JANT 한국항행학회논문지
Journal of Advanced Navigation Technology

응용 및 융합 기술

J. Adv. Navig. Technol. 18(4): 401-408, Aug. 2014

하둡 플랫폼을 이용한 대량의 스몰파일 처리방법

Processing Method of Mass Small File Using Hadoop Platform

김 창 복[1*] · 정 재 필[2]
[1]가천대학교 에너지 IT학과
[2]가천대학교 전자공학과

Chang-Bok Kim[1*] · Jae-Pil Chung[2]

Department of Energy IT, Gachon University, Gyeonggi-do, 461-701, Korea
Department of Electronic Engineering, Gachon University, Gyeonggi-do, 461-701, Korea

[요 약]

하둡(Hadoop)은 맵리듀스(MapReduce) 분산처리 프로그래밍 모델과 HDFS(Hadoop distributed file system) 분산 파일시스템으로 구성된다. 하둡은 빅데이터 처리에 적합한 프레임워크로서, 대량의 스몰파일 처리에 문제점이 있다. 하둡에서 대량의 스몰파일 처리는 하나의 파일마다 매퍼가 생성되며, 파일의 메타정보를 저장하기 위해 많은 메모리가 필요한 문제점이 있다. 본 논문은 하둡 플랫폼에서 다양한 방법으로 대량의 스몰파일 처리방법을 비교 검토하였다. 일반 압축은 데이터의 크기와 상관없이 하나의 매퍼로 처리해야 하기 때문에, 하둡 처리 포맷으로 적절하지 않다. 시퀀스 와 하둡 아카이브 파일의 처리는 스몰파일을 압축 및 병합을 통해 네임노드의 메모리 문제가 제거되었다. 하둡 아카이브 파일은 스몰파일의 병합시간이 시퀀스 파일보다 빠른 속도를 보였다. CombineFileInputFormat 클래스를 이용한 처리는 병합과정이 필요 없으며, 빅데이터 처리방법과 유사한 속도를 보였다.

[Abstract]

Hadoop is composed with MapReduce programming model for distributed processing and HDFS distributed file system. Hadoop is suitable framework for big data processing, but processing of mass small files have many problems. The processing of mass small file in hadoop have problems to created one mapper per one file, and it have problems to needed many memory for store of meta information of file. This paper have comparison evaluation processing method of mass small file with various method in hadoop platform. The processing of general compression format is inadequate because of processing by one mapper regardless of data size. The processing of sequence and hadoop archive file is removed memory problem of namenode by compress and combine of small file. Hadoop archive file is faster then sequence file about combine time of small file. The processing using CombineFileInputFormat class is needed not combine of small file, and it have similar speed big data processing method.

Key word : Big data, CombineFileInputFormat, Hadoop distributed file system, MapReduce, Small file.

http://dx.doi.org/10.12673/jant.2014.18.4.401

Received 22 July 2014; **Revised** 28 August 2014
Accepted (Publication) 23 August 2014(30 August 2014)

***Corresponding Author; Chang-Bok Kim**

Tel: +82-32-820-4294
E-mail: cbkim@gachon.ac.kr

Ⅰ. 서 론

빅데이터(big data)는 기존 플랫폼으로 저장, 관리, 처리 및 분석하기 어려운 정형화 및 비정형화 구조의 매우 큰 데이터를 의미한다[1]. 최근 빅데이터에 대한 대처에 다양한 의견이 제시되고 있으며, 그 중심에는 하둡(Hadoop)과 클라우드(cloud)가 있다. 하둡은 빅데이터 분산저장 및 분산처리를 위한 핵심 플랫폼으로, 분산저장을 위한 HDFS(hadoop distributed file system)와 분산처리를 위한 맵리듀스(MapReduce) 프로그래밍 모델로 구성된다[2]-[4].

하둡은 빅데이터를 기본적으로 64 MB 단위로 분할하여, 맵리듀스로 분산처리 한다. 그러나 정보처리 중에 발생하는 데이터는 로그 데이터와 같이 대량의 스몰파일(small file)들이 생성되는 경우가 빈번하다. 하둡은 빅데이터 처리 플랫폼이기 때문에, 대량의 스몰파일의 처리에 문제점이 있다. 하둡에서 스몰파일 처리의 핵심 문제점은 첫째, 하나의 파일이 64 MB 블록보다 매우 작더라도, 파일마다 매퍼가 생성되기 때문에, 많은 매퍼의 생성에 의한 잡 스케줄링과 JVM 실행시간으로 많은 시간이 소요된다. 둘째, HDFS에 저장되는 파일은 파일의 메타정보들을 네임노드(Namenode)에 저장하기 때문에, 파일 개수가 많아지면, 네임노드에 많은 메모리가 필요하게 된다[5]-[6].

본 논문은 대량의 스몰파일을 다양한 방법으로 처리하여 결과를 비교 평가하였다. 비교 평가 시스템은 하둡 의사 분산모드(pseudo distributed mode)와 완전 분산모드(fully distributed mode)를 사용하였다. 완전 분산모드는 총 6대의 컴퓨터로 구성하였으며, 네임노드와 2차 네임노드 그리고 5대의 데이터 노드로 구성하였다. 본 논문에서 사용한 데이터는 미국 규격협회(American standards association)에서 공개한 1987년도부터 2008년도까지의 미국 항공편 운항 통계 데이터이다. 또한, 비교평가를 위해 연도, 월별로 항공기 출발 지연건수를 계산하는 맵리듀스 프로그램을 이용하였다[7]. 본 논문의 스몰데이터는 원 데이터를 리눅스의 split 명령을 이용하여, 6 MB 단위로 분할하여 사용하였다. 비교 평가방법은 원 데이터 처리, 스몰파일 처리, Gzip 압축 처리, 시퀀스(sequence) 및 하둡 아카이브(archive) 파일 포맷으로 처리, 스몰파일 처리 클래스를 이용한 처리 등을 사용하였다.

본 논문은 2장에서 관련연구로서 하둡 개요와 하둡 관련 파일 포맷에 대해서 서술하였으며, 3장에서 스몰파일의 문제점 및 평가방법을 제시하였다. 또한, 4장에서 스몰파일 처리 비교평가를 하며, 5장에서 결론을 맺는다.

Ⅱ. 하둡 개요와 파일 포맷

2-1 하둡 개요

하둡은 마스터-슬레이브(master- slave) 구조로서 네임노드,

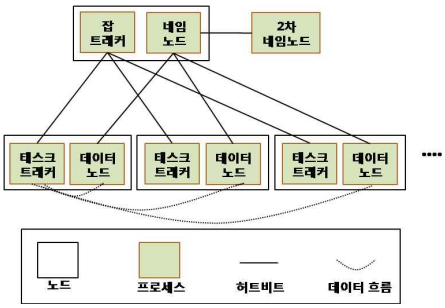


그림 1. 하둡 플랫폼
Fig. 1. Hadoop platform.

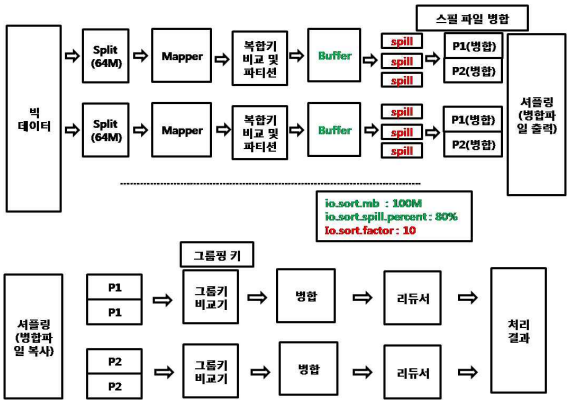


그림 2. 맵리듀스 처리과정
Fig. 2. Processing of mapreduce.

2차 네임노드(secondary name node), 잡트래커(job tracker) 등의 마스터와 데이터 노드(data node), 태스크트래커(task tracker) 등의 슬레이브로 구성된다. 그림 1에 하둡 플랫폼에 대해서 나타냈다[7].

맵리듀스는 빅데이터를 블록으로 분할(split), 매퍼에 레코드 입력, 출력결과 정렬(sort), 파티션(partition), 스필(spill), 병합(combine), 셔플링(shuffling) 등의 단계를 거쳐 리듀서에 전송한다. 리듀서는 전송된 데이터를 동일키를 기준으로 정렬, 스필, 병합, 리듀서에 레코드 입력, 출력 결과를 HDFS에 저장 등의 단계를 거친다[7]-[8]. 그림 2에 맵리듀스의 처리 처리과정에 대해서 나타냈다.

2-2 하둡 파일 포맷

하둡은 정형 데이터, 반정형 데이터, 비정형 데이터 뿐 아니라 다양한 압축포맷을 가진 압축파일을 처리할 수 있다. 하둡의 압축포맷은 Gzip, BZip2, 스내피(snappy) 등이 있다. Gzip(GNU zip)은 리눅스 오픈소스 압축 프로그램이다. Bzip2는 좀 더 효율적인 압축을 제공하며, 스내피는 압축률을 높이기보다는 적당한 압축률로 빠르게 압축을 수행하는 것을 목표로 한다[9].

시퀀스(sequence)파일은 이진 키-밸류(key-value) 쌍의 데이터 구조를 가진 파일로서, MapFile, SetFile, ArrayFile,

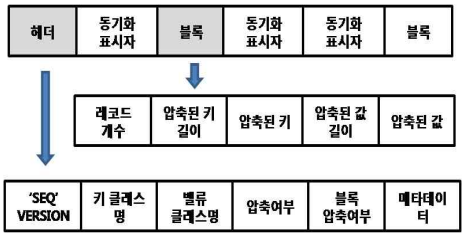


그림 3. 시퀀스 파일 구조(블록 압축)
Fig. 3. Structure of sequence file(block compression).

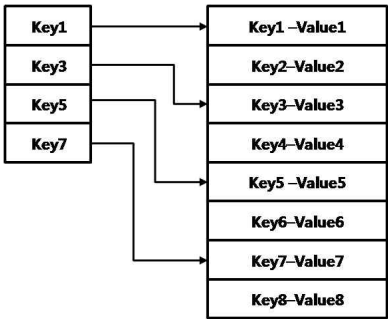


그림 4. 맵 파일 구조
Fig. 4. Structure of map file.

BloomMapFile 등의 파일을 위한 기초 자료체계이다. 시퀀스 파일은 비 압축, 레코드 압축, 블록 압축 등 3가지 포맷을 가진다. 특히, 블록 압축 포맷은 강력한 압축기능이 있으며, Gzip 또는 스내피 압축 등을 이용한다. 그림 3에 블록 압축을 사용할 때의 시퀀스 파일 구조에 대해서 나타냈다[10].

맵(Map) 파일은 인덱스 및 데이터 등 두 개의 시퀀스 파일이 있다. 인덱스 파일은 키와 레코드의 시작 바이트 위치를 가지고 있으며, 데이터 파일은 모든 키-밸류 쌍의 레코드를 가지고 있다. 그림 4에 맵 파일 구조에 대해서 나타냈다[10].

하둡 아카이브(archive)는 메타정보가 저장된 _masterindex 및 _index 파일과 실제 데이터가 저장된 part 파일로 구성된다. _index파일은 아카이브에 포함된 파일들의 이름과 해당 파일이 part 파일 어디에 있는지를 나타내는 위치가 저장되어 있다. 그림 5에 하둡 아카이브의 구조에 대해서 나타냈다[11].

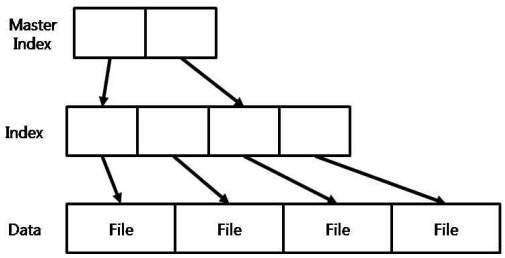


그림 5. 아카이브의 구조
Fig. 5. Structure of Archive

Ⅲ. 스몰파일의 문제점 및 평가 방법

하둡은 대량의 스몰파일도 처리시간을 단축시킬 수 있으나 다음과 같은 문제점이 있다.

첫째, 기본적으로 매퍼의 입력은 FileInputFormat이기 때문에, 하나의 파일에 대해서 하나의 매퍼가 생성된다. 즉, 100만개의 스몰파일을 처리하기 위해서는 100만개의 매퍼가 필요하다. 이 경우 잡트래커가 잡을 스케줄링 하는 시간에 문제가 발생한다. 또한, 각 노드는 기본적으로 2개씩 매퍼가 생성되어 처리되기 때문에, 각 노드마다 매퍼의 생성과 종료에 많은 시간을 할애하게 된다.

둘째, 파일의 메타정보는 네임노드에서 관리되며, 각 파일의 메타정보를 저장하기 위해 150 Byte 정도의 메모리가 필요하다. 만약 100만개의 파일은 30 GB의 메모리의 필요하게 된다. 따라서 대량의 스몰파일을 저장한다면, 메모리 부족 문제가 발생할 수 있다.

셋째, 많은 스몰파일은 검색과 파일 읽기 성능이 떨어진다. 스몰파일이 많다는 것은 많은 수의 데이터노드와 네임노드를 요구하기 때문에, 검색과 읽기에 대한 처리 시간에 문제가 발생한다.

이와 같이 대량의 스몰파일을 효율적으로 처리하기 위해서는 매퍼의 생성 및 처리시간 뿐만 아니라 스몰파일의 메타정보를 저장하는 네임노드의 메모리문제를 고려해야 한다. 대량의 스몰파일을 하둡 맵리듀스로 처리하는 방법은 다음과 같다.

첫 번째 방법은 덧붙이기 방법으로 스몰파일을 하나의 파일에 직접 덧붙이기 하여, 하나의 빅 파일로 가공하여 처리하는 것이다. 이 방법은 처리할 스몰파일이 10만개라면, 10만 번의 덧붙이기를 해야 하기 때문에 의미 없는 방법이다.

두 번째 방법은 스몰파일들을 리눅스 표준 압축포맷인 Gzip 압축 포맷으로 압축하여 처리하는 것이다.

세 번째 방법은 Gzip, bzip2, 스내피 등의 압축 유틸리티를 이용하여, 스몰파일이 저장된 디렉토리를 하둡이 처리할 수 있는 시퀀스 또는 맵 포맷의 빅파일로 가공한 후 처리하는 것이다. 스몰파일이 저장된 디렉토리 압축은 저장된 파일의 용량을 줄여주기 때문에, 네트워크 또는 디스크 간의 전송속도를 높여주는 이점이 있다.

네 번째 방법은 스몰파일의 디렉토리를 하둡 아카이브로 생성한 후, 처리하는 것이다.

다섯 번째 방법은 하나의 매퍼에 여러 개의 스몰파일을 입력하여 처리하는 것이다. CombineFileInputFormat 클래스는 추상 클래스로서 하나의 매퍼에 설정된 블록 크기만큼 스몰파일이 입력될 수 있도록 설계할 수 있다. 즉, 64 MB 블록으로 설정하였다면, 하나의 매퍼에 64 MB 블록 크기만큼 여러 개의 스몰파일들을 입력하여 처리하게 된다. CombineFileInputFormat 클래스는 파일 처리 로직을 가진 RecordReader를 상속받아 클래스를 재설계해야 하며, 파일의 라인과 파일이름을 키-밸류로 하는 WritableComparable을 구현해야 한다. 표 1에 스몰파일 처리 방법에 대해서 나타냈다.

표 1. 스몰파일 처리 방법
Table 1. Small file processing method.

방법	기능
덧붙이기 (append)	스몰파일을 하나의 파일에 추가하여 하나의 빅 파일로 가공한 후 처리
일반 압축포맷	스몰파일을 리눅스 표준 압축 포맷인 Gzip 압축 포맷으로 가공하여 처리
시퀀스 파일	스몰파일의 디렉토리를 Gzip 또는 스내피 압축 포맷을 이용하여 시퀀스 파일을 생성한 후 처리
하둡 아카이브	스몰파일의 디렉토리를 하둡 아카이브를 생성한 후 처리
CombineFileInputFormat클래스	CombineFileInputFormat 클래스를 사용하여 여러개의 스몰파일을 하나의 맵에 입력하여 처리하는 방법

Ⅳ. 스몰파일 처리결과 분석

본 논문은 각 스몰파일 처리방법에 대해서 비교평가하기 위해 연도, 월별로 항공기 출발 지연건수를 출력하는 맵리듀스 프로그램을 사용하였다. 또한, 시퀀스 파일 포맷을 처리하기 위한 드라이버 클래스와 CombineFileInputFormat 클래스를 사용하기 위한 맵리듀스 프로그램을 구현하여 사용하였다. 표 2에 본 논문의 실험 환경에 대해서 나타냈다.

본 논문은 미국 규격협회에서 공개한 1987년도부터 2008년도까지의 항공편 운항 통계 데이터를 사용하였다. 본 데이터는 CSV(comma eparated values) 파일이며, 29개의 컬럼으로 구성되어 있다. 본 논문은 대량의 스몰파일 처리를 비교 평가하기 위해 하둡 의사 분산모드와 완전 분산 모드를 모두 사용하여 실험하였다. 의사 분산모드는 2008년도 항공 운항 통계데이터만을 사용하였다. 2008.csv는 657 MB의 데이터로서 동일한 크기의 많은 스몰파일을 생성하기 위해 리눅스 split 명령으로, 6 MB의 단위로 분할하여, 110개의 스몰파일을 생성하였다. 그림 6에 의사 분산모드의 스몰파일 생성 결과에 대해서 나타냈다.

표 2. 실험 환경
Table 2. Experiment environment

하드웨어	Pentium(R) Dual-Core CPU, 2GHz, 2GB
시스템 플랫폼	ubuntu 12.04 LTS
하둡 버전	hadoop-1.2.1L
JDK	java-1.7.1_51
개발환경	이클립스

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
xaa	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xab	file	6 MB	3	64 MB	2014-06-18 04:00	rw-r--r--	root	supergroup
xac	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xad	file	6 MB	3	64 MB	2014-06-18 04:00	rw-r--r--	root	supergroup
xae	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xaf	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xag	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xah	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xai	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xaj	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xak	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup
xal	file	6 MB	3	64 MB	2014-06-18 04:00	rw-r--r--	root	supergroup
xam	file	6 MB	3	64 MB	2014-06-18 04:01	rw-r--r--	root	supergroup

그림 6. 의사 분산 모드의 스몰파일
Fig. 6. Small file of pseudo distributed mode.

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
split00	dir				2014-07-07 13:42	rwxr-xr-x	root	supergroup
split01	dir				2014-07-07 13:25	rwxr-xr-x	root	supergroup
split02	dir				2014-07-07 13:39	rwxr-xr-x	root	supergroup
split03	dir				2014-07-07 13:45	rwxr-xr-x	root	supergroup
split04	dir				2014-07-07 13:29	rwxr-xr-x	root	supergroup
split05	dir				2014-07-07 13:30	rwxr-xr-x	root	supergroup
split06	dir				2014-07-07 13:31	rwxr-xr-x	root	supergroup

(a) 스몰파일 저장 디렉토리

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
xaa	file	6 MB	1	64 MB	2014-07-07 13:41	rw-r--r--	root	supergroup
xab	file	6 MB	1	64 MB	2014-07-07 13:40	rw-r--r--	root	supergroup
xac	file	6 MB	1	64 MB	2014-07-07 13:41	rw-r--r--	root	supergroup
xad	file	6 MB	1	64 MB	2014-07-07 13:41	rw-r--r--	root	supergroup
xae	file	6 MB	1	64 MB	2014-07-07 13:41	rw-r--r--	root	supergroup
xaf	file	6 MB	1	64 MB	2014-07-07 13:41	rw-r--r--	root	supergroup

(b) 디렉토리에 저장된 스몰파일

그림 7. 완전 분산 모드의 스몰파일
Fig. 7. Small file of fully distributed mode.

완전 분산모드는 총 6대의 컴퓨터로 구성하였으며, 네임노드, 2차 네임노드 그리고 5대의 데이터 노드로 구성하였다. 완전 분산모드는 1987.csv-2008.csv까지의 총 11 GB의 항공 운항 통계데이터를 사용하였으며, 각 년도의 데이터를 6 MB의 단위로 분할하여, 스몰파일을 생성하였다. 그림 7에 완전 분산모드의 스몰파일 생성결과에 대해서 나타냈다.

그림 7(a)는 1987.csv-2008.csv까지의 20개의 항공운항데이터에 대한 디렉토리이며, 그림 7(b)는 각 디렉토리에 6 MB 단위로 분할되어 저장된 스몰파일로서 총 1922개의 스몰파일이 저장되어 있다. 그림 8에 2008.csv의 원 데이터, 원 데이터 분할 스몰파일, 원 데이터 Gzip 압축 파일 등을 처리한 결과에 대해서 나타냈다.

그림 8(a)는 원 데이터인 2008.csv(654 MB)에 대한 처리 결과이며, 11개의 매퍼로 2분 9초가 소요되었다. 그림 8(b)는 스몰파일에 대한 처리 결과이며, 파일의 수와 동일한 110개의 매퍼로 5분 52초가 소요되었다. 그림 8(c)는 Gzip으로 압축(166 MB)하여 처리한 결과이며, 1개의 매퍼로, 1분 9초가 소요되었다. 그림 8(a)의 결과 보다 처리시간이 단축된 것은 압축 결과 데이터의 크기에 의한 것이다. 여기서 중요한 것은 Gzip 파일은 압축데이터의 크기에 상관없이 매퍼가 1개 생성되어 처리한다는 것이다. 즉, 수 TB 데이터를 하나의 매퍼로 처리한다면 많은 시간이 소요될 것이다. 그림 9에 시퀀스 파일 생성 및 처리결과에 대해서 나타냈다.

Started at: Wed Jul 02 21:51:21 KST 2014
Finished at: Wed Jul 02 21:53:31 KST 2014
Finished in: 2mins, 9sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	11	0	0	11	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(a) 원 데이터 처리

Started at: Thu Jul 03 01:51:33 KST 2014
Finished at: Thu Jul 03 01:57:26 KST 2014
Finished in: 5mins, 52sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	110	0	0	110	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(b) 스몰파일 처리

Started at: Thu Jul 03 01:44:35 KST 2014
Finished at: Thu Jul 03 01:45:44 KST 2014
Finished in: 1mins, 9sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(c) Gzip 압축 처리

그림 8. 빅데이터 처리
Fig. 8. Processing of big data.

그림 9(a)는 HDFS에 저장된 스몰파일을 시퀀스 파일로 생성한 결과이며, 110개의 매퍼로, 9분 48초가 소요되었다. 시퀀스 파일 생성 결과 226 MB로 압축되었다. 그림 9(b)는 시퀀스 파일을 처리한 결과로서, 4개의 매퍼로, 1분 41초가 소요되었다. 시퀀스 파일 처리시간은 큰 문제점이 없으나, 시퀀스 파일 생성에 매우 많은 시간이 소요되는 문제점이 있었다. 그림 10에 하둡 아카이브 파일 생성 및 처리 결과에 대해서 나타냈다.

Started at: Wed Jul 02 23:23:46 KST 2014
Finished at: Wed Jul 02 23:33:34 KST 2014
Finished in: 9mins, 48sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	110	0	0	110	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(a) 시퀀스 파일 생성

Started at: Wed Jul 02 23:50:44 KST 2014
Finished at: Wed Jul 02 23:52:26 KST 2014
Finished in: 1mins, 41sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	4	0	0	4	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(b) 시퀀스 파일 처리

그림 9. 시퀀스 파일 생성 및 처리(Gzip)
Fig. 9. Creation and processing of sequence file(Gzip).

Started at: Thu Jul 03 09:12:15 KST 2014
Finished at: Thu Jul 03 09:13:17 KST 2014
Finished in: 1mins, 2sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1	0	0	1	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(a) 하둡 아카이브 생성

Started at: Thu Jul 03 09:21:08 KST 2014
Finished at: Thu Jul 03 09:22:37 KST 2014
Finished in: 1mins, 28sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	2	0	0	2	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(b) 하둡 아카이브 처리_1

Started at: Thu Jul 03 09:26:11 KST 2014
Finished at: Thu Jul 03 09:32:26 KST 2014
Finished in: 6mins, 14sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	110	0	0	110	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(c) 하둡 아카이브 처리_2

그림 10. 하둡 아카이브 생성 및 처리
Fig. 10. Creation and processing of Hadoop archive.

그림 10(a)는 하둡 아카이브 생성 결과이며, 1개의 매퍼로 1분 2초가 소요되었다. 그림 10(b)와 그림 10(c)는 아카이브를 처리한 결과이다. 그림 10(b)는 병합된 아카이브 파일을 사용한 것이며, 2개의 매퍼로, 1분 28초가 소요되었다. 그림 10(c)는 아카이브 디렉토리를 사용한 것이며, 110개의 매퍼로, 6분 14초가 소요되었다. 그림 10(c)는 많은 매퍼가 생성되어, 처리시간에 문제가 있었다. 그림 11에 CombineFileInputFormat 클래스를 이용한 처리 결과에 대해서 나타냈다.

그림 11(a)는 하나의 맵에 64 MB의 파일을 입력하여 처리한 결과로서, 11개의 매퍼로, 2분 10초가 소요되었다.

Started at: Thu Jul 03 01:02:53 KST 2014
Finished at: Thu Jul 03 01:05:04 KST 2014
Finished in: 2mins, 10sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	10	0	0	10	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

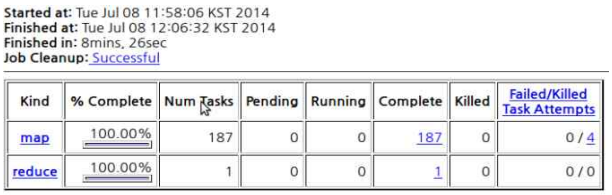
(a) 64 MB 블록

Started at: Thu Jul 03 01:08:46 KST 2014
Finished at: Thu Jul 03 01:10:19 KST 2014
Finished in: 1mins, 33sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	5	0	0	5	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(b) 128 MB 블록

그림 11. CombineFileInputFormat 클래스를 이용한 처리
Fig. 11. Processing using CombineFileInputFormat class.


Started at: Tue Jul 08 11:58:06 KST 2014
Finished at: Tue Jul 08 12:06:32 KST 2014
Finished in: 8mins, 26sec
Job Cleanup: Successful

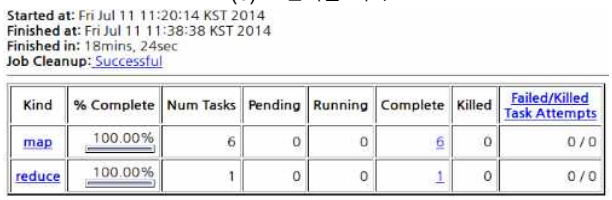
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	187	0	0	187	0	0 / 4
reduce	100.00%	1	0	0	1	0	0 / 0

(a) 11 GB 빅데이터 처리


Started at: Tue Jul 08 12:14:43 KST 2014
Finished at: Tue Jul 08 12:26:07 KST 2014
Finished in: 11mins, 24sec
Job Cleanup: Successful

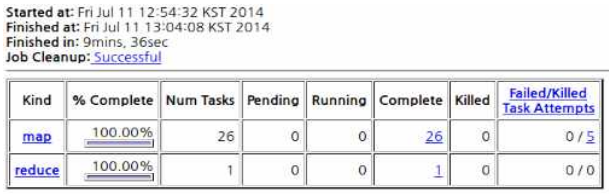
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	1922	0	0	1922	0	0 / 4
reduce	100.00%	1	0	0	1	0	0 / 0

(b) 스몰파일 처리


Started at: Fri Jul 11 11:20:14 KST 2014
Finished at: Fri Jul 11 11:38:38 KST 2014
Finished in: 18mins, 24sec
Job Cleanup: Successful

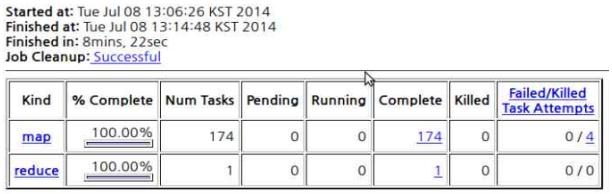
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	6	0	0	6	0	0 / 0
reduce	100.00%	1	0	0	1	0	0 / 0

(c) 하둡 아카이브 생성


Started at: Fri Jul 11 12:54:32 KST 2014
Finished at: Fri Jul 11 13:04:08 KST 2014
Finished in: 9mins, 36sec
Job Cleanup: Successful

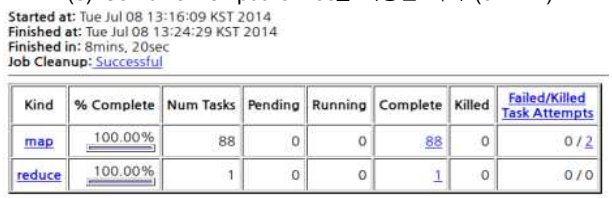
Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	26	0	0	26	0	0 / 5
reduce	100.00%	1	0	0	1	0	0 / 0

(d) 하둡 아카이브 처리


Started at: Tue Jul 08 13:06:26 KST 2014
Finished at: Tue Jul 08 13:14:48 KST 2014
Finished in: 8mins, 22sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	174	0	0	174	0	0 / 4
reduce	100.00%	1	0	0	1	0	0 / 0

(e) CombineFileInputFormat을 이용한 처리 (64 MB)


Started at: Tue Jul 08 13:16:09 KST 2014
Finished at: Tue Jul 08 13:24:29 KST 2014
Finished in: 8mins, 20sec
Job Cleanup: Successful

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	88	0	0	88	0	0 / 2
reduce	100.00%	1	0	0	1	0	0 / 0

(f) CombineFileInputFormat을 이용한 처리 (128 MB)

그림 12. 완전 분산 모드 처리
Fig. 12. Processing of fully distributed mode.

그림 11(b)는 128 MB 블록으로 처리한 결과로서, 5개의 매퍼로, 1분 33초가 소요되었다. 이와 같이 64 MB 보다 128 MB의 블록 단위로 처리하는 것이 처리속도가 좋았다. 그림 12에 완전 분산모드의 결과에 대해서 나타냈다.

그림 12(a)는 대략 11 GB를 처리한 결과로서, 187개의 매퍼로, 8분 26초가 소요되었다.그림 12(b)는 11G를 6 MB씩 분할한 스몰파일을 처리한 결과로서, 스몰파일과 동일한 수인 1922개의 매퍼로, 11분 24초가 소요되었다. 그림 12(c)는 아카이브 파일을 생성한 것으로, 6개의 매퍼로, 18분 24초가 소요되었다. 그림 12(d)는 아카이브 파일을 처리한 것으로, 26개의 매퍼로, 9분 36초가 소요되었다. 그림 12(e)는 CombineFileInputFormat을 이용하여 64 MB 블록단위로 여러개의 스몰파일을 입력하여 처리한 결과이며, 174개의 매퍼로, 8분 22초가 소요되었다. 그림 12(f)는 CombineFileInputFormat을 이용하여 128 MB 블록단위로 여러개의 파일을 입력하여 처리한 결과이며, 88개의 매퍼로, 8분 20초가 소요되었다. 그림 12(a)와 그림 12(e)에서 생성된 매퍼의 수가 다른 것은 그림 12(a)에서 20개의 데이터를 64 MB로 나눈 결과, 각 빅데이터마다 마지막 부분의 블록이 생성되며, 그림 12(e)는 생성된 매퍼에 스몰파일이 64 MB단위로 입력되어 처리되기 때문이다. 즉, CombineFileInputFormat을 이용한 처리는 매퍼의 수를 줄일 수 있는 장점이 있다는 것을 알 수 있다. 표 3에 평가내용과 처리결과에 대해서 나타냈다.

하둡 플랫폼에서 스몰파일을 처리하는 방법에 대한 비교평가 결과는 다음과 같다.

표 3. 평가내용과 처리결과
Table 3. Evaluation contents and processing result.

모드	평가 내용	처리시간	매퍼	재가공 여부
의사분산 모드 (657 MB)	빅데이터 처리	2분 9초	11	X
	스몰파일 처리	5분 52초	110	X
	Gzip 압축 처리	1분 9초	1	X
	시퀀스 파일 생성 (Gzip 압축)	9분 48초	110	O
	시퀀스 파일 처리 (Gzip)	1분 41초	4	
	시퀀스 파일 생성 (Snappy압축)	7분 30초	110	O
	시퀀스 파일처리 (Snappy)	1분 48초	7	
	하둡 아카이브 생성	1분 2초	1	O
	하둡 아카이브 처리 1	1분 28초	2	
	하둡 아카이브 처리 2	6분 14초	110	
	CombineFileInput Format (64 MB)	2분 10초	10	X
	CombineFileInput Format(128 MB)	1분 33초	5	
완전분산 모드 (11 GB)	빅데이터 처리	8분 26초	187	X
	스몰파일 처리	11분 24초	1922	
	하둡 아카이브 생성	18분 24초	6	O
	하둡 아카이브 처리 1	9분36초	26	
	CombineFileInput Format(64 MB)	8분 22초	174	X
	CombineFileInput Format(128 MB)	8분 20초	88	

첫째, 스몰파일이 저장된 디렉토리를 실행할 경우, 스몰파일의 문제점인 파일 수만큼의 맵 실행에 따른 많은 처리시간과 네임노드의 메모리 문제점이 발생하였다. 그러나 JVM 재사용을 통해 처리시간을 보상받을 수 있다.

둘째, Gzip 압축은 헤더가 없기 때문에, 블록의 시작위치를 찾지 못해 블록으로 나눌 수 없다. 따라서, 파일의 크기와 상관없이 1개의 매퍼로 처리되며, 하둡의 장점인 병렬처리를 할 수 없기 때문에, 맵리듀스 파일포맷으로는 적합하지 않다.

셋째, 시퀀스 파일 처리는 HDFS에 저장된 스몰파일을 시퀀스 파일로 변환하고, 변환 결과 스몰파일을 제거하는 등의 불필요한 작업이 필요하다. 더욱이 스몰파일을 시퀀스 파일 생성하는 과정에 파일 수만큼 매퍼가 생성되어 실행되며, 처리시간이 가장 많이 소요되었다.

넷째, 하둡 아카이브 생성은 시퀀스 파일 생성 시간 보다는 짧은 시간이 소요되었다. 또한, 아카이브 디렉토리의 처리 보다는 병합된 파일만을 사용한 것이 처리시간이 적게 소요되었다.

다섯째, CombineFileInputFormat 클래스를 이용한 처리는 시퀀스 파일이나 하둡 아카이브와는 달리 전처리 과정이 필요 없으며, 빅데이터 처리방법과 유사한 속도를 보였다. 그러나 네임노드의 메모리 문제점은 여전히 남아있다.

Ⅴ. 결 론

본 논문은 빅데이터 플랫폼인 하둡에서 스몰파일의 문제점을 제시하고, 다양한 방법으로 스몰파일을 처리하여, 그 결과를 비교 분석하였다. 대량의 스몰 파일은 네임노드 메모리 문제와 처리시간에 문제가 있다. 네임노드 메모리 문제는 시퀀스 파일과 하둡 아카이브로 변환하여 해결할 수 있으나, 변환단계가 추가되고 변환시간에 문제가 있다. 또한, CombineFileInputFormat 클래스를 이용한 처리는 처리시간에는 문제가 없으나 네임노드 문제는 여전히 발생한다. 스몰파일 처리 시간은 매퍼 및 리듀서의 버퍼 크기 조절, 스필파일 병합갯수 조절, JVM 재사용, 압축 등 다양한 튜닝을 통해 처리시간을 보완할 수 있다. 그러나 네임노드 문제를 해결하기 위해서는 반드시 빅 파일로 변경해야 하는 문제점이 있다. 따라서, 대량의 스몰파일은 스몰파일의 크기 및 데이터 종류에 따라 적절한 방법으로 처리해야 할 것이다. 본 논문은 대량의 스몰파일을 처리하는 다양한 방법으로 비교 분석하였으나, 네임노드 문제점과 처리시간을 동시에 해결할 수 있는 방법에 대해서 제시하지 못한 한계점이 있다. 스몰파일 문제는 하둡 플랫폼의 문제이며, 특히, 로컬 파일 시스템의 스몰파일들을 HDFS에 저장할 때, 네임노드의 메모리 문제를 해결할 수 있는 방법에 대해서 연구가 지속적으로 진행되어야 할 것이다.

김 창 복 (Chang-Bok Kim)

1986년 2월 : 단국대학교 전자공학과(공학사)
1989년 2월 : 단국대학교 전자공학과(공학석사)
2008년 2월 : 인천대학교 컴퓨터 공학과(공학박사)
1994년 ~ 현재 : 가천대학교 IT대학 에너지 IT학과 교수
관심분야 : 인터넷보안, 클라우드 컴퓨팅, 분산처리시스템

참고문헌

[1] C. W. An, and S. K. Hwang, “Big data technologies and main issues,” *Journal of Korean Institute of Information Scientists and Engineers*, Vol. 30, No. 6, pp.10-17, Jun. 2012.

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

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on IEEE,* Las Vegas: NV, pp. 1-10, 2010.

[4] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, Vol. 51, Issue 1, pp. 107-113, Jan. 2008.

[5] B. G. Gu, “FiVE: File Virtual Expanding technique to efficiently process small data on Hadoop,” *Journal of Korean Institute of Information Technology,* Vol 10, No.10, pp.69-78, Oct. 2012.

[6] G. Mackey, S. Sehrish, and J. Wang. “Improving metadata management for small files in HDFS,” in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on IEEE*, New Orleans: LA, pp. 1-4, Aug. 2009.

[7] J. H. Jung, “Beginning, Hadoop Programming,” *Wikibooks,* Oct. 2012.

[8] C. He, Y. Lu, and D. Swanson, “Matchmaking: a new MapReduce scheduling technique,” *Proceedings of Cloud Computer'11*, pp. 40-47, 2011.

[9] http://code.google.com/p/snappy

[10] http://blog.cloudera.com/blog/2011/01/hadoop-io-sequence-map-set-array-bloommap-files/

[11] http://blog.cloudera.com/blog/2009/02/the-small-files-problem

정 재 필 (Jae-Pil Chung)

1985년 2월 : 단국대학교 전자공학과 (공학사)
1989년 8월 : 단국대학교 대학원 전자공학과 (공학석사)
2000년 8월 : 한국항공대학교 대학원 통신정보공학과 (공학박사)
1989년 8월~1990년 12월 : (주)동양전자통신 중앙연구소 연구원
1990년 12월~1992년 3월 : (주)케피코 기술연구소 연구원
1994년 2월~현재 : 가천대학교 IT대학 전자공학과 교수
[주 관심분야] 무선통신, 통신신호처리, 컴퓨터 네트워크