Journal of Korea Multimedia Society Vol. 18, No. 9, September 2015(pp. 1098-1104)
http://dx.doi.org/10.9717/kmms.2015.18.9.1098

MapReduce에서 Reuse JVM을 이용한 대규모 스몰파일 처리성능 향상 방법에 관한 연구

최철웅[†], 김정인[††], 김판구[†††]

A Study on the Improving Performance of Massively Small File Using the Reuse JVM in MapReduce

Chul Woong Choi[†], Jeong In Kim[††], Pan Koo Kim[†††]

ABSTRACT

With the widespread use of smartphones and IoT (Internet of Things), data are being generated on a large scale, and there is increased for the analysis of such data. Hence, distributed processing systems have gained much attention. Hadoop, which is a distributed processing system, saves the metadata of stored files in name nodes; in this case, the main problems are as follows: the memory becomes insufficient; load occurs because of massive small files; scheduling and file processing time increases because of the increased number of small files. In this paper, we propose a solution to address the increase in processing time because of massive small files, and thus improve the processing performance, using the Reuse JVM method provided by Hadoop. Through environment setting, the Reuse JVM method modifies the JVM produced conventionally for every task, so that multiple tasks are reused sequentially in one JVM. As a final outcome, the Reuse JVM method showed the best processing performance when used together with CombineFileInputFormat.

Key words: Hadoop, SmallFile, Reuse JVM, MapReduce, Distributed Processing

1. 서 론

스토리지 전문기업인 EMC가 발표한 자료에 따르면, 2011년에 전 세계에서 생성된 디지털 데이터양은 1.8제타바이트(Zetabytes)에 이른다[1]. 스마트폰의 보급과 IOT시대가 도래하면서 사람들은 데이터가 폭증하는 빅데이터 시대에 살고 있다[17]. 빅데이터는 기존의 데이터를 처리하던 RDBMS (Relational DataBase Management System)나 DW(Data Warehoust)와 같은 방식으로는 처리할 수 없고 새로운 기술을 필요로 하게 되는데[16], 하둡(Hadoop)이 빅데이터를 저장하고 분석하는 대표적인 기술이다[7]. 하둡은 소스코드가 공개되는 오픈소스이며, 크게 빅데이터를 저장하는 HDFS(Hadoop Distributed File System) 분산파일시스템과 빅데이터를 처리하는 MapReduce 분산처리 시스템 2가지의 컴포넌트

※ Corresponding Author : Pan Koo Kim, Address: (501-759) IT Building, Chuson Unversity, Pilmun-daero 309, Dong-gu, Gwangju, Korea, TEL : +82-62-230-7799, FAX : +82-62-233-6896, E-mail : pkkim@chosun.ac.kr
Receipt date : Aug. 10, 2015, Approval date : Aug. 24, 2015
[†] Dept. of Software Convergence Engineering, Graduate School of Industrial Technology Convergence, Chosun University (E-mail : sentilemon02@gmail.com)
[††] Dept. of Computer Engineering, Graduate School, Chosun University (E-mail : jungingim@gmail.com)
[†††] Dept. of Computer Engineering, Graduate School, Chosun University
※ This research was supported by SW Master's course of hiring contract Program grant funded by the Ministry of Science, ICT and Future Planning (H0116-15-1013) and This work was supported by the Human Resource Training Program for Regional Innovation and Creativity through the Ministry of Education and National Research Foundation of Korea(NRF-2014H1C1A1066494)

로 구성되어 있다[9]. HDFS는 블록 구조의 파일 시스템으로 파일을 저장할 때 일정한 크기의 블록으로 나누어 저장하는데, 대부분 64MB와 128 MB를 사용한다[2,10]. 하둡에서 의미하는 스몰파일이란 기본 블록 크기보다 작은 크기의 파일을 의미한다. 기본적으로 하둡은 크기가 큰 빅파일에 최적화되어있어, 로그데이터와 같은 파일 크기가 작은 스몰파일에는 취약한 단점이 있다[13]. 하둡에서 스몰파일로 인해 발생하는 문제점은 크게 두 가지가 있다. 첫 번째는 HDFS에 파일이 블록으로 나누어 저장될 때 파일의 메타데이터가 네임노드(Namenode)의 메모리에 저장되기 때문에 스몰파일을 저장하게 되면 네임노드의 메모리 부족문제로 데이터노드의 저장 공간이 여유가 있더라도 저장을 하지 못하는 문제가 발생한다[3,6]. 두 번째는 MapReduce는 기본적으로 파일 하나당 매퍼(Mapper)가 생성되어 잡을 처리하는데 대량의 스몰파일마다 각각 매퍼가 하나씩 생성되며 이로 인해 스케줄링 및 처리시간이 증가하는 문제점이 있다[2]. 따라서 위 두 문제점을 극복하고 스몰파일 처리성능 향상을 위해서는 처리 과정에서 불필요한 부분을 줄이고 처리 과정을 간소화하는 방법이 요구된다. 본 논문에서는 하둡에서 제공하는 Reuse JVM 방식을 이용하여 대량의 스몰파일 처리성능 향상 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 하둡과 하둡 스몰파일 문제를 해결하는 기존연구에 관해 서술하였으며, 3장에서는 본 논문에서 제안하는 스몰파일 처리성능 향상방법을 설명한다. 4장에서는 기존 방식과 Reuse JVM을 이용한 방법을 비교평가하였으며, 마지막으로 5장에서는 결론과 향후 연구에 관해 서술한다.

2. 관련연구

2.1 하둡

하둡은 분산병렬처리 시스템으로 마스터(Master)/슬레이브(Slave) 아키텍처로 Fig. 1과 같이 구성된다. 마스터서버는 네임노드(Namenode), 보조 네임노드(Secondary Namenode)와 잡트래커(Jobtracker)로 구성되며, 전체적인 하둡 아키텍처에 단 하나만 존재한다[12,15]. 슬레이브서버는 데이터노드(Datanode)와 태스크트래커(Tasktracker)로 구성되는데 서버를 계속 추가해서 사용 할 수 있으며 최대 4천대까지 사용 할 수 있다.

하둡은 대량의 스몰파일을 처리 할 때 네임노드에 저장되는 다수의 메타데이터로 인한 네임노드 메모리 부족 문제와 잡 실행 시 스몰파일마다 생성되는 매퍼로 인해 스케줄링 및 처리시간이 증가하는 문제

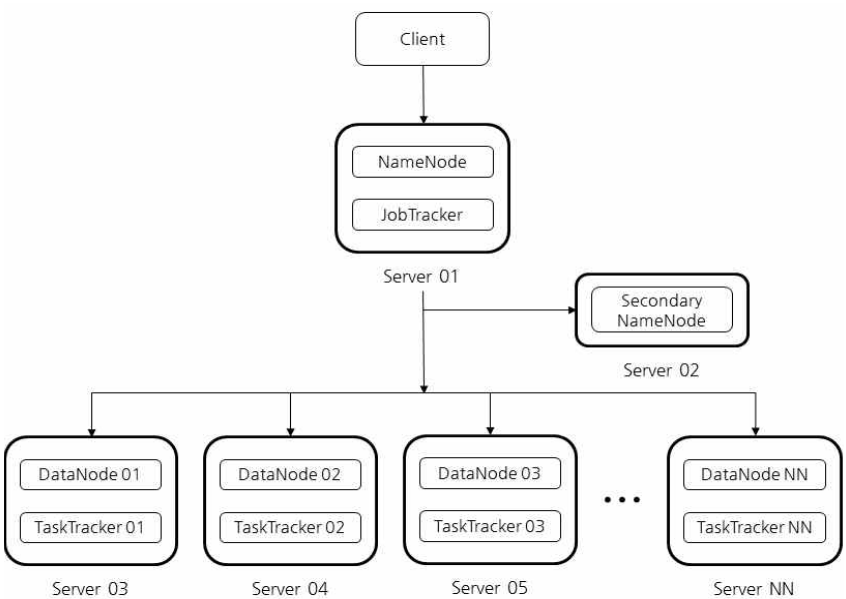

Fig. 1. Hadoop Architecture.

점이 있다[2]. 이와 같은 문제점을 해결하기 위하여 기존 연구는 데이터 병합[3], CombineFileInput-Format[2], 캐시메모리[4,8,11] 크게 세 가지 방법을 이용하고 있다.

데이터 병합 방법은 스몰파일을 하둡 아카이브(Hadoop Archive), Sequence File 등의 파일포맷방식으로 스몰파일을 병합하여 빅파일을 만드는 방법이다[14]. 빅파일로 병합하게 되면, 스몰파일 수가 감소하기 때문에 네임노드에 저장되는 메타데이터의 수도 감소하여, 첫 번째 문제인 네임노드의 메모리 부족 문제를 해결 할 수 있다. 하지만 빅파일로 병합하는데 시간이 많이 소비되어 처리시간이 증가한다는 단점이 지적되었다[3].

CombineFileInputFormat 방법은 잡 처리 시 기존에 스몰파일 하나마다 생성되던 매퍼를 사용자가 임으로 크기를 정하고 스몰파일을 모아 일정한 크기에 도달했을 때 매퍼를 생성하여 처리하는 방법이다. 데이터 병합 방법과는 달리 전처리 과정이 없고 기존 빅데이터 처리속도와 비슷한 성능을 보인다. 하지만 HDFS에 저장되어있는 스몰파일의 수는 감소하지 않기 때문에 네임노드의 메모리 부족 현상은 해결하지 못하는 단점이 있다[2].

캐시메모리 방법은 마스터 서버와 슬레이브 서버 사이에 캐시메모리를 추가하고 자주 사용되는 스몰파일을 적재해 사용하는 방법이다. 캐시메모리를 통해 대량의 스몰파일에 대한 빠른 Read가 가능하지만, 추가로 하둡시스템의 자원을 사용한다는 점과 Hit ratio에 의존적이라는 단점이 있다[4,8,11].

2.2 Reuse JVM

Reuse JVM 방식은 기존에 태스크마다 생성되는 JVM을 환경설정을 통해 수정하여 재사용하는 방식이다. 재사용할 경우 하나의 JVM에서 동시에 태스크가 수행되는 것이 아니라 순차적으로 태스크가 수행된다. 하둡환경설정파일(mapred-site.xml)에서 mapred.job.reuse.jvm.num.tasks 파라미터를 이용해 Reuse JVM을 이용할 수 있다[1].

Table 1은 Reuse JVM을 사용하기 위한 파라미터 설정값에 대해 설명하고 있다. Set Value는 태스크트래커의 각 JVM이 잡 하나에서 수행할 수 있는 최대 태스크 수이다. 예를 들어 사용자가 Set Value를 10으로 설정하였다면, 10개의 태스크가 하나의 JVM에서 순차적으로 실행된다. 본 논문에서는 스몰파일마다 JVM이 생성되는 기존방식과는 달리 Reuse JVM을 이용하여 JVM 생성시간을 감소시키고 처리성능을 향상한다.

Table 1. mapred.job.reuse.jvm.num.tasks parameter set value

Set Value	Explanation
1(Default)	Do not use the Reuse JVM
User Set	Number of tasks to be processed on a single JVM
-1	Unlimited Use

3. Reuse JVM을 이용한 처리성능 향상방법

본 장에서는 하둡에서 대량의 스몰파일 처리성능을 향상하기 위하여 Reuse JVM을 이용한 방법을 제안한다.

기존의 맵리듀스 처리 과정 중 Map 단계는 Fig. 2와 같이 2단계로 진행된다. 1단계는 HDFS에 블록으로 나누어져 저장된 파일을 불러와 잡 실행 시 입력 단위인 Split에 넣는다. 2단계는 Split 하나당 맵 태스크 하나가 생성되며 맵 태스크는 각각 별도의 JVM에서 실행된다. Fig 3은 임으로 Reuse JVM 파라미터 설정값을 3으로 설정 후 잡이 처리되는 과정을 나타내고 있다. 본 논문에서는 위 Fig 3에서 보이는 것과 같이 Reuse JVM을 사용하여 다수의 맵 태

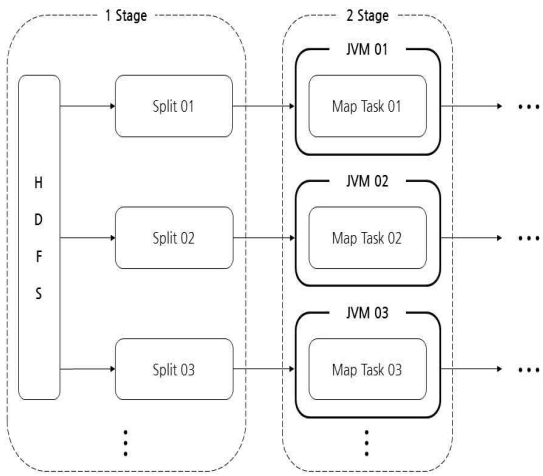

Fig. 2. Basic Processing Methods in MapReduce.

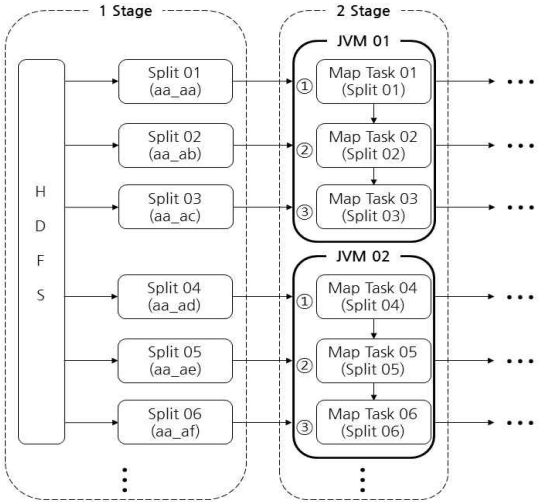

Fig. 3. Reuse JVM Processing Methods in MapReduce (Reuse JVM Set Value: 3).

스크를 하나의 JVM에서 순차적으로 실행되게 한다. 처리 과정은 Fig 2와 동일하게 2단계로 진행된다. 1단계 에서는 HDFS에 저장한 스몰파일을 불러와 각각 Split01, Split02, Split03 등에 넣는다. 2단계는 Split마다 매퍼가 하나씩 생성되며 Reuse JVM 설정값이 3이기 때문에 3개의 매퍼가 하나의 JVM안에서 실행된다. 1단계는 기본방식과 같지만 2단계에서 매퍼마다 생성되던 JVM이 설정값에 따라 하나의 JVM에서 순차적으로 실행된다. 태스크마다 JVM이 실행되는 시간은 1초 내외로 짧은 시간이지만, 스몰파일과 같이 짧은 시간 안에 수행하는 파일이 많은 경우 처리시간에 영향을 줄 수 있다. Reuse JVM을 이용한 방법을 통해 하둡 플랫폼에서 대량의 스몰파일에 대한 한계를 극복하고, 각각의 스몰파일에서 생성되는 JVM 생성시간을 줄여 처리성능을 향상할 수 있을 것으로 예상한다.

4. 실험 결과 및 평가

본 장에서는 기존 방식과 3장에서 제안한 Reuse JVM방식의 처리시간을 비교한다. 또한, 기존 데이터병합방법과 CombineFileInputFormat방법에 Reuse JVM 방법을 더하여 실험한 뒤 처리성능을 평가한다.

실험을 위해 사용한 데이터 셋은 미국 규격 협회(American Standards Association)에서 제공하는 미국 항공편 운항 통계 데이터 셋으로 1987년도부터 2008년도까지 총 22개의 CSV(comma eparated values)파일이며 각 파일은 29개의 칼럼으로 구성되어 있다. 실험에 필요한 스몰파일을 얻기 위해 리눅스(Linux) Split 명령어를 사용해서 기존 22개의 미국 항공편 운항 통계 데이터 셋을 6MB와 3MB로 분할하여 Table 2와 같은 데이터 셋을 생성하였다. 빅파일, 3MB로 분할한 스몰파일, 6MB로 분할한 스몰파일 총 3가지 데이터 셋을 사용하여 실험을 진행한다. 실험환경은 같은 성능의 컴퓨터 6대에 Table 3과 같은 환경의 VMware 가상머신을 생성하여 네임노드 1대, 보조 네임노드 1대, 데이터 노드 5대로 실험 환경을 구축하였다. 논문에서 제안하는 Reuse JVM 방식의 처리시간을 비교 평가하기 위해 맵리듀스 프로그램을 사용하여 항공 출발 지연 건수를 연도별, 월별로 출력하는 실험을 진행하였다.

6MB로 분할한 스몰파일을 데이터 셋으로 Reuse JVM의 다양한 설정값을 실험한 결과 Table 4와 같은 실험결과를 얻었다. Reuse JVM의 설정값을 -1로 사용하였을 때 가장 좋은 처리성능을 보였고, Reuse JVM 설정값 2부터 6까지는 점점 처리속도가 상승했지만, 그 후 Reuse JVM 설정값 7부터 11까지는 04분 40초대로 비슷한 성능을 보이는 것을 알 수 있다. 추후 Reuse JVM을 사용한 모든 실험에는 처리성능이 가장 우수했던 -1로 Reuse JVM의 설정값을 적용하고 동일하게 실험을 진행하였다.

Table 5, 6, 7, 8은 본 논문에서 제안하는 Reuse JVM 방식과 기존 방식의 처리시간을 보여주고 있다. After 실험결과는 기존방식에 Reuse JVM 설정

Table 2. DataSet(11GB)

List	Explanation
BigFiles	22 Files
SmallFiles(3MB)	3835 Files
SmallFiles(6MB)	1922 Files

Table 3. Experiment Environment

List	Explanation
Virtual Machine	VMware 8.0.2
RAM	2GB
HDD	200GB
OS	CentOS 6.6
Hadoop Ver.	Hadoop 1.2.1

Table 4. Result of Experiment(Reuse JVM)

Data Set	Reuse JVM Set Value	Processing Time (Before)	Mapper
SmallFiles (6MB)	-1	04mins, 13sec	1922
	2	06mins, 34sec	
	3	05mins, 34sec	
	4	05mins, 14sec	
	5	04mins, 52sec	
	6	04mins, 43sec	
	7	04mins, 42sec	
	8	04mins, 45sec	
	9	04mins, 41sec	
	10	04mins, 37sec	
	11	04mins, 40sec	

값을 -1로 설정하고 실험을 진행한 결과이다. Table 5의 Before 실험결과는 빅파일의 처리시간은 187개의 매퍼를 사용하여 03분 18초의 시간이 소요되었고 3MB로 분할한 3835개의 스몰파일은 3835개의 매퍼를 사용하여 16분 23초, 6MB로 분할한 1922개의 스몰파일은 1922개의 매퍼를 사용하여 09분 18초의 시간이 소요되었다. 스몰파일은 빅파일에 비해 3배에서 5배 정도 처리시간이 증가하였다. 하둡에서 대량의 스몰파일이 겪는 문제점을 잘 보여주는 실험결과이다. 반면 Table 5의 Reuse JVM을 사용한 After 처리시간은 빅파일 02분 14초, 3MB로 분할한 스몰파일 07분 07초, 6MB로 분할한 스몰파일 04분 13초로 기존방식보다 약 2배정도 처리시간이 단축된 것을 볼 수 있다.

Table 6의 기존 하둡 아카이브방식은 스몰파일을 HAR 파일로 변환하는데 6개의 매퍼를 사용하여 17분 16초의 시간이 소요되었고, 변환된 HAR파일을 처리한 결과 26개의 매퍼를 사용하여 02분 39초의 시간이 소요되었다. HAR방식에 Reuse JVM을 적용한 결과 또한 전과 비슷한 성능을 보였다. HAR 방식의 처리시간은 2분 39초, 2분 30초로 빅파일보다 향상된 처리속도를 보였지만 HAR파일로 변환하는 과정에 많은 시간이 소요되는 문제가 있다. 6MB로 분할한 스몰파일 실험결과 3MB로 분할한 스몰파일 또한 HAR파일로 변환하시는 과정에 많은 시간이 소요될 것으로 예상하여 3MB로 분할한 스몰파일은 실험을 진행하지 않았다.

Table 5. Result of Experiment(Basic)

Data Set	Processing Time (Before)	Processing Time (After)	Mapper
BigFiles	03mins, 18sec	02mins, 14sec	187
SmallFiles(3MB)	16mins, 23sec	07mins, 07sec	3835
SmallFiles(6MB)	09mins, 18sec	04mins, 13sec	1922

Table 6. Result of Experiment(Hadoop Archive)

Data Set		Processing Time (Before)	Processing Time (After)	Mapper
SmallFiles(6MB)	Create	17mins, 16sec	18mins, 06sec	6
	Process	02mins, 39sec	02mins, 30sec	26

Table 7. Result of Experiment(CombineFileInputFormat 64MB)

Data Set	Processing Time (Before)	Processing Time (After)	Mapper
SmallFiles(3MB)	04mins, 25sec	03mins, 15sec	174
SmallFiles(6MB)	03mins, 30sec	03mins, 02sec	

Table 8. Result of Experiment(CombineFileInputFormat 128MB)

Data Set	Processing Time (Before)	Processing Time (After)	Mapper
SmallFiles(3MB)	03mins 50sec	03mins 20sec	89
SmallFiles(6MB)	03mins 16sec	03mins	

Table 7, 8의 CombineFileInputFormat 방식은 각각 스몰파일을 64MB와 128MB 크기로 모아 처리했을 때 3MB 분할한 스몰파일은 각각 174개의 매퍼를 사용하여 04분 25초, 89개의 매퍼를 사용하여 03분 50초의 시간이 소요되었고, 6MB 분할한 스몰파일은 각각 174개의 매퍼를 사용하여 03분 30초, 89개의 매퍼를 사용하여 03분 16초의 시간이 소요되었다. CombineFileInputFormat 방식은 기존 빅파일 처리시간보다 성능이 저하되지만, 스몰파일을 데이터 셋으로 실험한 결과 중 단일로 실험했을 때 가장 우수한 처리성능을 보였다. CombineFileInputFormat 방식에 Reuse JVM을 적용한 결과 3MB 분할한 스몰파일은 각각 174개의 매퍼를 사용하여 03분 15초, 89개의 매퍼를 사용하여 03분 20초의 시간이 소요되었고, 6MB 분할한 스몰파일은 각각 174개의 매퍼를 사용하여 03분 02초, 89개의 매퍼를 사용하여 03분의 시간이 소요되었다. CombineFileInputFormat과 Reuse JVM을 함께 사용한 결과 기존 빅파일 처리시간과 비슷하거나 더욱더 향상된 성능을 보였다. 11GB 크기보다 더욱 큰 대량의 스몰파일을 데이터 셋으로 실험한다면 지금보다 더 우수한 성능을 보일 것으로 예상한다.

5. 결 론

본 논문에서는 하둡에서 대량의 스몰파일이 갖는 문제점을 해결하기 위해 하둡 Reuse JVM을 이용한 대량의 스몰파일 처리성능 향상방법을 제안하였다. 하둡에서 제공하는 Reuse JVM을 사용하여 기존에 스몰파일 하나마다 생성되던 JVM을 설정값을 수정하여 다수의 스몰파일이 하나의 JVM에서 처리되도록 하였다. 3MB로 분할한 3835개의 스몰파일과 6MB로 분할한 1922개의 스몰파일을 데이터 셋으로 Reuse JVM을 이용한 방법과 기존방식을 비교 평가한 결과 단일로 사용했을 때 Reuse JVM을 이용한 방식이 CombineFileInputFormat 다음으로 향상된 성능을 보였으며, 빅파일에도 성능향상의 효과가 있는 것을 알 수 있었다. CombineFileInputFormat 방식에 Reuse JVM을 이용한 방법을 추가하여 처리한 결과 기존 빅파일 처리와 비슷하거나 더욱더 향상된 우수한 성능을 보였다. 하둡아카이브 같은 경우 파일을 병합하여 스몰파일의 개수가 줄어들기 때문에 생성되는 JVM의 수가 많지 않아 Reuse JVM을 사용했을 경우 처리속도가 저하되었다. 본 논문은 대량의 스몰파일 처리성능은 향상했지만 네임노드 메모리 부족문제는 해결하지 못한 한계가 있다. 향후에는 네임노드의 메모리 부족 문제를 해결하는 방법에 관련된 연구와 스몰파일의 개수에 따른 가장 최적화된 Reuse JVM 파라미터 설정값을 찾는 알고리즘에 대해 연구할 예정이다.

REFERENCE

[1] J.H. Jung, *Beginning, Hadoop Programming*, Wikibooks, Gyeonggi-do, Korea, 2012.

[2] C.B. Kim and J.P. Chung, "Processing Method of Mass Small File using Hadoop Platform," *Journal of Advanced Navigation Technology,* Vol. 18, No. 4, pp. 401-408, 2014.

[3] W.J. Yi and S. Park, "A Data Merging Technique based on Clustering for Solving Problems of Massive Small Files in Hadoop with Performance Enhancement of Map/Reduce," *Proceeding of the Winter Conference of the Korean Institute of Information Scientists and Engineers*, pp. 180-182, 2014.

[4] H.K. Oh, K.Y. Kim, J.M. Hwang, J.H. Park, J.T. Lim, K.S. Bok, et al., "A Distributed Cache Management Scheme for Efficient Accesses of Small Files in HDFS," *Journal of the Korea Contents Association,* Vol. 14, No. 11, pp. 28-38, 2014.

[5] W. Tom, *Hadoop : The Definitive Guide*, Hanbit Media, Seoul, Korea, 2013.

[6] K.Y. Han, *Do it! Hadoop with Big Data*, Easys Pub, Seoul, Korea, 2013.

[7] IDG Korea, *Open DB Framework for Big Data: Understanding Hadoop*, IDG Tech Report, pp. 1-9, 2012.

[8] D. Chandrasekar, R. Dakshinamurthy, P.G. Sechakumar, and B. Prabavathy, "A Novel Indexing Scheme for Efficient Handling of Small Files in Hadoop Distributed File System," *Proceeding of International Conference on Computer Communication and Infor-*

matics, pp. 1-8, 2013.

[9] Hadoop 1.2.1 Documentation, http://hadoop.apache.org/ (accessed Aug., 05, 2015)

[10] HDFS File Storage Method, http://blrunner.com/category/Development/Hadoop (accessed Aug., 05, 2015)

[11] B. Dong, J. Qiu, O. Zheng, X.Zhong, J.Li, and Y. Li, "A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop:a Case Study by Power Point Files," *Proceeding of Institute of Electrical and Electronics Engineers International Conference on Services Computing,* pp. 65-72, 2010.

[12] H.W. Kim, S.E. Park, and S.Y. Euh, "The Distributed Encryption Processing System for Large Capacity Personal Information based on MapReduce," *Journal of the Korea Institute of Information and Communication Engineering,* Vol. 18, No. 3, pp. 576-585, 2014.

[13] The Small Files Problem, http://blog.cloudera.com/blog/2009/02/the-small-files-problem/ (accessed Aug., 07, 2015)

[14] Hadoop I/O: Sequence, Map, Set, Array, BloomMap Files, http://blog.cloudera.com/blog/2011/01/hadoop-io-sequence-map-set-array-bloommap-files/ (accessed Aug., 07, 2015)

[15] T.W. Kim, H.J. Chung, J.M. Kim, "The Creation and Placement of VMs and Tasks in Virtualized Hadoop Cluster Environment," Journal of *Korea Multimedia Society*, Vol. 15, No. 12, pp. 1-7, 2012.

[16] U.G. Kim, J.Y. Kim, "Research on Object-Oriented Relational Database Model and its Utilization for Dynamic Geo-spatial Service through Next Generation Ship Navigation System," *IT CoNvergence PRActice (INPRA),* Vol. 1, No. 2, pp. 1-10, 2013.

[17] S. Vidalis, O. Angelopoulou, "Assessing Identity Theft in the Internet of Things," *IT CoNvergence PRActice(INPRA),* Vol. 2, No. 1, pp 15-21, 2014.

최 철 웅

2014년 조선대학교 컴퓨터통계학과 이학사

2014년 현재 조선대학교 산업기술융합대학원 소프트웨어융합공학과 석사과정

관심분야 : 빅데이터 처리 및 분석, 분산처리, 검색엔진

김 정 인

2011년 조선대학교 컴퓨터공학부 공학사

2014년 조선대학교 박사수료

2015년 현재 조선대학교 석박사 연계과정

관심분야 : 정보처리, 소셜네트워크, 시맨틱 웹과 온톨로지

김 판 구

1988년 조선대학교 컴퓨터공학과 공학사 졸업.

1990년 서울대학교 컴퓨터공학과 공학석사 졸업.

1994년 서울대학교 컴퓨터공학과 공학박사 졸업.

1994년~현재 조선대학교 교수

관심분야 : 정보검색, 시맨틱웹, 자연어처리, 빅데이터