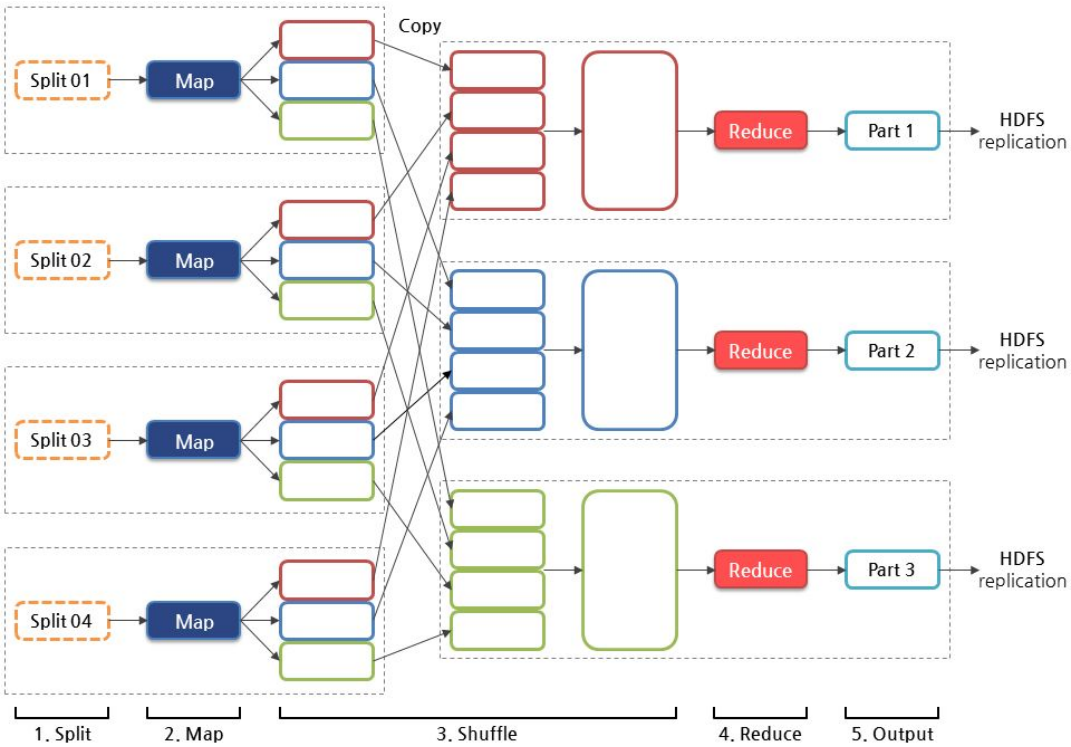


[그림 2-7] MapReduce Architecture

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

맵리듀스는 크게 맵(Map)과 리듀스(Reduce)라는 2개의 메서드로 실행되며 모든 데이터를 키(Key)/밸류(Value) 데이터세트로 변환하여 처리한다[18]. [그림 2-8]은 맵리듀스의 데이터 처리 과정을 나타내고 있다[6].

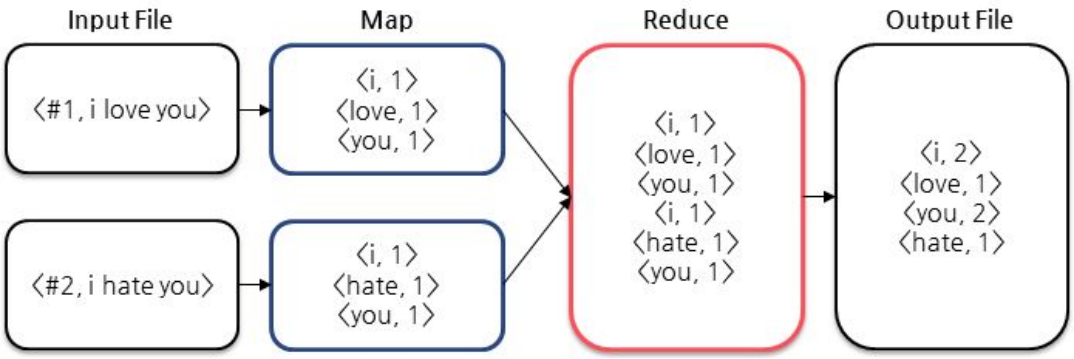


[그림 2-8] MapReduce 처리과정

맵리듀스는 크게 Split, Map, Shuffle, Reduce, Output 5단계로 [그림 2-8]과 같이 데이터를 처리한다. 1단계 잡에 필요한 데이터를 HDFS에서 호출하여 논리적 단위인 Split으로 분리한다. 2단계 Split 된 파일을 JVM 위에 생성되는 Map Task에 넣고 작은 조각으로 나눈다. 3단계 Shuffle은 Map의 출력결과를 정렬하고 병합하여 Reduce에게 보낸다. 4단계 Reduce는 사용자가 작성한 Reduce 프로그램에 따라 원하는 파일만 추출한다. 5단계 Reduce의 출력 데이터를 HDFS에 저장한다[5][8]. 이외에도 파티셔너(Partitioner), 콤바이너(Combiner) 등 다양한 처리 과정이 존재한다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 2-9] MapReduce 'WordCount' 프로그램 처리과정

[그림 2-9]는 하둡을 설치하면 기본적으로 제공하는 글자 수 세기 프로그램인 'WordCount' 프로그램의 처리 과정을 나타내고 있다. MapReduce는 모든 데이터를 Key/Value 형태로 처리하는데 [그림 2-9]의 쉼표(,)를 기준으로 앞이 Key가 되고 뒤가 Value가 된다. Input File이 들어오면 Map 단계에서는 문장의 Space(공백)을 기준으로 나누어 처리 된 Output을 Reduce의 Input으로 제공한다. Reduce는 Map이 나누어 놓은 결과물을 처음부터 순차적으로 Count 하여 Output 결과물을 만들어 HDFS에 저장한다[9].

a. 잡트래커(JobTracker)

잡트래커는 하둡에서 진행되는 전체 잡(Job)의 스케줄링을 관리하고 모니터링한다. 클라이언트가 실행을 요청하는 맵리듀스 프로그램은 잡이라는 하나의 작업 단위로 관리되며, 하둡 클러스터에서 단 한 대만 존재하고 일반적으로 하둡 네임노드 서버에서 실행된다. 사용자가 잡을 요청 하면 잡트래커는 몇 개의 태스크가 필요한지 계산하고, 각 태스크트래커에게 잡을 할당한다. 태스크트래커는 잡트래커에게 요청을 받고 맵리듀스 프로그램을 실행한다[5].

b. 태스크트래커(TaskTracker)

태스크트래커는 하둡 데이터노드에서 구동되며, 다수의 태스크를 생성하여 잡을 처리 한다. 태스크는 사용자가 설정한 맵과 리듀스 프로그램을 의미하며, 맵 태스크와 리듀스 태스크로 구성된다. 기본적으로 태스크는 각각 태스크트래커에 2개씩

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

생성되며, mapred-site.xml 환경설정 파일에서 [표 2-1]의 파라미터를 수정하여 태스크 수를 조절할 수 있다[5][7].

[표 2-1] 태스크 수 조절 파라미터

파라미터	기본 값	내용
mapred.tasktracker.map.tasks.maximum	2	Map 태스크 수
mapred.tasktracker.reduce.tasks.maximum	2	Reduce 태스크 수

3. JVM 재사용

JVM 재사용 방법은 기존에 태스크마다 생성되는 JVM을 하둡 환경설정파일의 파라미터 값을 수정하여, 다수의 태스크가 하나의 JVM을 순차적으로 재사용하는 방법이다. 하둡 환경설정파일(mapred-stie.xml)에서 [표 2-2]의 3번째 줄의 파라미터 mapred.job.reuse.jvm.num.tasks의 값을 사용자임으로 수정하여 사용할 수 있다[9].

[표 2-2] JVM 재사용 파라미터(mapred-site.xml 환경설정파일)

파라미터
<configuration>
<property>
<name>mapred.job.reuse.jvm.num.tasks</name>
<value>3</value> // 설정값
</property>
</configuration>

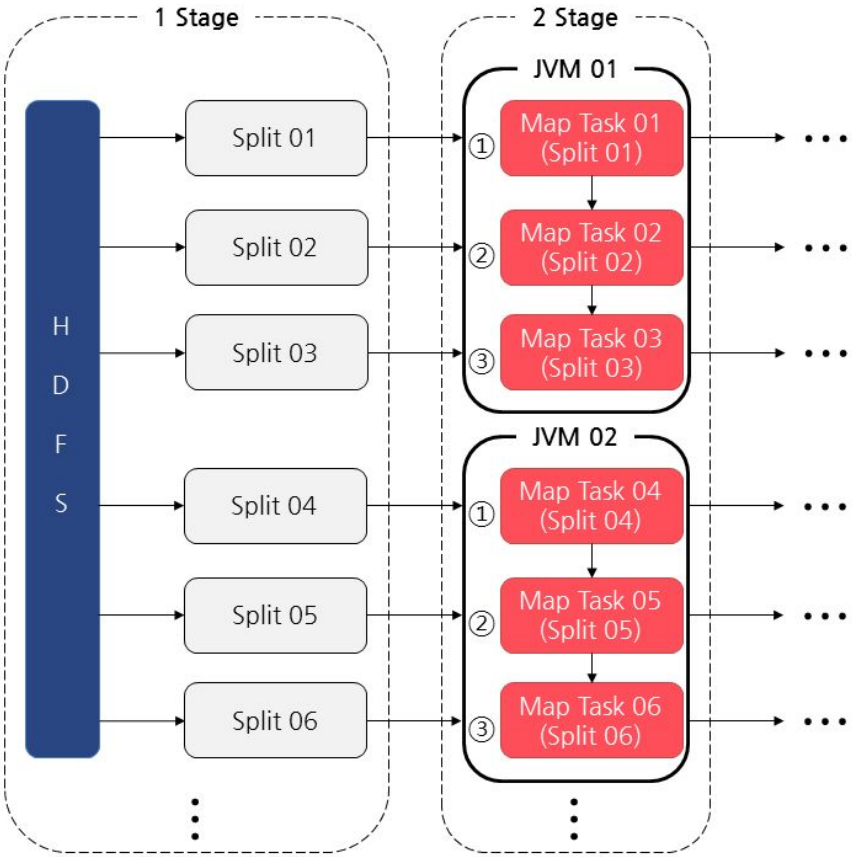
[표 2-3]은 JVM 재사용 방법을 사용하기 위한 3가지 설정값을 보여주고 있다. 설정값의 수는 하나의 JVM 내에서 순차적으로 실행되는 태스크의 수이다. 기본값인 1은 JVM을 재사용하지 않는다는 의미이고, -1은 하나의 JVM만을 사용하여 모든 태스크를 실행한다는 의미이다. 마지막으로 사용자 설정값은 사용자가 임으로 원하는 값을 입력하면 입력한 수만큼의 태스크가 하나의 JVM을 재사용 한다[9].

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[표 2-3] mapred.job.reuse.jvm.num.tasks 파라미터 설정값

설정 값	내 용
1(Default)	JVM을 재사용하지 않음
-1	무제한으로 JVM 재사용
사용자 설정	입력한 수만큼 JVM 재사용



[그림 2-10] JVM 재사용 방법(설정값: 3)

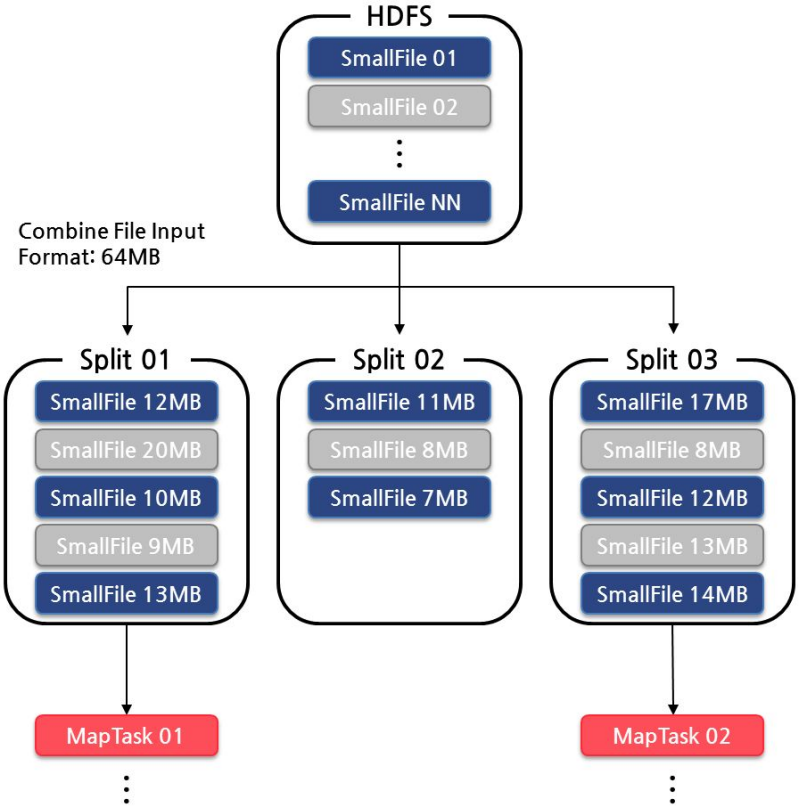
[그림 2-10]은 JVM 재사용 방법의 설정값을 3으로 설정한 후 처리 과정을 나타내고 있다[9]. 기존방식은 Map Task마다 개별적으로 JVM이 생성되어 Map Task 1은 JVM 1에서 Map Task 2는 JVM 2에서 실행되었지만, JVM 재사용 방법을 사용하게 되면 [그림 2-10]과 같이 3개의 Task가 하나의 JVM에서 순차적으로 실행된다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

4. CombineFileInputFormat

CombineFileInputFormat은 기존에는 파일마다 생성되는 Split을 다수의 파일을 하나의 Split으로 묶어 사용하는 방법이다. 각 Node와 Rack의 위치를 고려하여 Split 파일을 결정하며, 블록 크기와 비슷한 64MB, 128MB 등을 사용한다.



[그림 2-11] Combine File Input Format 처리과정(64MB)

[그림 2-11]은 Combine File Input Format 방법의 처리 과정을 보여주고 있다. 설정값이 64MB가 될 때까지 파일을 묶어 처리하게 되며, 우선순위로 64MB로 스플릿된 파일들을 먼저 처리하게 된다. 스플릿으로 묶어 처리하게 되므로 매퍼 수를 줄여 스케줄링 및 처리성능을 향상할 수 있다. CombineFileInputFormat 방법은 추상클래스이기 때문에 사용하기 위해서는 Combine 클래스를 생성하고 RecordReader() 메소드를 구현해야 한다[4].

Collection @ chosun

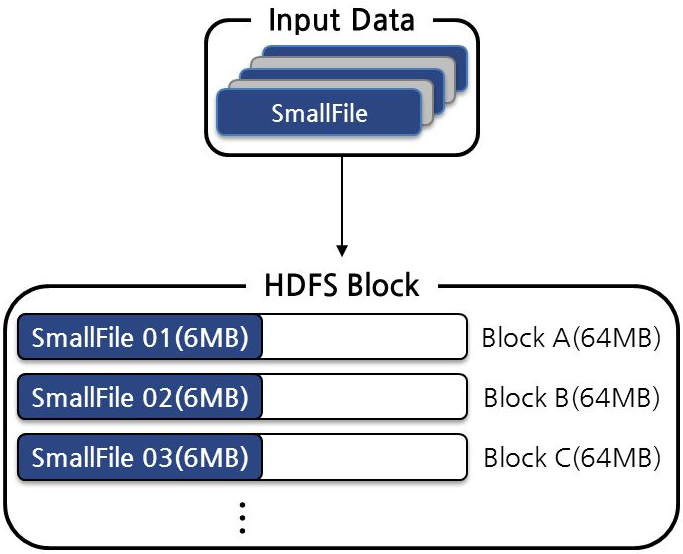
조선대학교
CHOSUN UNIVERSITY

B. 하둡에서 스몰파일이 갖는 문제점

하둡에서 대량의 스몰파일이 갖는 문제점은 크게 네임노드의 메모리가 부족한 문제점과 MapReduce의 스케줄링 및 처리시간이 증가하는 문제점이 있다.

1. 하둡 스몰파일

하둡은 HDFS에 파일을 저장할 때 일정한 크기의 블록으로 나누어 파일을 저장한다. 기본 블록사이즈는 64MB로, [그림 2-12]와 같이 블록사이즈보다 작은 파일을 본 논문에서는 스몰파일이라고 한다. 대량의 스몰파일이란 블록사이즈보다 작은 크기의 파일이 무수히 많이 존재하는 것을 의미한다.



[그림 2-12] 하둡 스몰파일

2. 네임노드 메모리 부족 문제

하둡 HDFS에 파일을 저장하게 되면 실질적인 데이터는 데이터노드에 저장되고, 데이터노드에 저장 된 파일의 위치나 블록 복제본 위치 등과 같은 메타데이터는 [그림 2-3]과 같이 네임노드에 인 메모리(In-Memory) 방식으로 저장된다. HDFS에 대량의 스몰파일을 저장하게 되면, 저장되는 파일 개체 수의 증가로 인한 네임노

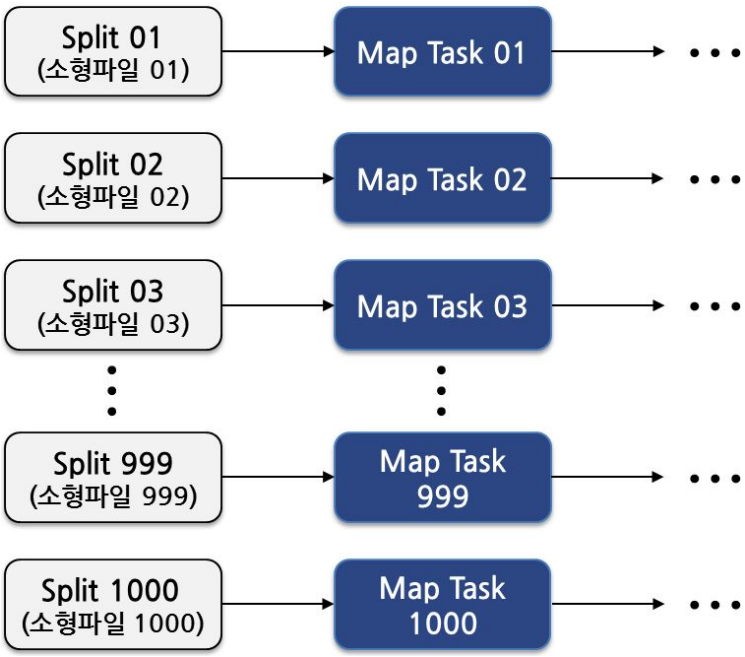
Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

드 메모리가 부족한 문제점이 발생한다[21]. 데이터노드에 데이터를 적재할 여유 공간이 있더라도 메타데이터를 저장하는 네임노드의 메모리가 부족하다면 더는 데이터를 저장할 수 없다. 이와 같은 문제점을 해결하기 위한 기존 연구로는 하둡 아카이브나 Sequence 방법과 같은 파일 병합방법이 있다.

3. 맵리듀스 스케줄링 및 처리시간 증가 문제

하둡 MapReduce에서 데이터를 처리할 때 [그림 2-13]과 같이 기본적으로 블록 하나당 Map Task가 1개씩 생성되어 처리된다. 대량의 스몰파일과 같이 다수의 파일을 MapReduce로 처리하게 되면 스몰파일 개수와 동일한 Map Task가 생성되어 스케줄링 및 처리시간이 증가하는 문제점이 발생한다. 이와 같은 문제점을 해결하기 위한 기존 연구로는 분산 캐시 방법, Spill파일 병합 개수 조절 등이 있다.



[그림 2-13] 하둡 스케줄링 및 처리시간 증가 문제

Collection @ chosun

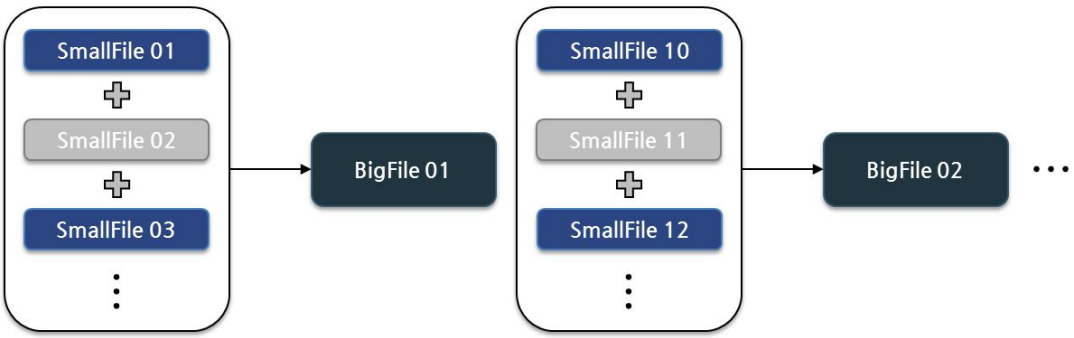
조선대학교
CHOSUN UNIVERSITY

C. 하둡 스몰파일문제 관련 기존연구

하둡의 스몰파일 문제와 관련된 기존 연구로는 크게 파일병합 방법과 캐시 메모리 방법이 있다. 그 외에도 파일 압축, 버퍼크기 조절, 기타 등등 다양한 연구가 진행되고 있다.

1. 파일병합 방법

파일병합방법은 기존과는 다른 파일포맷 방법을 사용하여 다수의 스몰파일을 하나의 빅파일로 만드는 방법으로 하둡 아카이브(Hadoop Archive), Sequence File 등이 있다. [그림 2-14]는 파일병합 방법의 처리 과정을 보여주고 있다.



[그림 2-14] 파일병합 방법

[그림 2-14]와 같이 다수의 스몰파일을 빅파일로 병합하게 되면 스몰파일 수가 감소하고 네임노드에 저장되는 메타데이터 수 또한 감소하게 되어 실제 데이터가 저장되는 데이터노드에 여유 공간이 있더라도 네임노드의 메모리가 부족하여 더는 데이터를 저장할 수 없었던 문제를 해결할 수 있다. 하지만 다수의 스몰파일을 빅파일로 병합하는데 많은 소비시간이 소요되어, 전체적으로 봤을 때 기본 방식보다 처리시간이 증가하는 단점이 있다[10].

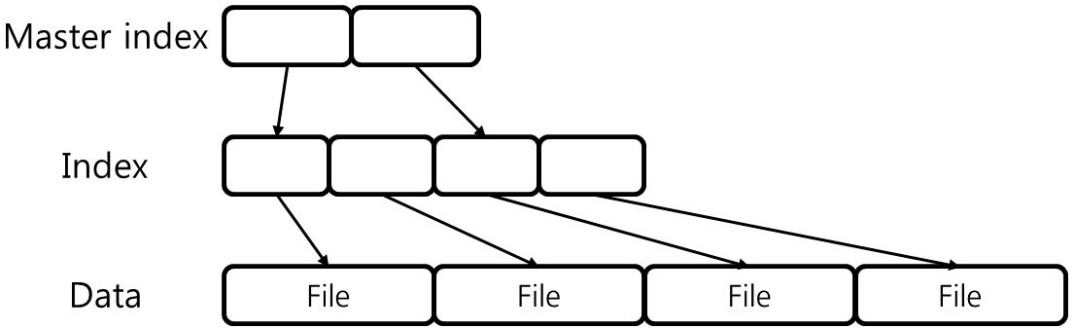
a. 하둡 아카이브(HAR: Hadoop Archive)

하둡 아카이브는 파일포맷방식 중의 하나로 Layout은 [그림 2-15]와 같다[14].

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

다수의 스몰파일을 MapReduce 처리 과정을 거쳐 빅파일로 병합하고 HAR 파일을 생성한다. 하둡 아카이브는 파일을 읽을 때 [그림 2-15]와 같이 Master index와 index 2번의 과정을 통하여 수행됨으로 기존의 HDFS 보다 파일을 읽는 시간이 증가한다[14][20].



[그림 2-15] Hadoop Archive Layout

2. 분산 캐시 방법

분산 캐시 방법은 클라이언트와 데이터노드의 캐시를 사용하는 방법으로 각 데이터노드들의 캐시에 클라이언트가 자주 사용하는 스몰파일을 유지하는 방법이다. 이로써 데이터를 처리할 때 데이터노드의 물리적인 디스크 접근 횟수를 감소시켜 처리성능을 향상한다. 클라이언트 캐시만 사용해오던 [11]의 연구에서 데이터노드의 캐시를 함께 사용하는 [12]의 연구 그리고 최근에는 네임노드에서 추가로 캐시 메타데이터를 관리하는[13]의 연구까지 발전하며 점진적으로 진행되고 있다. 하지만 데이터노드의 분산 캐시 같은 하둡의 자원을 사용한다는 점과 캐시에 없는 스몰파일은 다시 데이터노드 디스크에서 불러와야 하므로 Hit ratio에 의존적이라는 단점이 있다[10][13].

3. 기타 방법들

파일병합 방법과 분산 캐시 방법 외에 하둡 스몰파일 문제를 해결하기 위한 다양한 방법들이 연구되고 있다. append 방법은 대량의 스몰파일을 하나의 파일에 계속 덧붙여 가공한 후 처리하는 방식이다. 스몰파일이 1000개가 있다면 1000번의

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

덧붙이기가 실행되어야 하므로 의미가 없는 방법이다[2]. 데이터 압축방법은 Gzip, bzip2, 스내피, LZO 등 다양한 압축 코덱을 사용하여 스몰파일을 압축하여 처리하는 방법이다[5]. 이외에도 스필파일 병합개수 조절, 매퍼 및 리듀서 버퍼 크기 조절 등 다양한 방법들이 존재한다.

Collection @ chosun

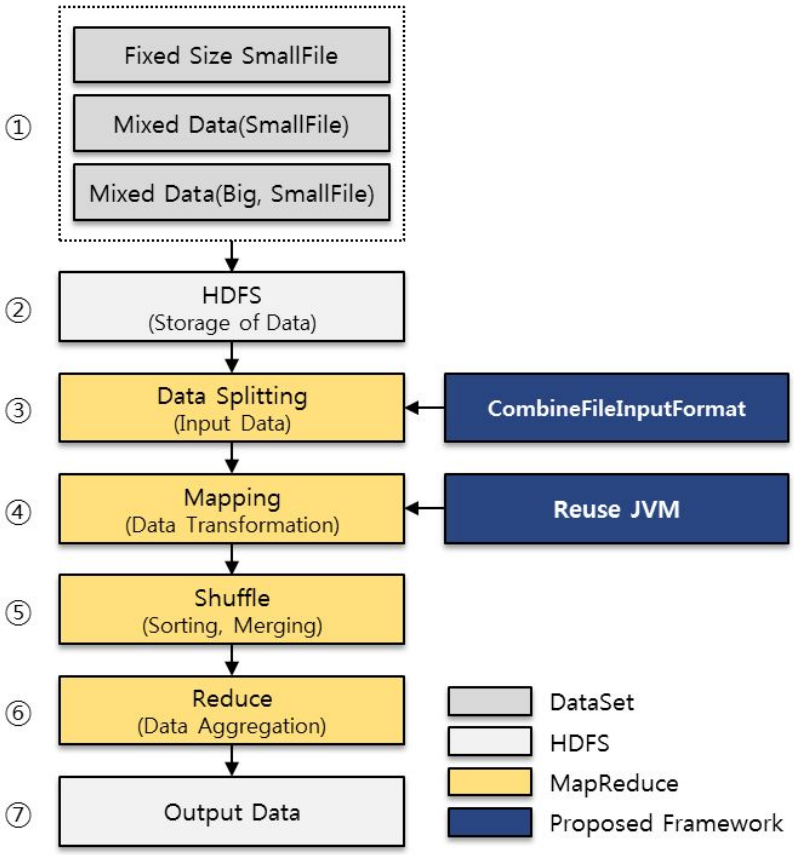
조선대학교
CHOSUN UNIVERSITY

Ⅲ. 대량의 스몰파일 처리성능 향상 방법

본 장에서는 하둡 MapReduce에서 대량의 스몰파일을 대상으로 다수의 파일을 하나의 Split으로 묶어 처리하는 CombineFileInputFormat 방법과 기존에 태스크마다 생성되던 JVM을 재사용하여 다수의 태스크가 하나의 JVM에서 순차적으로 실행되는 JVM 재사용 방법을 함께 사용한 MapReduce 처리성능 향상방법을 제안한다.

A. 시스템 구성도

[그림 3-1]은 본 논문에서 제안하는 MapReduce 처리성능 향상방법의 처리 과정을 도식화한 시스템 구성도이다.



[그림 3-1] 전체 시스템 구성도

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

본 논문에서 제안하는 처리성능 향상방법의 시스템 처리 과정은 다음과 같다.

① 고정된 크기의 스몰파일, 혼합 스몰파일, 빅파일과 스몰파일이 섞인 혼합 파일을 데이터세트로 사용한다.

② ①의 데이터세트를 HDFS에 블록으로 저장한다.

③ HDFS에 저장된 데이터 블록을 불러와 CombineFileInputFormat 설정값의 크기만큼 하나의 Split으로 묶어 Map Task에 전달한다.

④ Map Task는 설정된 JVM 재사용 설정값에 따라 JVM을 재사용하며 차례대로 ③에서 전달받은 파일들을 변형(Transformation)한다.

⑤ Map Task의 출력물을 정렬(Sorting)하고 병합(Merging)한다.

⑥ Shuffle 과정을 거친 데이터에서 사용자가 원하는 값만 추출 및 집계한다.

⑦ Reduce 과정을 거쳐 출력된 데이터를 다시 HDFS에 저장한다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

B. 대량의 스몰파일 처리성능 향상 방법

본 절에서는 2장에서 소개한 CombineFileInputFormat 방법과 JVM 재사용 방법을 함께 사용한 하둡 대량의 스몰파일 MapReduce 처리성능 향상 방법을 제안한다.

1. CombineFileInputFormat 적용 방법

CombineFileInputFormat 방법은 기존에 파일 하나당 Split을 생성하던 하둡의 다른 Input Format 방법과는 다르게, 다수의 파일을 하나의 Split으로 묶어 처리한다. CombineFileInputFormat 방법을 사용하면 대량의 스몰파일을 대상으로 생성되던 다수의 매퍼 수를 줄여 매퍼의 생성시간을 감소시키고, MapReduce 스케줄링 및 처리성능을 향상할 수 있다.

```
1.  public class smallcombinefileinputformat
2.  extends CombineFileInputFormat<smallcombinewritable, Text>  {
3.    public smallcombinefileinputformat (){
4.        super();
5.          setMaxSplitSize(134217728); // 128MB 설정값
6.      }
7.      public RecordReader<smallcombinewritable,Text>
8.      createRecordReader(InputSplit split,
9.          TaskAttemptContext context) throws IOException {
10.             return new
11.             CombineFileRecordReader<smallcombinewritable,
12.             Text>((CombineFileSplit)split, context,
13.             smallcombinerecordreader.class);
14.         }
15.     protected boolean isSplitable(JobContext context, Path file){
16.         return false; // 블록 단위로 못 나누게 한다
17.     }
18. }
```

[그림 3-2] CombineFileInputFormat을 사용하기 위한 Combine 클래스

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[그림 3-2]는 Combine 추상 클래스의 소스코드이다. CombineFileInputFormat은 추상 클래스임으로 사용하기 위해서는 [그림 3-2]와 같이 클래스를 생성하고, RecordReader 메소드를 구현해야 한다. [그림 3-2]의 소스 내용은 다음과 같다.

5번째 줄 setMaxSplitSize는 CombineFileInputFormat의 설정값을 설정하는 곳으로 다수의 파일을 하나로 묶는 Split의 크기를 설정한다. [그림 3-2]는 128MB로 설정돼있다.

7번째 줄의 RecordReader메서드는 smallcombinewritable로 읽어진 파일을 InputFormat에 따라 Mapper가 처리하기 편한 Pair(Key, Value) 형식으로 변환하는 역할을 한다.

15번째 줄의 isSplitable을 입력 포맷 클래스에서 지원하는 메서드로 true이면 블록 단위로 파일을 나눠서 split하고, false이면 파일 전체를 하나로 처리한다. CombineFileInputFormat은 다수의 파일을 하나의 Split으로 묶어 처리하기 때문에 [그림 3-2]에서는 false를 return하고 있다.

본 논문에서는 연도별, 월별로 항공기 출발지연 건수를 출력하는 맵리듀스 프로그램(DepartureDelayCount)에 CombineFileInputFormat을 사용하기 위해 [그림 3-2]와 같은 CombineFileInputFormat 클래스를 생성하고 RecoreReader 메서드를 추가로 작성하였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

2. JVM 재사용 방법

하둡 맵리듀스는 데이터를 분산병렬처리하기 위해 매퍼마다 별도의 JVM(Java Virtual Machine)이 생성되어 데이터를 처리한다. JVM 재사용 방법은 일정한 수의 매퍼를 하나의 JVM에서 순차적으로 실행하여 기존에 매퍼마다 생성되던 JVM의 생성시간을 감소시키고 처리성능을 높이는 방법이다.

```
1. <?xml version="1.0"?>
2. <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3.
4. <!-- Put site-specific property overrides in this file. -->
5.
6. <configuration>
7.         <property>
8.                 <name>mapred.job.tracker</name>
9.                 <value>oracle:9001</value>
10.        </property>
11.        <property>
12.                <name>mapred.job.reuse.jvm.num.tasks</name>
13.                <value>-1</value>
14.        </property>
15. </configuration>
```

[그림 3-3] JVM을 재사용하기 위한 환경설정 파일(mapred-site.xml)

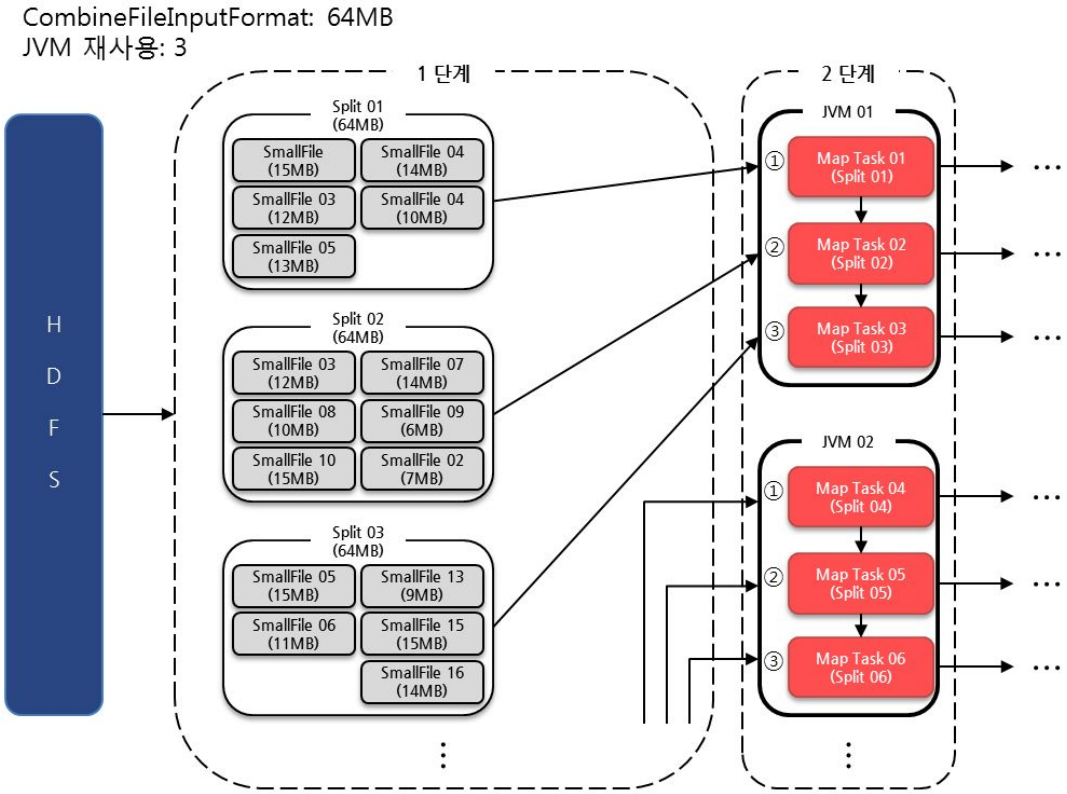
[그림 3-3]은 본 논문에서 JVM 재사용 방법의 실험을 진행하기 위해 작성한 mapred-site.xml 환경설정 파일이다. 8번째 줄의 mapred.job.tracker 파라미터는 잡트래커의 주소를 의미하며, 데이터노드는 mapred.job.tracker에게 맵리듀스 작업을 요청한다. JVM 재사용을 사용하기 위해서는 환경설정 파일의 12번째 줄의 mapred.job.reuse.num.tasks 파라미터를 정의하고 13번째 줄의 Value에 값을 넣어 사용할 수 있다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

3. 제안한 방법 처리과정

[그림 3-4]는 본 논문에서 제안하는 처리성능 향상방법의 처리 과정을 나타내고 있다.



[그림 3-4] 제안방법 처리과정(CombineFileInputFormat 64MB, JVM 재사용 3)

[그림 3-4]의 제안방법 처리 과정은 다음과 같다.

(1) HDFS에 저장되어있는 대량의 스몰파일을 불러온다.

(2) 불러온 데이터세트를 CombineFileInputFormat 설정값(64MB)에 맞게 하나의 Split으로 묶는다.

(3) 64MB 크기로 묶인 Split 파일들을 Mapper로 전송한다.

(4) JVM 재사용 설정값(3)에 따라 3개의 Mapper가 하나의 JVM에서 순차적으로 실행된다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

위의 과정을 풀어서 설명하면, 1단계는 HDFS에 블록으로 저장된 대량의 스몰파일 데이터세트를 불러온다. 다음 CombineFileInputFormat 설정값인 64MB의 크기로 스몰파일을 하나의 Split으로 묶는다. 64MB 크기로 먼저 묶인 Split부터 차례대로 Map Task로 보내진다. 2단계는 Split으로 묶여 보내진 파일들을 JVM을 재사용하며 Map Task로 처리한다. JVM 재사용 설정값이 3이기 때문에 3개의 Map Task가 하나의 JVM에서 순차적으로 실행되며, JVM 01에서 Map Task01이 먼저 처리되고 처리가 완료되면 Map Task02가 실행된다. Map Task02의 실행이 완료되면 Map Task03이 실행된다. Map Task에서 처리된 결과는 다음과정으로 보내진다.

CombineFileInputFormat 방법과 JVM 재사용 방법을 함께 사용함으로써 대량의 스몰파일로 인해 기존에 다수의 Map Task와 JVM이 생성되는 것과는 다르게 Map Task와 JVM의 생성시간을 줄여 스케줄링 및 처리성능을 향상할 수 있다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

Ⅳ. 실험 및 분석

본 장에서는 Ⅲ장에서 제안한 MapReduce 처리성능 향상 방법을 비교 평가하기 위해 하둡 의사분산모드와 완전분산모드 2가지 방식으로 하둡 클러스터를 구축하고 빅파일, 스몰파일, 혼합파일 등 다양한 데이터세트를 대상으로 실험을 진행하였다. 의사분산모드는 1대의 서버에서 5GB 파일을 대상으로 실험을 진행하였고, 완전분산모드는 총 6대의 컴퓨터에서 VMware 가상머신을 사용하여 11GB 파일을 데이터세트로 실험하였다.

A. 실험환경

실험환경은 하둡에서 테스트환경으로 많이 사용되는 의사분산모드와 실제로 기업에서 사용되는 완전분산모드 2가지 모드로 하둡 클러스터를 구축하였다.

[표 4-1] 하둡 의사분산모드 실험환경(서버 1대)

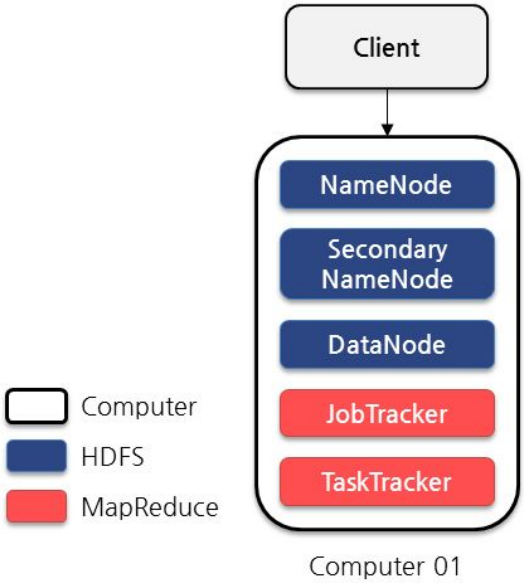
구 분		내 용
H/W	CPU	Intel(R) Xeon(R) CPU E5
	RAM	16GB
	HDD	1TB
S/W	운영체제	CentOS 6.6
	Hadoop 버전	Hadoop 1.2.1

[표 4-2] 하둡 완전분산모드 실험환경(서버 6대)

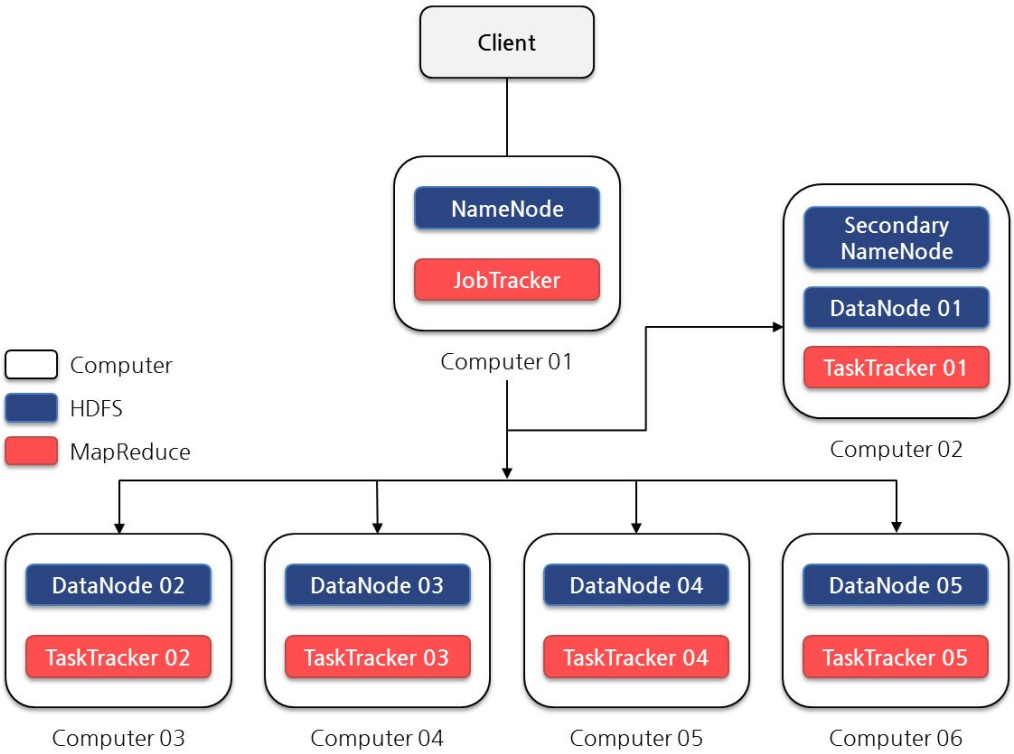
구 분		내 용
H/W	CPU(core)	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
	RAM	2GB
	HDD	200GB
S/W	가상머신	VMware 8.0.2
	운영체제	CentOS 6.6
	Hadoop 버전	Hadoop 1.2.1

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 4-1] 하둡 의사분산모드 구성도



[그림 4-2] 하둡 완전분산모드 구성도

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[표 4-1]과 [표 4-2]는 각각 하둡의사분산모드와 완전분산모드의 실험환경이고, [그림 4-1]과 [그림 4-2]는 [표 4-1]과 [표 4-2]의 실험환경을 사용하여 구축한 하둡 의사분산모드와 완전분산모드의 시스템 구성도이다. 의사분산모드는 [그림 4-1]과 같이 한 대의 서버 안에서 네임노드, 세컨더리네임노드, 데이터노드, 잡트래커, 태스크트래커가 모두 작동하는 방식으로, 하둡에서 테스트를 할 때 많이 사용되는 모드이다. 완전분산모드는 [그림 4-2]와 같이 물리적으로 다수의 서버를 하나의 클러스터로 묶어 사용하는 완전한 분산병렬처리시스템이다. 하둡 완전분산모드는 기업에서 하둡을 구축할 때 사용하는 모드로써, 본 논문에서는 6대의 컴퓨터에 VMware 가상머신을 사용하여, [그림 4-2]와 같이 하나의 클러스터로 묶어 구축하였다. [그림 4-2]의 시스템 구성도와 [그림 2-1]의 하둡 아키텍처의 다른 점은 [그림 4-2]의 Computer2, 즉 세컨더리네임노드에 데이터노드와 태스크트래커가 추가된 점이다. [그림 4-2]와 같이 구축하여도 하둡을 실행하는 데는 아무런 문제가 없다. 다만 보통 기업이나 실질적으로 하둡을 완전분산모드로 구축하여 사용하는 곳에서는 세컨더리네임노드를 [그림 2-1]의 하둡 아키텍처처럼 서버 1대에서 단독으로 사용하고 있다. 이는 시스템의 안전과 리커버리(Recovery)를 위함이다.

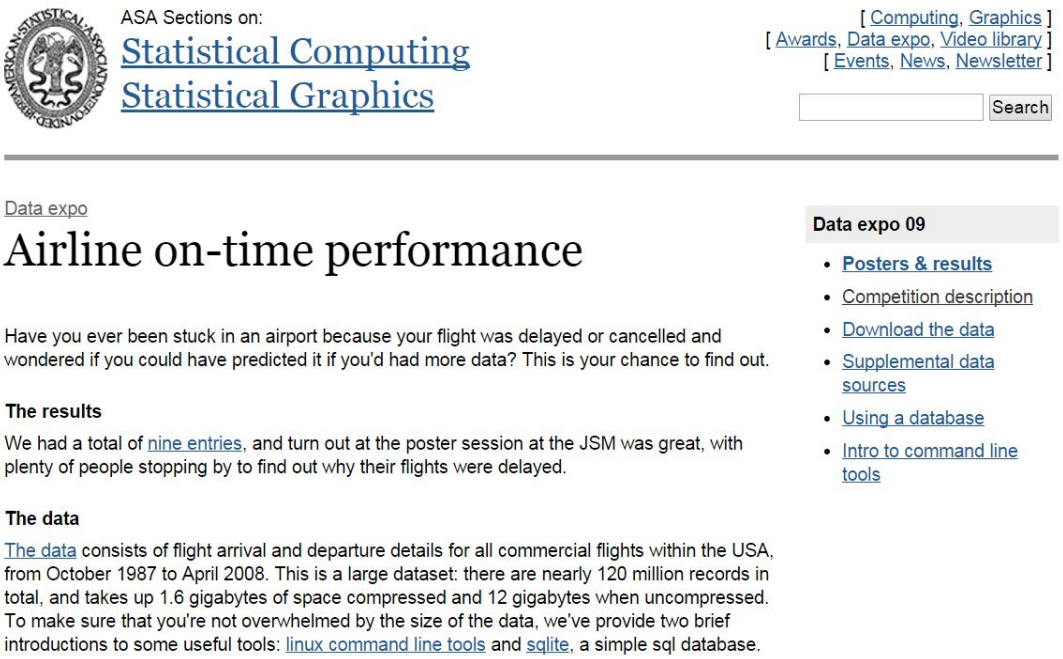
하둡 의사분산모드와 완전분산모드 2가지 모두 동일하게 운영체제는 CentOS 6.6 버전을 사용하였고, 하둡은 1.2.1 버전을 사용하였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

B. 실험데이터세트(Data Set)

실험에 사용한 데이터세트는 미국 규격협회(American Standards Association)에서 제공하는 미국 항공편 운항 통계 데이터로 1987년도부터 2008년도까지 총 22개의 파일로 구성되어 있으며, 약 11GB 크기이다. 파일형식은 쉼표(,)를 구분자로 하는 CSV(Comma Separated Value)형식의 텍스트 파일이며, 파일마다 29개의 칼럼(Column)으로 구성되어 있다. [그림 4-3]에서 내려받을 수 있다[15].



[그림 4-3] 미국 규격협회에서 제공하는 미국 항공편 운항 통계 데이터

[표 4-3] Linux Split 명령어

명령어	내 용
Split	Split [Option] [Filename] 예) Split -b 6m 2008.csv 6mb_ : 2008.csv 파일을 6mb로 분할한다. 분할된 파일: 6mb_aa, 6mb_ab, 6mb_ac, …

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

실험에 필요한 스몰파일을 얻기 위해 [표 4-3]의 리눅스(Linux) Split 명령어를 사용하여 [표 4-4], [표 4-5]와 같은 스몰파일 및 혼합파일 데이터세트를 생성하였다. 파일 수만 다를 뿐 파일 크기는 의사분산모드는 5GB, 완전분산모드는 11GB로 같다.

[표 4-4] 의사분산모드 실험데이터세트(5GB)

구 분		파일 수
빅파일		9개
스몰파일	4MB	1371개
	5MB	1098개
	6MB	915개

[표 4-5] 완전분산모드 실험데이터세트(11GB)

구 분		파일 수
빅파일		22개
스몰파일	4MB	2879개
	5MB	2305개
	6MB	1922개
	혼합스몰파일(4,5,6MB)	2374개
혼합파일 (빅파일, 스몰파일)	빅파일(7) : 스몰파일(3)	877개
	빅파일(3) : 스몰파일(7)	2024개

[표 4-5]의 4MB, 5MB, 6MB 혼합파일은 11GB 크기의 빅파일을 3:3:4의 비율로 Split 명령어를 사용하여 각각 4MB, 5MB, 6MB로 나눈 데이터이다. [표 4-5] 하단의 빅파일과 스몰파일이 합쳐진 혼합파일은 비율별로 빅파일은 기존의 csv 파일을 그대로 사용하였고, 스몰파일은 4MB로 분할시켜 생성된 데이터세트이다. 본 논문에서는 4MB, 5MB, 6MB의 고정된 크기의 스몰파일뿐만 아니라, 다양한 크기가 혼합되어있는 스몰파일과 빅파일과 스몰파일이 3:7 또는 7:3으로 섞여 있는 데이터세트를 사용함으로써 실험결과 및 연구의 정확성을 높였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

C. 실험평가 방법 및 결과분석

본 절에서는 본 논문에서 제안하는 방식과 기존방식의 처리성능을 비교하기 위해 다양한 데이터세트와 CombineFileInputFormat과 JVM 재사용방법의 설정값을 사용하여 실험을 진행하고 처리성능을 비교 평가한다.

1. 실험 방법

본 논문의 실험은 의사분산모드와 완전분산모드 2가지 하둡 모드에서 각각 [표 4-4]와 [표 4-5]의 데이터세트를 대상으로 실험을 진행하였다.

[표 4-6] 실험방법

실 험	내 용
CombineFileInputFormat 방법	CombineFileInputFormat 설정값을 32, 64, 128, 256, 512, 1024MB로 설정 후 실험
JVM 재사용 방법	JVM 재사용 설정값을 -1과 2, 3 … 등 값을 1씩 증가하며 최적의 설정값을 찾는 실험
기존 MapReduce 처리 방법	아무런 설정도 하지 않은 기존 하둡 MapReduce의 실험
제안한 방법	CombineFileInputFormat과 JVM 재사용을 함께 사용하여 실험

실험 방법은 [표 4-6]과 같이 총 4가지 방법으로 실험을 진행하였다.

(1) CombineFileInputFormat 방법의 최적 설정값을 찾기 위해 32MB, 64MB, 128MB, 256MB, 512MB, 1024MB 총 6가지 설정값을 사용하여 설정값에 따른 처리성능을 비교하고 최적의 설정값을 찾는 실험을 진행하였다.

(2) JVM 재사용 방법의 최적 설정값을 찾기 위해 -1 설정값과 2부터 1씩 증가하며 설정값에 따른 처리성능을 비교하고 최적의 설정값을 찾는 실험을 진행하였다.

(3) 하둡 클러스터에 어떠한 세팅도 하지 않은 순수한 MapReduce의 처리 성능을 비교하기 위한 실험을 진행하였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

(4) 제안한 방법의 처리성능을 비교 평가하기 위해 (1)과 (2)의 실험을 통해 알게 된 최적의 설정값을 사용한 제안 방법과 (1), (2), (3)의 처리 성능을 비교 평가하였다.

모든 실험에는 [표 4-7]의 연도별, 월별 항공기 출발지연 건수를 출력해주는 MapReduce 프로그램을 같이 사용하였으며, CombineFileInputFormat 방법에는 3장의 [그림 3-2]와 같이 CombineClass를 생성하고 RecordReader 메서드를 추가로 작성하여 실험을 진행하였다.

[표 4-7] MapReduce 프로그램

프로그램	내 용
DepartureDelayCount	연도별, 월별 항공기 출발지연건수를 출력

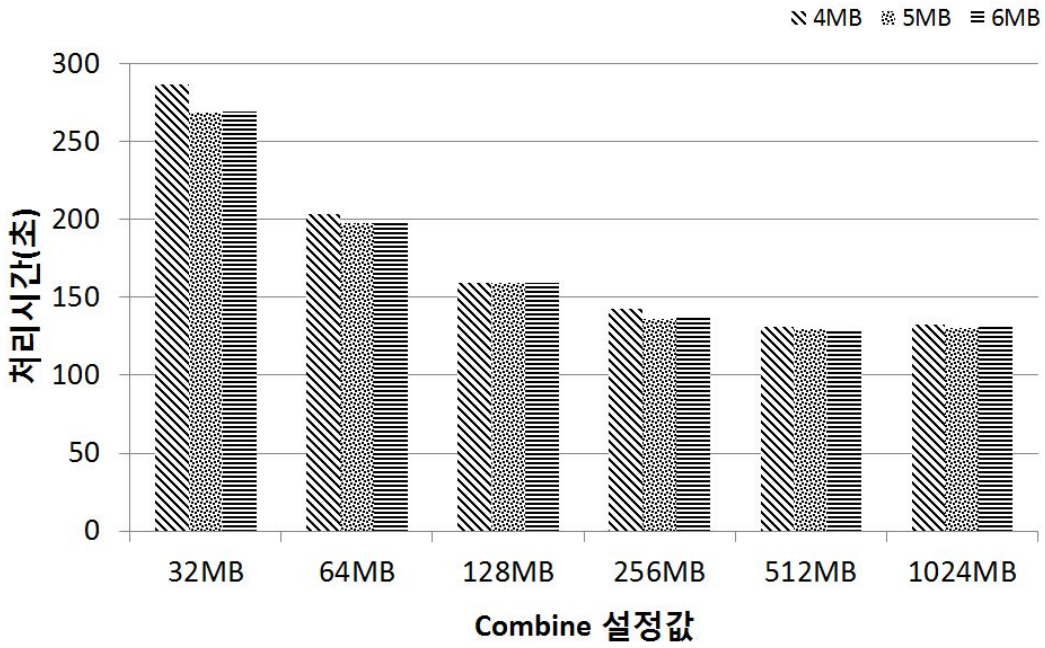
2. 실험 결과 분석

a. CombineFileInputFormat 적용 방법

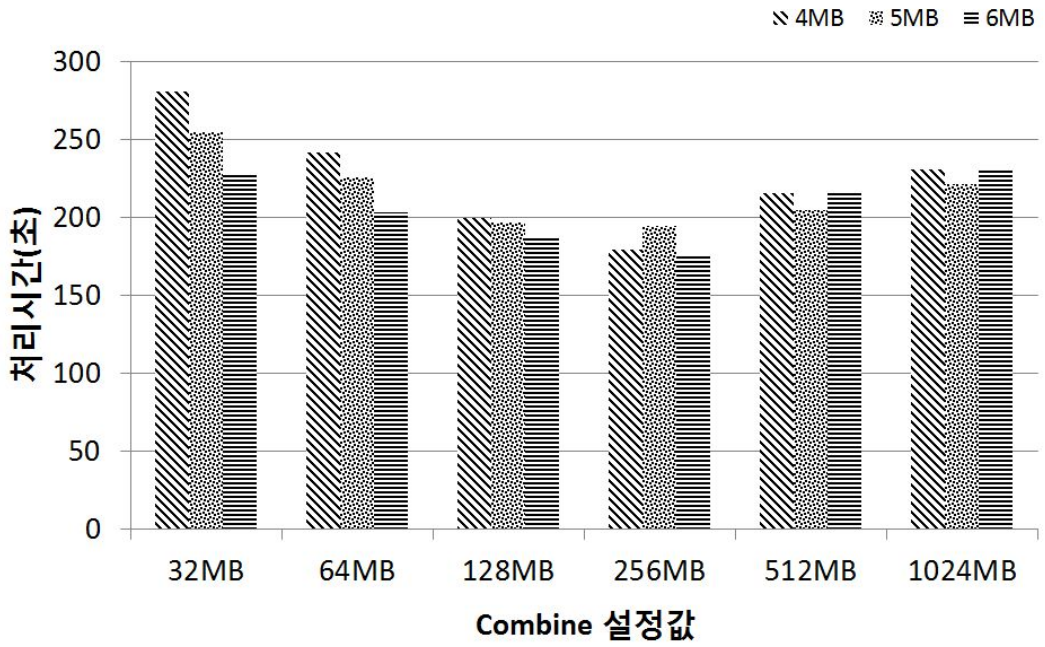
[그림 4-4]와 [그림 4-5]는 CombineFileInputFormat 설정값의 처리시간을 하둡 의사분산모드와 완전분산모드에서 비교 평가한 실험 결과인 [그림 4-6]과 [그림 4-7]을 토대로 작성한 그래프이다. 하둡 블록 크기를 기준으로 32MB, 64MB, 128MB, 256MB, 512MB, 1024MB 총 6가지의 설정값을 사용하여 실험을 진행하였다. CombineFileInputFormat 설정값이 하둡 의사분산모드는 512MB일 때, 완전분산모드는 256MB일 때 가장 우수한 성능을 보이는 것을 [그림 4-4]와 [그림 4-5]의 그래프를 보고 알 수 있다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 4-4] Combine 설정값에 따른 처리결과(의사분산모드)



[그림 4-5] Combine 설정값에 따른 처리결과(완전분산모드)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

a) 4MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	04분46초	171개
64MB	03분23초	86개
128MB	02분39초	43개
256MB	02분22초	22개
512MB	**02분11초**	11개
1024MB	02분12초	6개

b) 5MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	04분28초	157개
64MB	03분17초	84개
128MB	02분39초	43개
256MB	02분16초	22개
512MB	**02분09초**	11개
1024MB	02분10초	6개

c) 6MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	04분29초	153개
64MB	03분19초	83개
128MB	02분39초	42개
256MB	02분17초	22개
512MB	**02분09초**	11개
1024MB	02분12초	6개

[그림 4-6] CombineFileInputFormat 설정값에 따른 처리시간(의사분산모드)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

a) 4MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	04분41초	358개
64MB	04분02초	179개
128MB	03분20초	90개
256MB	**03분**	45개
512MB	03분36초	23개
1024MB	03분51초	12개

b) 5MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	04분15초	329개
64MB	03분46초	176개
128MB	03분17초	89개
256MB	**03분15초**	45개
512MB	03분25초	23개
1024MB	03분42초	12개

c) 6MB 처리시간

CombineFileInputFormat 설정값	처리시간	Map 태스크 수
32MB	03분48초	320개
64MB	03분24초	174개
128MB	03분07초	88개
256MB	**02분57초**	45개
512MB	03분36초	23개
1024MB	03분52초	12개

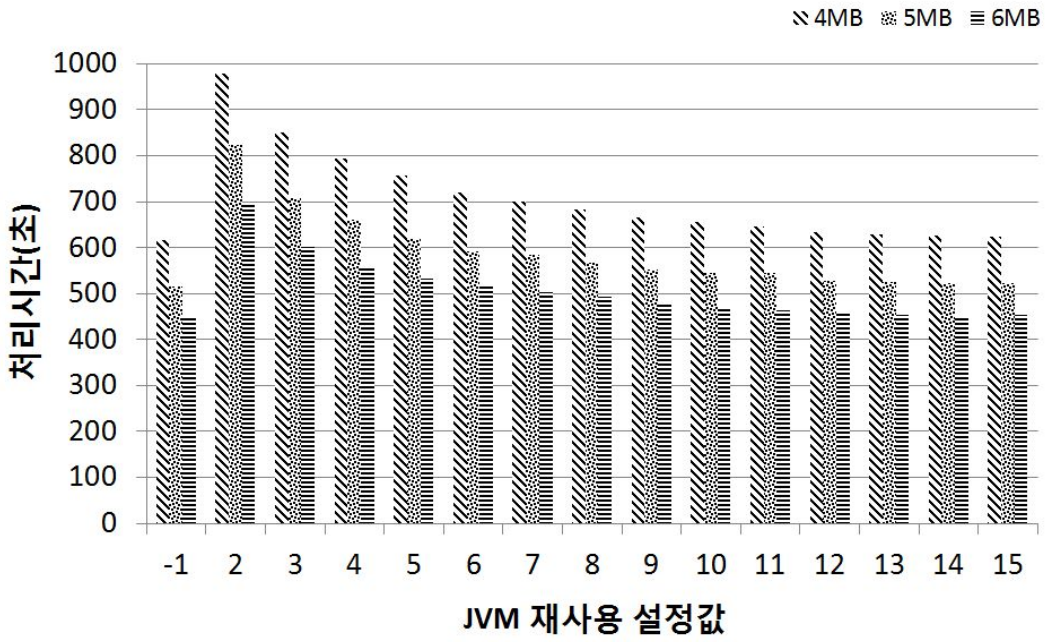
[그림 4-7] CombineFileInputFormat 설정값에 따른 처리시간(완전분산모드)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

b. JVM 재사용 방법

[표 4-8]과 [표 4-9]는 JVM 재사용 방법의 다양한 설정값에 따른 처리시간을 하둡 의사분산모드와 완전분산모드에서 비교 평가한 실험 결과이다. [그림 4-8]과 [그림 4-9]는 [표 4-8]과 [표 4-9]는 실험결과를 토대로 작성한 그래프이다.

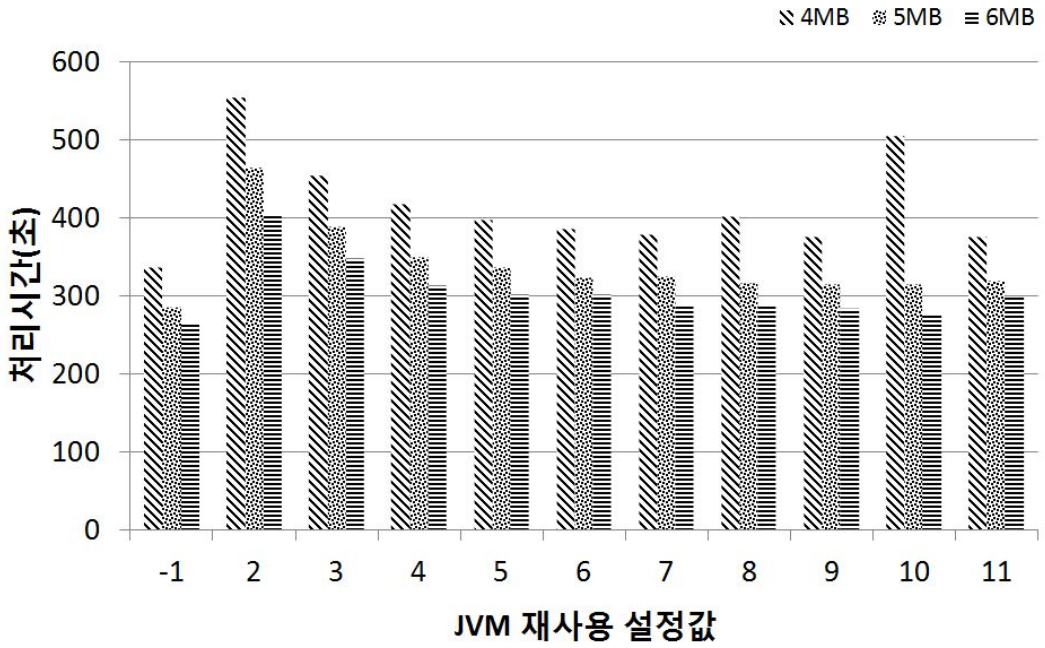


[그림 4-8] JVM 재사용 설정값에 따른 처리결과(의사분산모드)

[그림 4-8]의 하둡 의사분산모드에서 JVM 재사용 설정값에 따른 처리시간을 비교 평가한 결과 스몰파일 데이터세트 4MB, 5MB, 6MB 모두 설정값 2에서 11까지는 값이 증가함에 따라 처리성능이 향상되었다. 설정값 12 이상부터는 비슷한 처리성능을 보였고, 4MB, 5MB, 6MB 데이터세트 모두 -1로 하나의 JVM에서 모든 매퍼를 처리하였을 때 가장 우수한 처리성능을 보였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 4-9] JVM 재사용 설정값에 따른 처리결과(완전분산모드)

[그림 4-9]의 하둡 완전분산모드에서 JVM 재사용 설정값에 따른 처리시간을 비교 평가한 결과 [그림 4-8]의 하둡의사분산모드의 비슷한 결과를 보였으며, 설정값을 -1로 하였을 때 4MB, 5MB, 6MB 데이터세트 모두 가장 우수한 처리성능을 보였다. 본 장에서는 CombineFileInputFormat과 JVM 재사용 방법의 설정값에 따른 처리시간을 비교 평가하기 위해 4MB, 5MB, 6MB로 이루어진 스몰파일을 대상으로 실험을 진행하였다. CombineFileInputFormat 방법은 설정값을 256MB, 512MB로 설정하였을 때, JVM 재사용 방법은 설정값을 -1로 설정하였을 때 가장 우수한 처리성능을 보였다. 위의 2가지 방법을 함께 사용하여 Split 되는 매퍼 수를 줄이고 JVM을 재사용하면 더욱더 향상된 처리성능을 보일 것으로 예상한다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[표 4-8] JVM 재사용 설정값에 따른 처리결과(의사분산모드)

JVM 재사용 설정값	처리시간			Map 태스크 수
	4MB	5MB	6MB	
-1	**10분15초**	**08분35초**	**07분26초**	4MB: 1371개 5MB: 1098개 6MB: 915개
2	16분17초	13분42초	11분35초	
3	14분09초	11분46초	10분	
4	13분12초	10분59초	09분20초	
5	12분36초	10분18초	08분51초	
6	11분59초	09분52초	08분35초	
7	11분39초	09분44초	08분23초	
8	11분21초	09분27초	08분12초	
9	11분06초	09분11초	08분	
10	10분55초	09분03초	07분46초	
11	10분45초	09분03초	07분43초	
12	10분34초	08분48초	07분40초	
13	10분28초	08분45초	07분34초	
14	10분25초	08분42초	07분29초	
15	10분24초	08분41초	07분33초	

[표 4-9] JVM 재사용 설정값에 따른 처리결과(완전분산모드)

JVM 재사용 설정값	처리시간			Map 태스크 수
	4MB	5MB	6MB	
-1	**05분36초**	**04분45초**	**04분24초**	4MB: 2879개 5MB: 2305개 6MB: 1922개
2	09분13초	07분44초	06분43초	
3	07분33초	06분28초	05분48초	
4	06분57초	05분50초	05분13초	
5	06분37초	05분37초	05분02초	
6	06분26초	05분24초	05분02초	
7	06분19초	05분25초	04분48초	
8	06분42초	05분16초	04분47초	
9	06분15초	05분15초	04분44초	
10	08분24초	05분14초	04분36초	
11	06분16초	05분19초	04분58초	

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

c. 기존 MapReduce 처리

[표 4-10]과 [표 4-11]은 하둡 의사분산모드와 완전분산모드에서 아무런 세팅도 하지 않은 순수한 하둡 MapReduce의 처리시간을 나타내고 있다. [표 4-10]의 빅파일은 187개의 매퍼를 사용하여 02분 48초의 시간이 소요되었다. 반면 4MB, 5MB, 6MB 스몰파일은 각각 13분 35초, 11분 14초, 09분 51초의 시간이 소요되어 빅파일에 비해 작게는 3배에서 크게는 4배까지 처리시간이 증가한 것을 알 수 있다. 또한, 하둡 의사분산모드와 완전분산모드 모두 빅파일을 제외한 처리결과가 스몰파일 수와 같은 Map 태스크 수를 사용하였다. 이는 서론에서 제기했던 하둡에서 대량의 스몰파일을 처리할 때 스몰파일마다 매퍼가 생성되어 MapReduce의 스케줄링 및 처리시간이 증가하는 단점을 잘 보여주고 있다.

[표 4-10] MapReduce 기존 처리시간(의사분산모드)

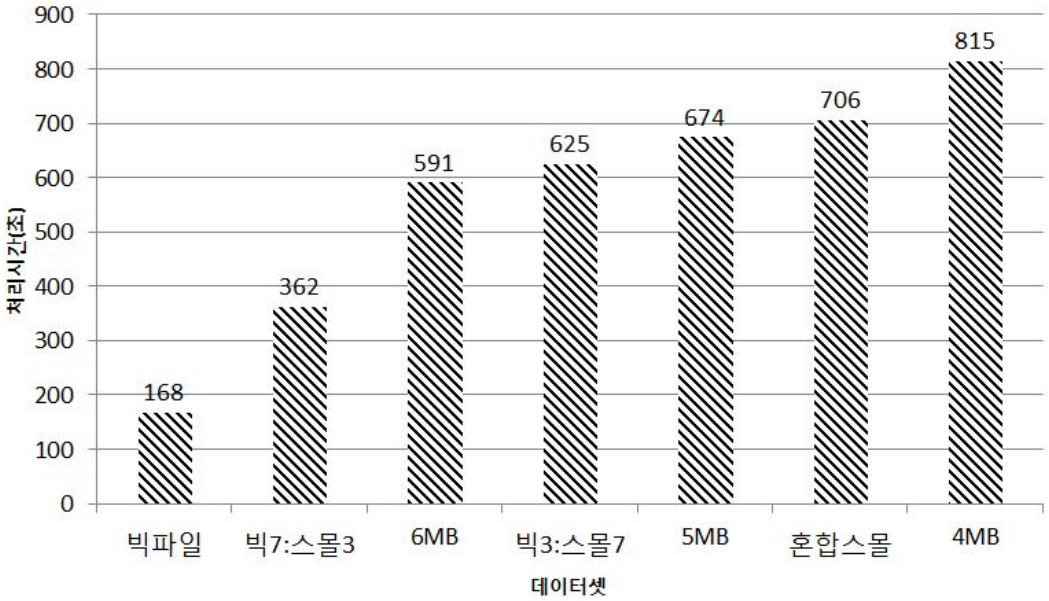
구 분		처리시간	Map 태스크 수
빅파일		03분28초	88개
스몰파일	4MB	23분20초	1371개
	5MB	19분17초	1098개
	6MB	16분23초	915개

[표 4-11] MapReduce 기존 처리시간(완전분산모드)

구 분		처리시간	Map 태스크 수
빅파일		02분48초	187개
스몰파일	4MB	13분35초	2879개
	5MB	11분14초	2305개
	6MB	09분51초	1922개
	혼합스몰파일(4,5,6MB)	11분46초	2374개
혼합파일 (빅파일,스몰파일)	빅파일(7):스몰파일(3)	06분02초	992개
	빅파일(3):스몰파일(7)	10분25초	2074개

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 4-10] MapReduce 기존 처리시간(완전분산모드)

[그림 4-10]은 하둡 완전분산모드에서 실험한 MapReduce 기존 처리시간을 [표 4-11]을 바탕으로 그린 그래프이다. 빅파일에서부터 4MB 스몰파일까지 데이터세트의 파일 수에 따라 처리시간이 증가하는 것을 알 수 있다. 스몰파일마다 JVM이 생성되고 그 위에 매퍼가 생성되기 때문에 파일 수가 늘어날수록 처리시간 또한 증가한다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

d. 제안한 방법

[그림 4-11]은 본 논문에서 제안한 방법의 처리시간이다. JVM 재사용 설정값은 가장 우수한 처리성능을 보였던 -1로 같게 설정하고, CombineFileInputFormat 설정값은 32, 64, 128, 256, 512, 1024 총 6가지로 설정하여 실험을 진행하였다. 실험결과 CombineFileInputFormat 설정값을 128MB 또는 256MB로 설정하고, JVM 재사용 설정값을 -1로 설정하였을 때 가장 좋은 성능을 보였다. [그림 4-12]부터 [그림 4-17]까지는 [그림 4-11]의 가장 좋은 성능을 보인 제안방법 처리시간을 토대로 기존처리시간과 제안방법 처리시간, CombineFileInputFormat 방법 및 JVM 재사용 방법을 각각 데이터세트 별로 비교하는 그래프이다.

a) 제안한방법 처리시간 1

CombineFileInputFormat 설정값	처리시간		
	4MB	5MB	6MB
32MB	03분40초	03분01초	03분29초
64MB	02분54초	02분56초	02분58초
128MB	02분54초	**02분49초**	**02분52초**
256MB	**02분51초**	03분05초	02분54초
512MB	03분03초	03분15초	03분21초
1024MB	03분47초	04분03초	03분40초

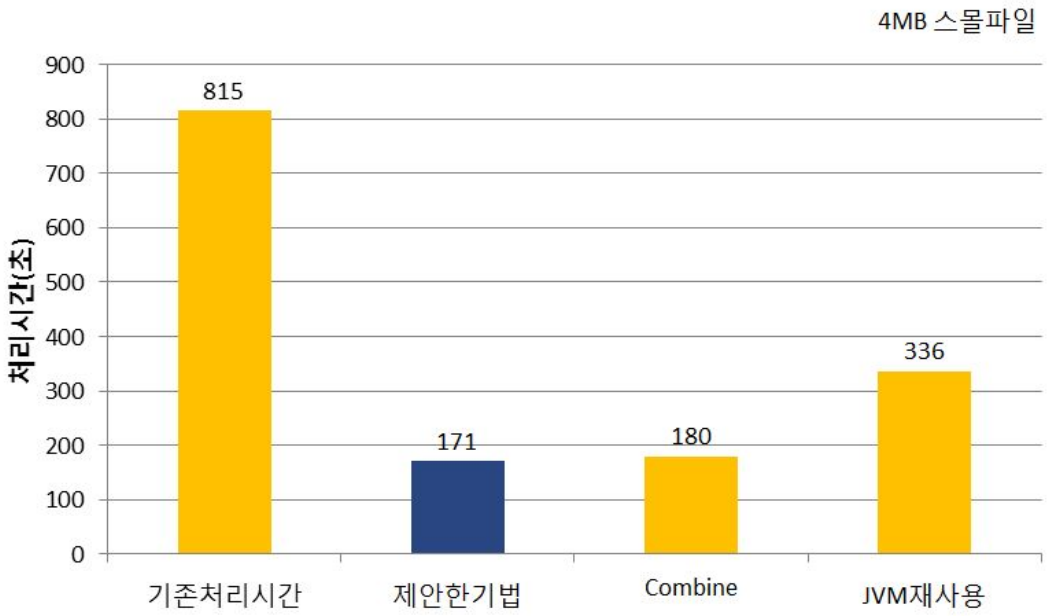
b) 제안한방법 처리시간 2

CombineFileInputFormat 설정값	처리시간		
	혼합 스몰파일	빅7:스몰3	빅3:스몰7
32MB	03분13초	04분	04분26초
64MB	03분07초	03분50초	04분23초
128MB	**03분05초**	**03분05초**	**03분30초**
256MB	03분12초	03분35초	03분45초
512MB	03분43초	04분35초	04분03초
1024MB	03분43초	04분17초	04분30초

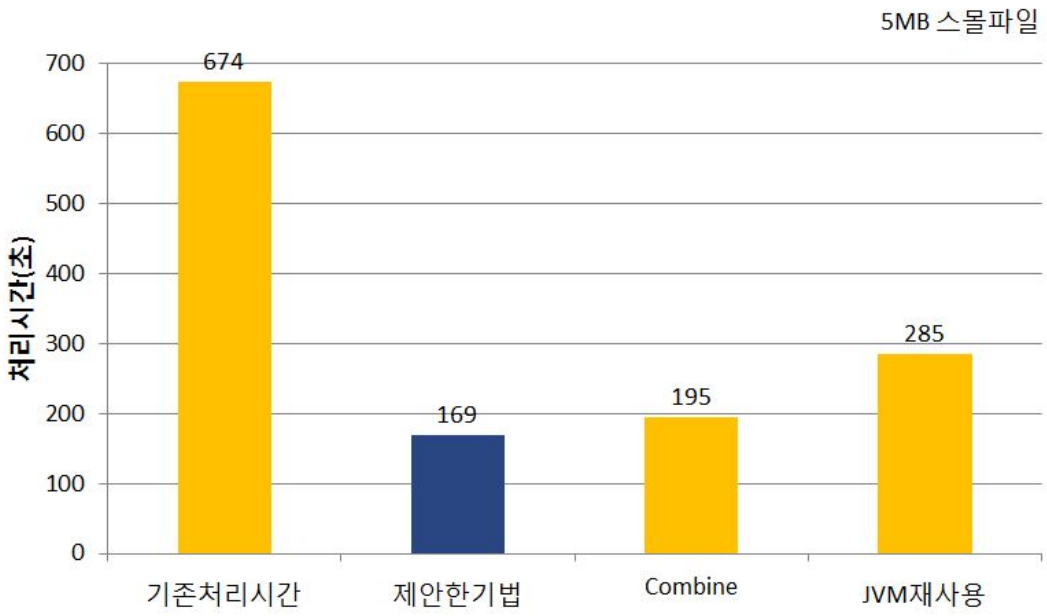
[그림 4-11] 제안한방법 처리시간 비교(완전분산모드)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



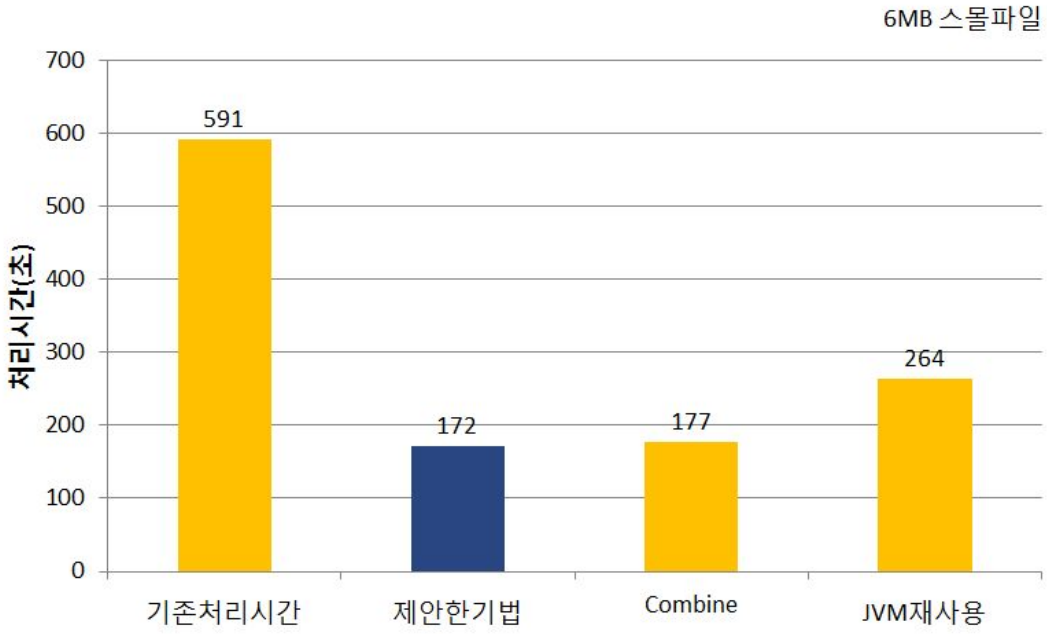
[그림 4-12] 제안방법 처리시간(데이터세트: 4MB)



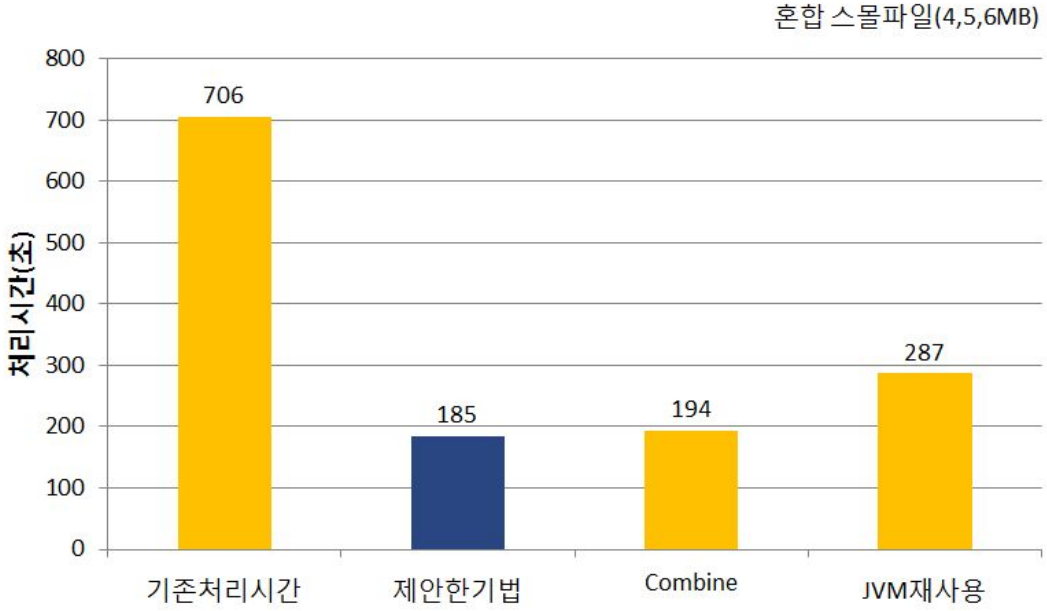
[그림 4-13] 제안방법 처리시간(데이터세트: 5MB)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



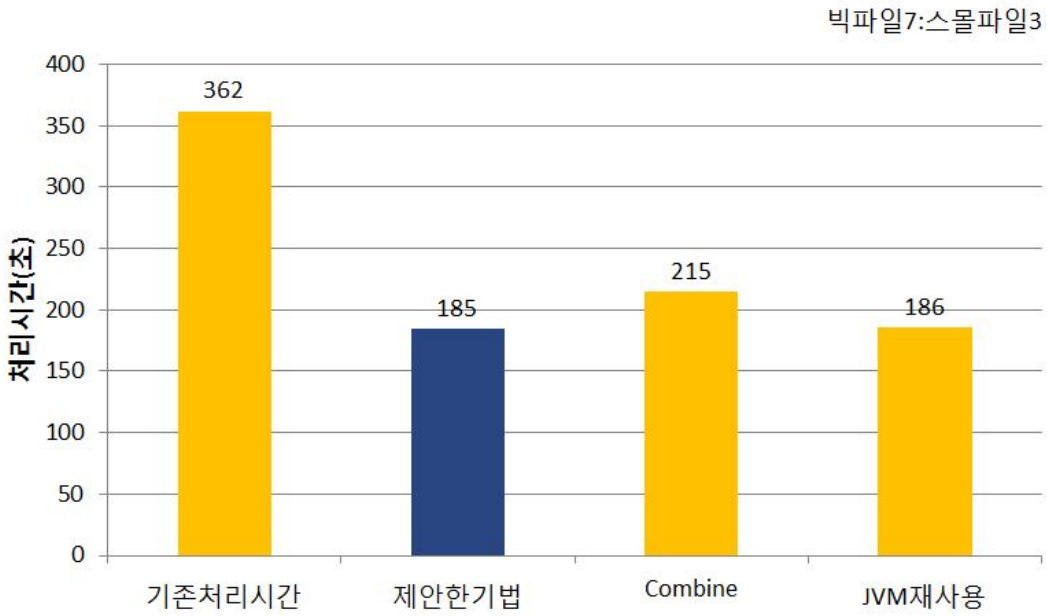
[그림 4-14] 제안방법 처리시간(데이터세트: 6MB)



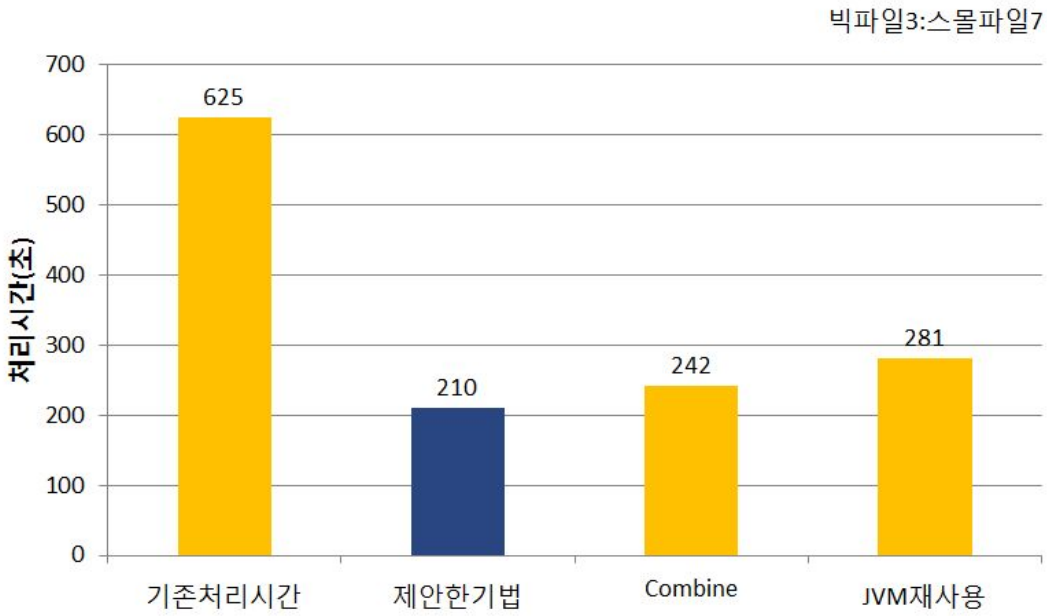
[그림 4-15] 제안방법 처리시간(혼합 스몰파일 4,5,6MB)

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY



[그림 4-16] 제안방법 처리시간(빅파일7:스몰파일3)



[그림 4-17] 제안방법 처리시간(빅파일3:스몰파일7)

[그림 4-12]부터 [그림 4-17]까지 모두 제안한 방법이 가장 우수한 처리성능을

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

보였으며, 그다음으로는 CombineFileInputFormat 방법, JVM 재사용방법 순으로 좋은 성능을 보였다. 제안방법과 기존처리시간을 비교했을 때 작게는 3배에서 크게는 10배까지 향상된 성능을 보이는 것을 알 수 있다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

3. 처리성능 비교 및 평가

본 논문에서 제안하는 방법의 처리성능을 비교 평가하기 위하여 [수식 4-1]을 사용하여 처리성능 향상도를 계산하였다.

$$\text{성능향상도}(\%) = \frac{\text{기존처리시간(초)} - \text{제안방법처리시간(초)}}{\text{기존처리시간(초)}} * 100 \quad \text{[수식 4-1]}$$

다음 [표 4-12]부터 [표 4-17]까지는 각각의 데이터세트별로 제안방법과 기존 맵리듀스 처리시간, 제안방법과 CombineFileInputFormat 처리시간, 제안방법과 JVM 재사용 처리시간의 처리성능을 [수식 4-1]을 사용하여 계산한 향상도를 나타낸 표다. 기존 맵리듀스 처리시간과 비교했을 때 최대 79%의 성능이 향상된 것을 알 수 있다.

[표 4-12] 4MB 스몰파일 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	815	171	79.0
CombineFileInputFormat	180		5.0
JVM 재사용	336		49.1

[표 4-13] 5MB 스몰파일 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	674	169	74.9
CombineFileInputFormat	195		13.3
JVM 재사용	285		40.7

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[표 4-14] 6MB 스몰파일 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	591	172	70.8
CombineFileInputFormat	177		2.8
JVM 재사용	264		34.8

[표 4-15] 혼합 스몰파일 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	706	185	73.7
CombineFileInputFormat	194		4.6
JVM 재사용	287		35.5

[표 4-16] 혼합파일(빅7:스몰3) 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	362	185	48.8
CombineFileInputFormat	215		13.9
JVM 재사용	186		0.5

[표 4-17] 혼합파일(빅3:스몰7) 처리성능 향상도

구 분	기존 처리시간(초)	제안방법 처리시간(초)	향상도(%)
기존 맵리듀스 처리	625	210	66.4
CombineFileInputFormat	242		13.2
JVM 재사용	281		25.2

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

CombineFileInputFormat은 스몰파일만 있는 데이터세트를 대상으로한 실험결과를 기존 맵리듀스 처리시간과 비교했을 때 스몰파일 개수가 증가함에 따라 처리성능이 향상하였다. 빅파일과 스몰파일이 섞인 혼합파일보다 스몰파일에서 더욱 좋은 성능을 보였다. 이는 빅파일 같은 경우 기존 파일 크기가 크기 때문에 HDFS에 데이터를 저장할 때 블록의 공간을 모두 채워 CombineFileInputFormat을 사용하지 않아도 좋은 성능을 보이기 때문이다. 또한, 5MB 스몰파일과 혼합스몰파일 실험결과보다 4MB 스몰파일 실험결과에서 스몰파일 개수가 늘어났음에도 불구하고 가장 좋은 처리성능을 보였다. 이는 4MB 스몰파일이 CombineFileInputFormat의 설정값(예: 64MB)에 따라 하나의 Split으로 묶일 때 5MB 스몰파일과 혼합스몰파일보다 수월하기 때문이라고 생각된다.

JVM 재사용은 CombineFileInputFormat과는 다르게 빅파일과 스몰파일이 섞인 혼합파일과 스몰파일 모두 고르게 처리성능이 향상되었다. 다수의 맵 태스크를 하나의 JVM에서 순차적으로 실행하기 때문에 파일의 크기보다는 파일의 개수에 따라 처리성능이 좌우된다는 것을 알 수 있다.

제안방법은 모든 데이터세트에서 기존 맵리듀스 처리, CombineFileInputFormat, JVM 재사용보다 향상된 처리성능을 보였다. 또한 빅파일과 스몰파일이 섞인 혼합파일보다 스몰파일에서 더욱 좋은 성능을 보인 CombineFileInputFormat과는 다르게 모든 데이터세트에서 고르게 좋은 성능을 보였다.

[표 4-18] 평균 처리성능 향상도

구 분	평균 향상도(%)
기존 맵리듀스 처리	68.9
CombineFileInputFormat	8.8
JVM 재사용	30.9

[표 4-18]은 [표 4-12]부터 [표 4-17]까지 총 6가지 데이터세트의 기존 맵리듀스 처리, CombineFileInputFormat, JVM 재사용 향상도를 각각 더하고 6으로 나누어 계산한 향상도의 평균값이다. 본 논문에서 제안한 방법과 각각 기존 맵리듀스 처리, CombineFileInputFormat, JVM 재사용 방법과 비교 평가했을 때 68.9%, 8.8%, 30.9% 처리성능이 향상하였다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

Ⅴ. 결론

본 논문에서는 하둡에서 CombineFileInputFormat 방법을 사용하여 다수의 스몰파일을 하나의 Split으로 묶어 매퍼수를 줄이고, JVM 재사용 방법을 사용하여 매퍼당 생성되는 JVM 생성시간을 감소시켜 MapReduce 처리성능을 향상하는 방법을 제안하였다.

하둡에서 대량의 스몰파일을 저장하고 처리할 때 생기는 문제점을 해결하는 기존 연구들은 빅파일병합과 같이 기존처리시간의 2~3배에 달하는 전처리과정이 필요하거나, 분산캐시 방법과 같이 하둡 시스템의 자원을 사용하는 단점들이 존재한다. 제안하는 방법은 전처리과정도 필요하지 않고, 하둡의 자원도 사용하지 않으며, 환경설정파일을 수정하고 JAVA 언어로 작성된 코드만 생성한다면 간단하게 처리성능을 향상할 수 있다. 제안한 방법에 대한 실험결과 CombineFileInputFormat 설정값을 128MB 또는 256MB로 설정하고 JVM 재사용 설정값을 -1로 설정하였을 때 가장 우수한 처리성능을 보였다. 기존 MapReduce 처리결과와 비교했을 때 작게는 3배에서 크게는 10배까지 처리성능이 향상되었다. 스몰파일 4MB, 5MB, 6MB 외에도 혼합스몰파일(4,5,6MB), 빅파일과 스몰파일이 각각 3:7, 7:3으로 섞인 혼합파일을 대상으로 한 실험에서도 우수한 처리성능을 보였다.

본 연구는 CombineFileInputFormat 방법과 JVM 재사용 방법을 함께 사용하여 처리성능을 향상했지만 CombineFileInputFormat 방법과 JVM 재사용 방법의 파일 수나 데이터 크기에 따른 최적의 설정값을 제시하지는 못했다. 따라서 본 논문의 향후 연구로는 파일 수와 데이터 크기에 따른 CombineFileInputFormat 방법과 JVM 재사용 방법의 최적 설정값을 찾아주는 알고리즘을 개발하는 연구가 필요하다. 또한, Job마다 생성해야 하는 Combine Class와 RecordReader 메서드를 라이브러리로 제공한다면 본 논문에서 제안하는 방법이 더욱더 많이 활용될 것으로 기대된다.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

참고문헌

[1] EMC, http://www.emc.com/about/news/press/2012/20121211-01.htm

[2] 김창복, 정재필, “하둡 플랫폼을 이용한 대량의 스몰파일 처리방법,” 한국항행학회논문지, 제 18권, 제 4호, pp. 401-408, 2014.

[3] K.Y. Han, “Do it! Hadoop with Big Data,” Easys Pub, 2013.

[4] W. Tom, “Hadoop : The Definitive Guide,” Hanbit Media, 2013.

[5] J.H. Jung, “Beginning Hadoop Programming Development and Operations,” Wikibooks, 2015.

[6] MapReduce Dataflow, http://blog.cloudera.com/blog/

[7] 최철웅, 김판구, “MapReduce에서 JVM 재사용 기법과 태스크 수에 따른 상관관계 연구,” 스마트미디어학회 추계학술대회 논문집, pp. 80-83, 2015.

[8] 김대호, “이기종 클러스터를 위한 효율적인 하둡 부하분산,” 숭실대학교, 석사학위논문, 2015.

[9] 최철웅, 김정인, 김판구, “MapReduce에서 Reuse JVM을 이용한 대규모 스몰파일 처리성능 향상 방법에 관한 연구,” 한국멀티미디어학회 눈문지, 제 18권, 제 9호, pp. 1098-1104, 2015.

[10] 이원재, 박석, “하둡에서 대규모 스몰파일 문제 해결을 위한 클러스터링 기반의 Map/Reduce 성능을 개선하는 데이터 병합 기법,” 한국정보과학회 동계학술발표 논문집, pp. 180-182, 2014.

[11] S. Chandrasekar, R. Dakshinamurthy, P. G. Sechakumar, B. Prabavathy and Chitra Babu, "A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System," Proceeding of International Conference on Computer Communication and Informatics(ICCCI-2013), pp. 1-8, January 2013.

[12] Jing Zhang, Gongqing Wu, Xuegang Hu and Xindong Wu, "A Distributed Cache for Hadoop Distributed File System in Real-time Cloud Services," Proceeding of International Conference on Grid Computing, pp. 12-21, September 2012.

Collection @ chosun

조선대학교
CHOSUN UNIVERSITY

[13] 오현교, 김기연, 황재민, 박준호, 임종태, 복경수, 유재수, “HDFS에서 스몰 파일의 효율적인 접근을 위한 분산 캐시 관리 기법,” 한국콘텐츠학회 논문지, 제 14권, 제 11호, pp. 28-38, 2014.

[14] The Small Files Problem, http://blog.cloudera.com/blog/

[15] 미국규격협회, http://stat-computing.org/dataexpo/2009/

[16] 최철웅, 김정인, 김판구, “MapReduce에서 Reuse JVM을 이용한 대량의 스몰 파일 처리성능 향상방법,” 한국멀티미디어학회 춘계학술대회 논문집, 제 18권 제 1호, 2015.

[17] Hadoop 1.2.1 Documentation, https://hadoop.apache.org/

[18] 곽재혁, 김상완, 허태상, 황순옥, “러스터 파일 시스템 기반 하둡 맵리듀스 실행 환경 구현 및 성능 분석,” 정보과학회 컴퓨팅의 실제 논문지, 제 21권, 제 8호, pp. 561-566, 2015.

[19] 최성운, 박승훈, 최현수, 윤성로, “파일 크기를 고려한 하둡 환경 설정 최적화,” 대한전자공학회 하계학술대회, 제 36권, 제 4호, pp. 1297-1300, 2013.

[20] Xiao Jun Liu, Chong Peng, Zhi Chao Yu, "Research on the Small Files Problem of Hadoop," International Conference on Education, Management, Commerce and Society(EMCS), pp. 39-43, 2015.

[21] Fang Zhou, Hai Pham, Jianhui Yue, Hao Zou and Weikuan Yu, "SFMapReduce: An Optimized MapReduce Framework for Small Files," Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on, pp. 23-32, August 2015.

Collection @ chosun