Journal of KIIT. Vol. 19, No. 6, pp. 9-19, Jun. 30, 2021. pISSN 1598-8619, eISSN 2093-7571 9
http://dx.doi.org/10.14801/jkiit.2021.19.6.9

Hadoop Parameter 조정을 통한 Hadoop MapReduce 성능 최적화 분석 연구

지경엽*, 권영미**

Hadoop MapReduce Performance Optimization Analysis by Calibrating Hadoop Parameters

Keungyeup Ji*, Youngmi Kwon**

요 약

본 논문은 분산처리 및 가상화 기반으로 대용량 데이터 처리를 하는 하둡(Hadoop) 프레임워크(framework)의 단점인 속도의 비효율성을 해결하기 위한 가장 효율적인 방안으로 하둡 환경의 파라미터(parameter) 조정을 통해서 성능 최적화 환경을 파악하고자 하는 것이 목적이다. 그러기 위해서 하둡 성능에 민감한 영향을 끼치는 Parameter 조정을 통해서 하둡 분산처리를 지원하는 HDFS(Hadoop Distributed File System)와 MapReduce 성능 개선 최적화를 도출하였다. 본 논문 실험을 통해서 대규모 데이터 파일 처리에 특성화된 하둡 프레임워크를 주어진 자원여건에서 최적의 상태에서 활용하기 위해서는 하둡 파라미터 조정을 통해서 적합한 구성 환경을 도출한 후에 데이터 처리(data processing)를 수행하는 것이 효율적이라고 판단된다.

Abstract

The purpose of this paper is to understand the performance optimization environment through parameter adjustment of the Hadoop environment as the most efficient way to solve the inefficiency of speed, which is the disadvantage of the Hadoop framework that processes large amounts of data based on distributed processing and virtualization. To this end, we derived optimization of the HDFS and MapReduce performance improvement that support Hadoop. In this paper, through this test, in order to utilize the Hadoop framework in an optimal state under a given resource condition, it is considered efficient to perform data processing after deriving an appropriate configuration environment through Hadoop parameter adjustment.

Keywords

HDFS, name node, data node, mapreduce, hadoop jar, parameter tuning, experiment-driven

* 충남대학교 전자정보통신공학과 박사과정
- ORCID: https://orcid.org/0000-0002-6610-1053
** 충남대학교 전파정보통신공학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0003-0318-0660

· Received: Apr. 01, 2021, Revised: May 11, 2021, Accepted: May 14, 2021
· Corresponding Author: Youngmi Kwon
Dept. of Radio and Info. Communications Eng., Chungnam National University
Tel.: +82-42-821-6890, Email: ymkwon@cnu.ac.kr

Ⅰ. 서 론

정보시스템의 발전으로 데이터 규모가 급격하게 증가하고 있고, 기존 자원을 활용해서 오픈소스 기반으로 추가 비용을 최소로 하면서 필요시 자원을 확장하는 방식으로 대규모 빅 데이터를 분석 및 군집화와 예측 필요성이 증가하고 있다. 이를 지원하기 위한 플랫폼으로 하둡과 스파크(Spark)가 있다. 데이터가 여러 장소에 분산되어 있고 많은 디스크(Disk) 용량을 차지하는 대형 데이터집합(Dataset)에는 in-memory 방식의 스파크보다 디스크 기반의 하둡이 더 적합한 방식일 수 있다[1].

하둡과 스파크를 간략히 비교한다면 공통점은 대규모 데이터집합을 처리하는 빅 데이터를 위한 프레임워크이고, 차이점은 하둡은 대규모 데이터집합을 여러 서버로 구성된 컴퓨터 클러스터에서 각각 로컬 연산 및 저장, 복제기능을 제공하는 디스크 기반의 프레임워크이고, 스파크는 데이터집합을 메모리 기반으로 처리하기 위한 프레임워크이다. 속도는 스파크가 하둡보다 100배 정도 빠른 것으로 평가되고 있으나[1], 각 Data node에 분산된 데이터를 동시에 처리하기 위해서는 하둡이 더 효율적이다. 스파크는 하둡과는 달리 메모리 기반 방식으로 소프트웨어 플랫폼인 MapReduce를 지원하며, 하둡은 디스크 기반의 MapReduce를 지원하는 구조이다[2].

본 논문에서는 하둡 설치 시 제공되는 하둡 파라미터 값으로는 하둡 성능이 비효율적으로 나오는 문제가 발생하여 대규모 데이터 분석에 어려움이 따르며, 이를 해결하기 위한 방안으로 하둡 파라미터중에서 성능에 영향을 미치는 항목들을 실험중심(Experiment-driven) 방법에 기반 하에 최적의 정보를 도출하여 하둡 성능 개선을 하고자 하였다. 파라미터 최적화 방법 중 가장 이상적인 방식은 기계학습을 통한 최적의 정보를 자동적으로 도출하는 알고리즘을 구현하는 것이지만 본 논문에서 실험중심 방식을 선택한 이유는 입력 데이터 규모와 자원의 성능, 용량 등을 파악하여 학습을 통해서 주요한 수십 개의 파라미터 값을 알고리즘에 의해 추출하는 것은 최적의 값을 도출하는 것이 어렵고, 구현된 알고리즘도 하둡 버전 변경 시 알고리즘 변경 등으로 추가적인 비용 발생이 되기 때문에 기업 생산성을 고려 시 현재로서는 비경제적이고 합리적이지 못하다고 판단하였기 때문이다.

본 논문에서 수행한 실험중심 기법은 하둡 성능에 큰 영향을 미치는 파라미터 요소를 선정하여 다양한 값을 통하여 최적으로 튜닝(Tuning)을 하고 하둡 버전 변경이나 자료수집 프로파일링 오버헤드(Profiling overhead) 부담이 없이 최적의 하둡 기반 환경의 파라미터를 도출하여 MapReduce 성능 개선을 하고자 한다.

본 논문의 구성은 Ⅱ장에서 하둡과 관련된 연구 현황을 기술하고, Ⅲ장은 성능 개선 및 최적화하기 위한 하둡 환경정보의 파라미터 제어 방법을, Ⅳ장에서는 실험결과 및 결과 분석, 마지막 장인 Ⅴ장에서는 결론과 향후 계획을 기술하고자 한다.

Ⅱ. 관련 연구

하둡 구조는 HDFS(Hadoop Distributed File System), MapReduce, 하둡 Yarn이 핵심적인 기능으로 구성되어 있다. HDFS는 Name node와 Data node로 구성되어 있는데, 데이터의 분산 복제와 Data node 관리, fault-tolerant 기능을 수행한다.

MapReduce는 많은 양의 데이터집합을 처리하는 응용프로그램(Application)을 관리하는 소프트웨어 프레임워크이고, 하둡 Yarn은 Resource manager와 Node manager로 구성되어 자원관리의 기능을 수행한다[3][4].

하둡 성능 개선 방법으로는 하둡 파라미터에 의한 튜닝을 하는 방안이 효율적이고, 파라미터 튜닝 방법으로는 6가지의 유형으로 분류할 수 있으며 하둡 구성(Configuration) 파라미터는 작업(Job) 수행시 러 관점(작업(Task) 동시수행, 메모리 할당, 입출력 성능, 네트워크 대역폭(Network bandwidth) 사용)에서 영향을 미친다. 또한 하둡은 200개 이상의 파라미터를 가지고 있으며 그중에서 약 60개 이상의 파라미터들이 수행되는 작업 성능에 많은 영향을 끼치고 있다[5][6]. 하둡 성능에 영향을 미치는 주요 파라미터는 표 1과 같다[5].

표 1. 하둡 성능에 영향을 미치는 주요 파라미터
Table 1. Parameters impacting in Hadoop performance

Parameter	Concept
dfs.block.size	Block size for files stored in HDFS
mapreduce.job.maps	Numbers of Map job
mapreduce.job.reduces	Numbers of Reducer job
mapreduce.map.output.compress	Flag that enables compression of Map output data
mapreduce.map.sort.spill.percent	The percentage of “mapreduce.task.io.sort.mb” to be filled before sending the map output data to the local disk
mapreduce.output.fileoutformat.compress	Flag that enables compression of job output data
mapreduce.reduce.input.buffer.percent	Ratio of Reducer memory used to store Map output data while executing Reduce task
mapreduce.reduce.shuffle.parallercopies	Maximum number of parallel threads transferring data from Map task to Reduce task
mapreduce.task.io.sort.factor	Maximum number of data streams to be combined during external sorting
mapreduce.task.io.sort.mb	Memory buffer size to store map output data
mapreduce.tasktracker.map.tasks.maximum	The maximum number of Map tasks running concurrently on the cluster node
mapreduce.tasktracker.reduce.tasks.maximum	The maximum number of Reduce tasks running concurrently on the cluster node

6가지 유형으로는 규칙기반(Rule-based), 비용중심 모델링(Cost modeling), 시뮬레이션 기반(Simulation - based), 실험중심, 기계학습(Machine learning), 적응형 튜닝(Adaptive tuning)으로 나눌 수 있고 본 논문은 위 방법 중에서 실험중심 방식을 선택한 이유는 최적의 방법인 기계학습과 적응형 튜닝은 기계학습을 통해서 도출된 파라미터 정보들이 실제 환경에 적합지 못한 경우도 발생하고, 하둡 버전 변경에 영향을 받으며 이는 비용 발생 문제가 생기기 때문에 환경 정보를 최대한 반영하여 주요 하둡 파라미터 정보를 대상으로 최적화하여 튜닝을 할 수 있는 실험중심 방법을 선택하게 된 것이다. 즉 알고리즘에 의해서 자동적으로 파라미터 정보를 추출하는 튜닝 방법은 하둡 버전에 영향을 받고 알고리즘이 주어진 환경을 적절하게 파악하여 자동으로 파라미터 최적화(Optimization)를 하는 데 어려움이 있다고 보인다[7]. 하둡 파라미터를 조정하여 성능 개선을 하는 관련 논문 중에서 블록 크기 조정이나 복제 개수, 메모리 버퍼 크기 조정을 통해서 하둡 mahaut 성능 개선 결과를 제시하였다[8][9].

튜닝방안에는 여러 방법들이 있는데 본 논문에서 채택한 물리적 클러스터링을 하여 하둡 성능을 개선하는 방안과 성능이 좋은 서버 하나를 가상화 환경에서 여러 개의 VM(Virtual Machine) 노드로 구성하고 각각의 VM에 하둡 환경을 구성하여 Name node와 Data node로 구성하여 파라미터 조정을 통해서 성능 최적화를 도출한 실험결과는 클라우드 환경에서 연구할 가치가 있다고 본다[10]. 특히 복제 파라미터나 블록 크기 조정 이외 데이터 양 조정을 통해서 성능 개선을 연구하는 방향도 또 다른 튜닝 방안으로 볼 수 있다[11].

하둡 구조는 master 역할을 하는 HDFS 와 Map Reduce로 구성되어 있다. HDFS는 데이터 저장관리를 하고 slave 역할을 하는 Data node에 데이터를 분산시키고 데이터 저장 위치를 추적하는 책임을 맡는다. MapReduce는 계산처리능력이 있으며 계산처리를 어떤 Data node에서 수행시켜야 하는지에 대한 책임이 있다[12].

2.1 HDFS

HDFS는 대용량 파일을 읽거나 저장 시에 높은 처리율로 성능이 훌륭한 프레임워크로써 master 역할을 하는 Name node와 slave 역할을 하는 Data node로 구성되어 있다[12]. HDFS의 특징은 크게 네 가지로 나눌 수 있는데 첫 번째로 하나의 파일을 여러 개의 블록으로 나누고 여러 개의 Data node에 분산 관리하는 것이다.

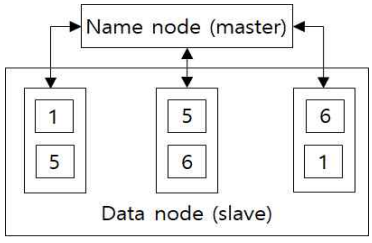

그림 1. HDFS 복제 구조
Fig. 1. HDFS Replication architecture

그림 1에서처럼 Name node는 파일을 두 개의 블록으로 나누어서 관리하는 방식을 지원한다. 두 번째 특징은 각 Data node는 자신의 운영 시스템을 변경 없이 활용할 수 있도록 하고 있고, 세 번째 특징은 데이터 블록(block)의 여러 Data node에 분산 복제가 됨으로써 fault-tolerant체계로 구성되어 있다. 마지막으로 네 번째 특징은 heartbit 기능을 지니고 있어서 Name node는 Data node에 대해서 주기적으로 heartbit을 시행하여 신호가 없는 Data node는 dead node로 간주하여 HDFS를 이용할 수 없게 하는 구조이다[3][4].

2.2 MapReduce

MapReduce는 배치(Batch)처리 기반으로써 분산처리 프레임워크이고, 대용량 데이터집합을 처리하는 응용 프로그램들의 계산처리를 지원하는 소프트웨어 플랫폼으로 정의할 수 있다[12]. MapReduce는 Map 작업과 Reduce 작업으로 구성되어 있으며, Map 작업은 Name node에 의해서 수행할 Data node를 지정하고, 입력 데이터를 하나 이상의 조각으로 나눈 후, 조각의 수만큼 각각의 Data node에서 병렬처리를 하고, 흩어져 있는 데이터를 연관성 있는 데이터 그룹으로 분류하는 기능을 수행하며, Reduce t 작업은 여러 개의 Map 작업으로부터 출력 데이터들을 모아서 유형별로 통합 분류하여 최종적으로 처리하는 과정을 지원하는 구조이다[3][4]. MapReduce의 구조는 master의 역할을 하는 Job tracker와 Job tracker의 지시를 받아서 시행하는 Task tracker로 구성되어 있다. HDFS와 MapReduce는 그림 2처럼 Master 역할을 하는 서버를 같이 활용하는 구조로 구성되어 있다[3].

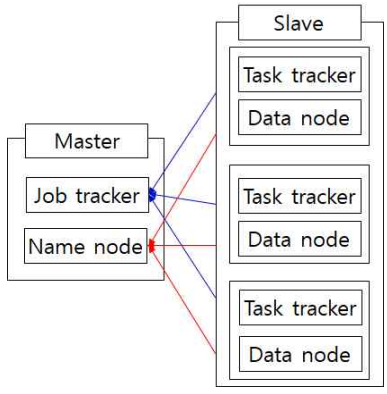

그림 2. HDFS와 MapReduce 연관 구성도
Fig. 2. Relationship architecture for HDFS and MapReduce

2.3 하둡 Yarn

하둡 Yarn은 Resource manager와 Node manager로 구성되어 있으며, 자원관리와 작업일정관리, 모니터링 기능을 여러 개의 데몬(Daemon)으로 분리시켜 수행하는 기능이 있다. Resource manager는 각 Data node에 있는 Node manager를 조정하고 응용프로그램 간에 resource 사용을 위한 중재 역할을 한다. Node manager는 Data node에 있는 container 관리, 자원사용 상태에 대한 모니터링을 수행한다[4].

III. 성능 개선 및 최적화를 위한 하둡 파라미터 제어 방법

성능 개선 및 최적화를 위해서 하둡 구성 file의 파라미터 조정을 통해서 성능 개선을 할 수 있다. 본 논문에서 채택한 튜닝방법은 실험중심 기반 하에 하둡 파라미터 중 성능에 영향을 크게 미치는 block_size 조정[7][12]이나 replication factor 조정, Mapreducer.child.java.opts, Mapreducer.speculative.execution을 조정하는 것이다. 언급한 파라미터가 위치하는 구성 file과 블록 이름(Block name)에 대한 세부 내역을 표 2에서 기술하였다.

관련된 구성 file은 Hdfs-site.xml과 Mapred_site.xml이다. 튜닝 대상의 하둡 환경의 파라미터로는 성능에 영향을 미칠 것으로 판단된 black box size, Replication node, Dataset node, Mapred.map.child.java.opt/Mapred.reduce.child.java.opt,Mapred.map.tasks.

speculative.executi-on/Mapred.reduce.tasks.speculative.execution을 대상으로 하였다[13][14].

첫 번째 튜닝방법으로는 블록 크기 조정을 하는 것으로 64MB에서 128MB, 256MB로 조정을 Hdfs_site.xml에서 하고, 변경 위치의 블록 이름은 dfs_block_size이다. 두 번째 방법인 replication factor는 시스템의 fault tolerance를 위해서 파일 블록을 복제하게 하는 역할을 한다. dfs_replication에서 지정된 수만큼 여러 Data node에 걸쳐서 복제한다. 세 번째 방법인 Reducer 개수 조정에 의한 방법은 Mapred.reduce.task에 의해서 제어되며, “Hadoop jar” 실행문에서 지정하며 다음과 같이 실 예를 들어 표현 reducer 개수를 지정하는 방법을 표현하였다. 예를 들어 10개의 Reducer를 지정하기 위해서는 “Hadoop jar Test_Parallel_for.jar Test_Parallel_for Matrix/test4.txt Result 3 -D mapred.map.tasks = 20 -D mapred.reduce.tasks =10”로 지정할 수 있다. Reducer 개수 조정에 의한 성능 개선 방법은 이번 실험에서는 제외하고 차기 실험에서 다른 새로운 patameter 방법과 같이 적용할 계획이다. 네 번째 방법인 Mapred.map.java.opts 조정은 Mapred_site.xml의 Child.java.opts 변경을 하는 방법인 것이다. 이 방법은 HDFS의 Heap memory 제어를 하는 것인데 적용(True), 미적용(False)으로 switching을 하여 기능 수행 여부를 결정할 수 있다. 마지막으로 다섯 번째 방법으로 Speculative.execution 조정 방법이 있는데 이 방법은 여러 개의 복제 작업을 병행으로 수행시키는 역할을 하는 데 이로 인해서 시스템에 부하를 줄 수 있으므로 조정에 의해서 성능 개선을 효과를 얻을 수 있다[14].

표 2. 파라미터 별 구성 file 내역
Table 2. Configuration file details by parameter

Parameter name	Configuration name	Block name
Block size adjustment	Hdfs_site.xml	dfs.block.size
Replication node adjustment	Hdfs_site.xml	dfs_replication
Adjustment of Reducer numbers	Hadoop jar	-
Child.java.opts adjustment	Mapred_site.xml	• Mapred.map.child.java.opts • Mapred.reduce.child.java.opts
Speculative.execution adjustment	Mapred_site.xml	• Mapred.map.tasks.speculative.execution • Mapred.reduce.tasks.speculative.execution

IV. 실험 결과 및 분석

4.1 실험 환경

5대로 PC를 활용하여 1대는 Name node 역할을 하고, 다른 4대는 Data node 역할을 수행하도록 구성하였다. Name node로 활용되는 서버의 하드웨어 및 시스템 소프트웨어 spec은 표 3에 기술하였고, Data node로 활용되는 서버의 주요 구성내역은 표 4에 기술하였다. OS는 Ubuntu 16.04를 설치하였고, 하둡은 버전 2.5.2를 적용하였다.

표 3. Data node system 구성내역
Table 3. Data node system specification

Item	Specification
CPU	Core i5-7500 @ 3.4GHz
number of cores	Quad-Core
OS	Ubuntu 16.04
Memory	DDR4 8GB
Network environment	1Gbps Wire LAN
Hadoop version	2.5.2
Python version	2.7 / 3.6

표 4. Name node system 구성내역
Table 4. Name node system specification

Item	Specification
CPU	Core i7-7700 @ 3.6GHz
number of cores	Quad-Core
OS version	Ubuntu 16.04
Memory	DDR4 8GB
Network environment	1Gbps Wire LAN
Hadoop version	2.5.2
Python version	2.7 / 3.6

실험하기 위해서 준비된 데이터집합은 669MB이고 처리시간 계산하기 위해서 Python으로 개발된 word count용 Map 프로그램과 Reduce 프로그램을 활용하였고, HDFS에 데이터를 저장하기 위해서 “Hadoop jar” 기능을 적용하였다.

4.2 실험결과

표 2에서 언급한 파라미터 조정 항목 중에서 4가지 항목과 각 시험 유형의 관계를 표 5에서 설명하였다.

실험 성능 측정방법은 7개의 시험유형으로 시험조건을 나누어서 각 유형별로 3개의 실험 case로 구분하여 표 6처럼 실험측정을 위한 실험측정 계획을 기획하였다. 총 시험 Case는 21개로 구성되었다.

측정시간은 “Hadoop jar” 프로그램을 그림 3과 같이 수행하여 결과 값이 나왔다. 이 명령문을 수행하기 이전에 하둡 명령어에 의해서 /user/cloudera/input directory에 수행할 파일을 저장하였다. “Hadoop jar” 실행문에 의해서 /user/cloudera/input 에 있는 파일은 Map word count와 Reducer word count 프로그램에 이해서 Map, Reduce를 수행하고 결과값은 그림 4에서와 같이 /user/cloudera/output_new_2에 세 개의 파일로 나뉘어서 저장된다.

그림 3에서 언급된 “Hadoop jar” 실행문은 각 Case별로 수행하여 Real time, User time 및 System time 결과 값을 표 7과 같이 도출하였고, 각 처리시간의 정의는 다음과 같다.

-. Real time: 프로그램 Call의 시작부터 종료까지 전체 경과 시간 (성능측정기준)
-. User time: 해당 프로세스 수행을 위해서 Kernel 외부에서 user-mode로 처리된 CPU 시간
-. System time: Kernel 내부에서 처리된 CPU 처리시간

```
/user/cloudera/output_new_2/part-00000
/user/cloudera/output_new_2/part-00001
/user/cloudera/output_new_2/part-00002
```

그림 4. Hadoop jar의 실행 결과
Fig. 4. Execution result of Hadoop jar

표 5. 조정 파라미터와 시험유형의 연관성
Table 5. Correlation between adjustment parameters and test type

Parameter name	Test type
block size adjustment	type 1 ~ type 7, adjusted from 64MB to 128MB, and from 128MB to 256MB for each type
replication node adjustment	- application of 4 replication nodes: type 1, type 2, type 3 - application of 3 replication nodes: type 4, type 7 - application of 2 replication nodes: type 5 - application of 1 replication node: type 6
child.java.opts adjustment	- application of “false” in mapred.map/reduce,child.java.opts: type 1 - application of “true” in mapred.map/reduce.child.java.opts: type 2 ~ type 7
speculative.execution adjustment	- application of “true” in mapred.map/reduce.speculative.execution: type 1, type 2, type 7 - application of false in mapred.map/reduce.speculative.execution: type 3, type 4, type 5, type 6

```
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -input /user/cloudera/input -output /user/cloudera/output_new_2
-mapper /usr/lib/hadoop-mapreduce/wordcount_mapper.py  - reducer /usr/lib/hadoop - mapreduce/wordcount_reducer.py
- numReduceTasks 3
```

그림 3. Hadoop jar 실행문
Fig. 3. Hadoop jar execution sentence

표 6. 시험 유형별 성능 측정 계획
Table 6. Performance measure plan by test type

Type	Test method	Case
Type 1	• Adjustment of block size: 64 MB -> 128 MB -> 256 MB • Replication_node: 4 numbers/ Dataset node: 4 set • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt : False • Mapred.map.tasks.speculative.execution/Mapred.reduce.tasks.speculative.execution: True	Case 1, Case 2, Case 3
Type 2	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Mapred.map.child.java.opt: False -> True • Mapred.reduce.child.java.opt: False -> True • Replication_node: 4 numbers/ Dataset node: 4 set • Mapred.map/reduce.tasks.speculative.execution: True	Case 4, Case 5, Case 6
Type 3	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Mapred.map.tasks.speculative.execution adjustment: True -> False • Mapred.reduce.tasks.speculative.execution adjustment: True -> False • Replication_node: 4 numbers/ Dataset node: 4 set • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt: True	Case 10, Case 11, Case 12
Type 4	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Replication_node adjustment: 4 numbers -> 3 numbers • Dataset node: 4 set • Mapred.map.tasks.speculative.execution/Mapred.reduce.tasks.speculative.execution: False • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt: True	Case 10, Case 11, Case 12
Type 5	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Replication_node adjustment: 3 numbers -> 2 numbers • Dataset node: 4 set • Mapred.map.tasks.speculative.execution/Mapred.reduce.tasks.speculative.execution: False • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt: True	Case 13, Case 14, Case 15
Type 6	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Replication_node 변경: 2 numbers -> 1 number • Dataset node: 4 set • Mapred.map.tasks.speculative.execution/Mapred.reduce.tasks.speculative.execution: False • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt: True	Case 16, Case 17, Case 18
Type 7	• Adjustment of block size:: 64 MB -> 128 MB -> 256 MB • Replication_node adjustment: 1 number -> 3 numbers • Mapred.map/reduce.speculative.execution: change from “false” to “true“ • Dataset node: 4 set • Mapred.map.child.java.opt/Mapred.reduce.child.java.opt: True	Case 19, Case 20, Case 21

표 7. Type/Case별 측정 시간
Table 7. Measurement time by type/case

Type 구분	Case 구분	Block (MB)	Real time (sec)	User time (sec)	System time (sec)
Type 1	Case 1	64	721.958	757.804	92.888
	Case 2	128	774.304	759.120	85.712
	Case 3	256	642.382	744.108	83.884
Type 2	Case 4	256	625.826	734.004	83.756
	Case 5	128	649.953	751.596	83.856
	Case 6	64	645.681	741.104	84.516
Type 3	Case 7	64	660.375	755.504	84.908
	Case 8	128	649.317	748.772	83.276
	Case 9	256	652.706	749.732	84.080
Type 4	Case 10	256	636.216	737.668	84.388
	Case 11	128	615.363	723.112	83.312
	Case 12	64	661.599	758.972	84.576
Type 5	Case 13	64	658.333	758.380	83.048
	Case 14	128	638.595	740.280	83.820
	Case 15	256	643.629	793.336	82.884
Type 6	Case 16	256	659.241	752.976	84.424
	Case 17	128	632.436	732.800	83.412
	Case 18	64	638.039	735.476	83.904
Type 7	Case 19	64	625.755	741.756	83.544
	Case 20	128	642.945	748.696	84.836
	Case 21	256	643.998	747.180	83.976

4.3 실험 분석

Real time에서 성능이 가장 좋은 경우는 그림 5에서 살펴본다면 Case 11로써 Type 4에 해당하며 Real time은 615초이다. 실험환경은 128MB 블록 크기에서 Replication node는 3개(Replication_factor =3), speculative.execution = false, child.java.opt = false로 설정한 경우이다. 같은 환경에서 블록 크기가 256 MB인 경우가 636초가 진행된 것을 볼 때 블록 크기를 크게 설정하는 것이 반드시 시스템 성능 측면에서 효율적이지 않다. 또한 Replication node도 3개로 설정했을 때가 2개나 1개 설정할 때보다 더 짧게 걸린 것을 보면 복제를 많이 한다고 해서 처리시간이 오래 걸린다고 볼 수 없는 것이다. 적절하게 주어진 환경에서 최적의 환경을 찾는 것이 중요하다고 볼 수 있다. 가장 처리시간이 오래 걸린 경우는 Case 2로써 Type 1에 해당되며 Real time이 774초이다. Case 2의 시험 환경은 블록 크기는 128MB이고, Replication node는 4개, mapred.child. java.opt는 false, mapred.speculative.execution는 True이다.

User time에서 성능이 가장 좋은 경우는 그림 6에서 빨간 원으로 표현한 Type 4에 속하는 Case 11이 723초로 가장 짧은 시간 안에 종료되고, 가장 오랜 시간 동안 지속된 경우는 파랑 원으로 표현한 Type 5에 포함되어 있는 Case 15로 793초이다. Case 11은 블록 크기가 128MB이고, 실험환경은 128MB 블록 크기에서 Replication node는 3개, Speculative. execution은 false, Child.java.opt도 false로 설정한 경우이다. Case 15는 256 MB의 블록 크기로 설정하였고 Replication node는 2개로 설정하였으며 Mapred. speculative.execution:는 false로 지정, Mapred..java.opt는 true로 설정한 상태이다.

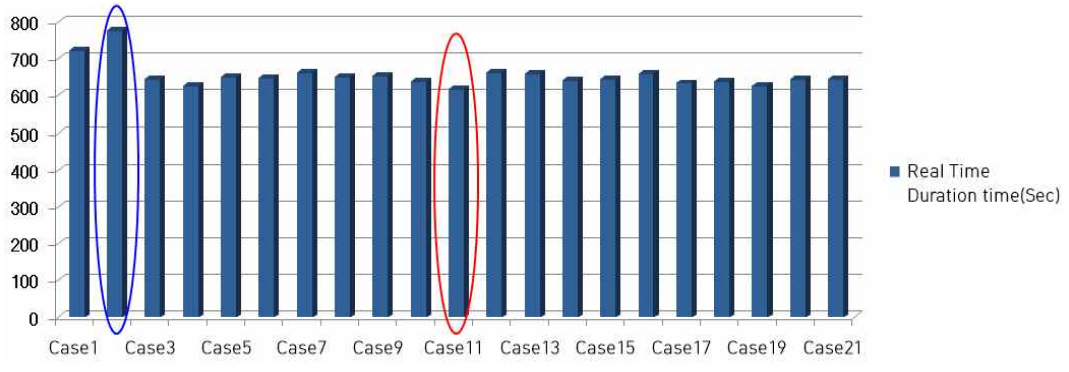

그림 5. Case별 Real time
Fig. 5. Real time by case

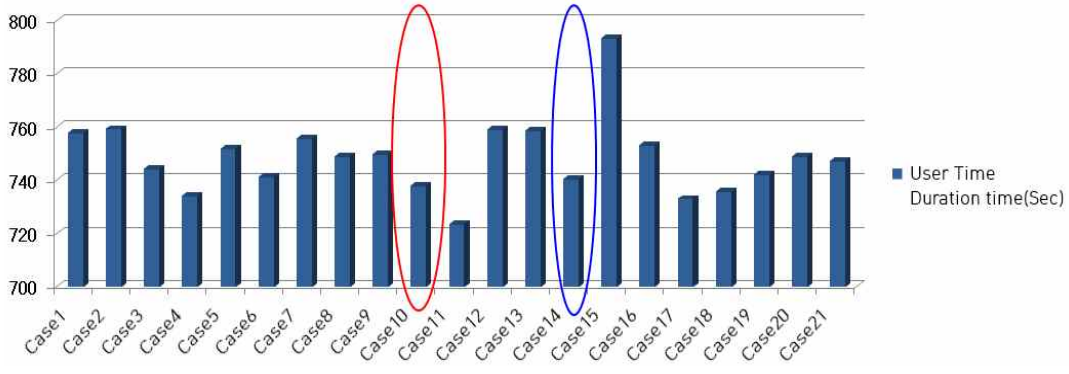

그림 6. Case별 User 처리시간
Fig. 6. User duration time by case

User time을 분석해보면 가장 성능이 좋은 Case 11은 Replication node가 세 개이고, 가장 성능이 안 좋은 Case 15는 Replication node가 2개이다. 즉 복제를 적게 한다고 해서 성능이 우수한 것은 아니다. Speculative.execution은 복사본 작업도 병행해서 같이 진행하는 기능인데 Case 15인 경우에는 false로 설정하였음에도 불구하고 가장 오랜 시간이 소요된 것을 보면 성능을 우선으로 하는 프로젝트에서 크게 고려할 요소는 아닌 것으로 판단된다. Case 3은 블록 크기를 256MB, Speculative.executive를 true로 설정하였어도 744초가 걸렸다.

시스템 time은 커널 내부에서 CPU가 사용하는 시간으로 그림 7에서처럼 Type 5에 속하는 Case 13이 83초 소요로 가장 성능이 좋게 나왔다. Replication node는 2개이고 Mapred.speculative.execution은 false, Mapred.child.java.opt는 true이다.

실험내용의 분석 결과에 따르면 시스템 성능 최적화를 위해서 본 논문의 자원 환경에서는 Replication node를 Data node 수인 4개로 하는 것보다는 3개정 도로 유지하고, 블록 크기는 최대 256MB보다는 128MB가 성능 최적화를 위해서 적정하다는 것이 실험결과로 나왔다. 또한 복제 작업을 병행 진행하도록 제어하는 Speculative.execution은 병행 처리를 하는 것인 중요한 경우를 제외하고는 원래 작업만 진행하는 것이 효율적이다. 이러한 실험결과가 나온 추정적 이유는 블록 크기가 너무 커지면 map 작업 개수가 감소함으로써 병렬처리성이 감소하여 overhead가 증가함으로써 성능에 오히려 부담을 주게 되기 때문에 블록크기를 적정하게 하는 것이 합리적이고, 원본 작업만 수행 시에는 그만큼 자원 소요가 복제 작업을 같이 수행함으로써 발생되는 자원 소요보다 작기 때문에 성능이 더 효율적이다. 최적의 환경은 블록 크기가 128MB이고 복제 개수는 3개로 설정된 경우로서 표 8로 정리하였다.

표 8. 하둡 최적화 파라미터 환경
Table 8. Hadoop optimized parameter environment

Parameter type	Default value	Optimized value
Block size	128MB	128MB
Replication node	4	3
Speculative.execution	true	false
Mapred.child.java.opts	true	false

V. 결 론

대규모 데이터집합을 활용해서 기계학습기반의 데이터 패턴 분석과 예측 필요성 요구가 커지고 있고, 이를 수행하기 위해서는 기반 프레임워크가 중요한데 관련 프레임워크로는 스파크와 하둡 방식이 있는데 cluster에 분산되어 있는 데이터집합을 동시에 관리 및 처리하기 위해서는 RDD 기반의 memory 방식으로 운영되는 스파크보다는 cluster 데이터 집합을 관리하는 하둡 방식이 선호된다. 그러나 하둡의 취약점인 성능 향상을 위해서는 공통 환경에 설정된 파라미터 튜닝을 통해서 최대한 최적화가 필요하다.

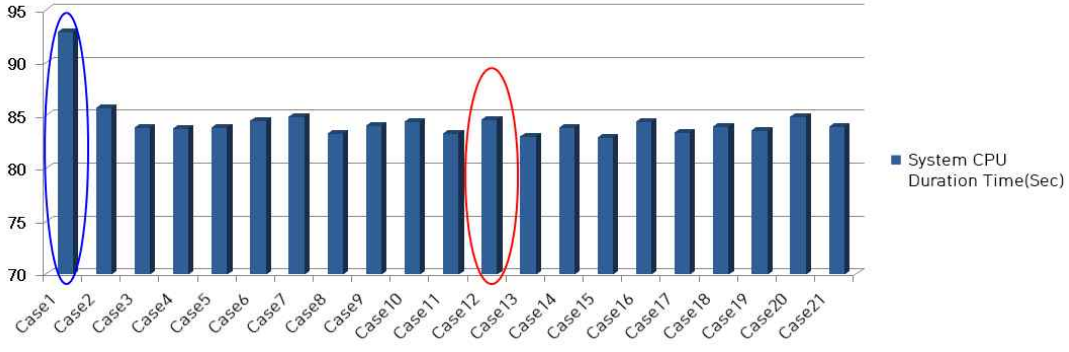

그림 7. Case별 System 처리시간
Fig. 7. System duration time by case

본 논문에서는 주어진 환경(Name node 1 set, secondary Name node 1 set, Data node 4 set)에서 실험중심 방식기반으로 최적의 환경을 도출하고자 input data 669 MB을 활용해서 MapReduce 파라미터 중 일부를 최적화해서 Real time과 User time, System time을 도출하였다. 특히 Real time과 User time의 최적화 환경은 128MB의 블록 크기, 복제 개수 3 (default는 Data node 개수인 4로 설정), Speculative.execution은 false(default는 true로 설정), Mapred.child.java.opts는 false(default는 true로 설정)가 최적의 성능을 나타냈다. 블록 크기는 이 실험 환경에서 크게 설정하는 것이 성능 측면에서 오히려 불리하다는 것을 알 수 있다. 또한 복제 개수도 적게 잡는 것이 성능측면에 유리하지 않는 것을 알 수 있었다.

자동적인 튜닝 방법은 시스템 환경을 적절하게 파악하는 것이 용이하지 않고, 하둡 버전에 영향을 받으며 특히 분산 환경 하에서는 전체 자원 상태를 파악하고 자료수집 프로파일링 오버헤드 등 성능 효율성, 비용과 시간 측면에서 합리적이지 않다고 생각한다.

향후 계획으로는 하둡 프레임워크에 기계학습(Machine learning)을 적용하여 성능에 영향을 미치는 파라미터 요소(factor)를 더 도출하여 성능 개선을 최적화하고, spam과 ham 에 대한 예측을 하는 알고리즘을 연구하고자 한다.

References

[1] "Hadoop and Spark’ compared with pros and cons, ecosystem, and use cases", http://www.itworld.co.kr/news/ 100370 [accessed: Mar. 28, 2021]

[2] ankus. http://www.openankus.org/zeus/?page_id=615&page=1&wr_id=22 [accessed: Mar. 28, 2021]

[3] K. Han, "Do it! Hadoop with big data", easyspub, pp. 53-69, Feb. 2013.

[4] Apache Hadoop, http://hadoop.apache.org/ [accessed: Mar. 21, 2020]

[5] Herodotos Herodotou, YUXING CHEN, and JIAHENG LU, "A Survey on Automatic Parameter Tuning for Big Data Processing Systems", ACM Computing Surveys, Vol. 53, No. 2, Article 43, pp. 43:1-43:37, Apr. 26, 2020. http://dx.doi.org/10.1145/3381027.

[6] K. Ji and Y. Kwon, "MapReduce Performance Optimization by Changing Hadoop Parameter", Proceedings of KIIT Conference, Cheongju Korea, pp. 1-5, Oct. 2020.

[7] Sandeep Kumar, Sindhu Padakandla, Chandrashekar L, Priyank Parihar, K Gopinath, and Shalabh Bhatnagar, "Performance Tuning of Hadoop MapReduce: A Noisy Gradient Approach", IEEE 10th International Conference on Cloud Computing, Honolulu, Hawaii, USA, pp. 25-30, Jun. 2017.

[8] S. H. Jeon, H. J. Chung, and Y. M. Nah, "Improving Machine Learning Perfomance by Hadoop Mapreduce Tuning", Proceedings of Korea Information Science Society, pp. 731-733, Dec. 2017.

[9] Tien Duc Dinh, "Hadoop Performance Evaluation", Ruprecht-Karls University, Heidelberg Institute of Computer Science Research Group Parallel and Distributed Systems, pp. 25-36, Mar. 2009.

[10] Y. H. Kim, H. Jeong. W. Choi, J. Kim, and J. Choi, "Hadoop performance analysis for distributed Big-data processing in a virtualized cluster environment", Korean Institute of Information Scientists and Engineers, Vol. 39(2C), pp. 13-15, Nov. 2012.

[11] S. H. Jeon, H. Chung, W. Choi, H. Shin, J. Chun, J. T. Kim, and Y. Nah, "MapReduce Tuning to Improve Distributed Machine Learning Performance", IEEE First International Conference on Artificial Intelligence and Knowledge Engineering(AIKE), Laguna Hills, CA, USA, Vol. 1, pp. 198-200, Sep. 26-28, 2018. https://doi.ieeecomputersociety.org/10.1109/AIKE.2018.00045.

[12] ALEX HOLMES, "Hadoop in Practice", Manning Publications Co, pp. 4-6, Oct. 13, 2012.

[13] J. Jeong, "MapReduce Performance Tuning for BIG DATA", GRUTER, pp. 16-19, Nov. 2012.

[14] Sanjay Sharma, "Advanced Hadoop tuning and Optimizations", https://www.slideshare.net/ImpetusInfo/ppt-on-advanced-hadoop-tuning-n-optimisation, [accessed: Mar. 27, 2021]

저자소개

지 경 엽 (Keungyeup Ji)

1989년 2월 : 광운대학교 전자계산학과(공학사)
2016년 2월 : 충남대학교 전자정보통신공학과(공학석사)
2017년 3월 ~ 현재 : 충남대학교 전자정보통신공학과 박사과정
관심분야 : Hadoop, Spark, Cloud Computing, Machine Learning, Data Mining

권 영 미 (Youngmi Kwon)

1986년 2월 : 서울대학교 컴퓨터공학과(공학사)
1988년 2월 : 서울대학교 컴퓨터공학과 공학석사
1996년 8월 : 서울대학교 컴퓨터공학과 공학박사
1993년 ~ 1995년 : ETRI 연구원
1996년 ~ 2002년 : 목원대학교 컴퓨터공학과 조교수
2006년 ~ 2007년 : Indian Statistical Institute 객원연구원
2002년 ~ 현재 : 충남대학교 전파정보통신공학과 교수
관심분야 : Internet Protocols, WSN, Embedded System, Cloud Computing, Distributed System