Regular paper
Journal of the KIECS. pp. 19-24, vol. 16, no. 1, Feb. 28. 2021, t. 105, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2021.16.1.19

스파크를 이용한 머신러닝의 분산 처리 성능 요인

류우석*

Performance Factor of Distributed Processing of Machine Learning using Spark

Woo-Seok Ryu*

요 약

본 논문에서는 아파치 스파크를 이용하여 머신러닝을 분산 처리할 때의 성능 요인을 분석하고 효율적인 분산 처리를 위한 실행 환경을 실험을 통해 제시한다. 먼저, 분산 클러스터 환경에서 머신러닝을 수행할 때 고려해야 하는 성능 요인으로 클러스터의 성능, 데이터의 규모, 스파크 엔진의 속성으로 구분하여 분석한다. 그리고 하둡 클러스터에서 동작하는 스파크 MLlib을 이용하여 회귀분석을 수행할 때 노드의 구성과 스파크 Executor의 설정을 변화하면서 성능을 측정한다. 실험 결과 최적의 Executor 개수는 데이터의 블록의 수에 영향을 받으나 클러스터 규모에 따라 최대값, 최소값은 각각 코어의 수, 워커 노드의 수로 제한됨을 실증하였다.

ABSTRACT

In this paper, we study performance factor of machine learning in the distributed environment using Apache Spark and presents an efficient distributed processing method through experiments. This work firstly presents performance factor when performing machine learning in a distributed cluster by classifying cluster performance, data size, and configuration of spark engine. In addition, performance study of regression analysis using Spark MLlib running on the Hadoop cluster is performed while changing the configuration of the node and the Spark Executor. As a result of the experiment, it was confirmed that the effective number of executors was affected by the number of data blocks, but depending on the cluster size, the maximum and minimum values were limited by the number of cores and the number of worker nodes, respectively.

키워드

Spark, Cluster, Machine Learning, Distributed Processing
스파크, 클러스터, 머신 러닝, 분산 처리

Ⅰ. 서 론

빅데이터 분석방법론 중 최근 가장 각광받고 있는 머신러닝은 다양한 학습 알고리즘을 통해 빅데이터에 내재되어 있는 의미를 도출하는 기법으로써 빅데이터를 활용하는 다양한 분야에서 적용 범위를 넓혀가고 있다[1][2][3]. 빅데이터의 효율적 분산 처리를 지원하는 하둡(Hadoop) 프레임워크는 수천 개의 노드 이상을 지원하는 규모 확장성을 가지고 있으나, 연산 결과를 디스크에 저장하고 클러스터 내 전송을 수행하므로 실행 성능의 제약이 발생하는 단점이 있다. 아파치 스파크(Apache Spark)는 인메모리 기반 병렬 분산 엔진으로서 데이터셋을 가능한 메모리 내에 유지시키고 연산을 수행하므로 하둡 대비 월등한 성능을 보이

* 교신저자 : 부산가톨릭대학교 병원경영학과
• 접 수 일 : 2020. 11. 23
• 수정완료일 : 2021. 01. 05
• 게재확정일 : 2021. 02. 17
• Received : Nov. 12. 2020, Revised : Jan. 05, 2021, Accepted : Feb. 17, 2021
• Corresponding Author : Woo-Seok Ryu
Dept. of Health Care Management, Catholic University of Pusan,
Email : wsryu@cup.ac.kr

고 있다[4]. 또한, 스파크는 스파크 스트리밍(Spark Streaming), MLlib, SparkSQL등의 다양한 라이브러리를 지원하여 병렬 분산 처리에서의 활용 영역을 넓혀가고 있다[5].

머신러닝을 지원하는 대표적인 도구로 텐서플로(Tensorflow)[6], 아파치 머하웃(Apache Mahout)[7] 등 다양한 도구가 개발되어 있다. 그중에서도 아파치 스파크에서 제공하는 MLlib 라이브러리는 다양한 개발환경 지원, 하둡과 같은 여러 분산 클러스터 지원 등의 장점을 통해 대용량 데이터를 이용한 대규모 머신러닝에서 많이 활용되고 있다[8]. 본 연구에서는 대량의 데이터를 하둡 클러스터에 적재시키고 스파크 MLlib을 이용하여 머신러닝을 수행할 때 발생하는 성능 요인들을 분석하고 실험을 통해 분산처리의 효율성을 높이는 최적의 설정값을 제시하고자 한다.

본 논문의 구성은 다음과 같다. 먼저 2장에서 아파치 스파크의 특성을 분석하고, 3장에서는 스파크에서의 분산 처리시 고려해야 하는 성능 요인을 제시한다. 4장에서는 다양한 데이터셋과 클러스터 설정에 따른 스파크의 회귀분석 성능을 비교 분석하고 5장에서 결론 및 향후연구를 기술한다.

II. 아파치 스파크

아파치 스파크는 대규모 분산 데이터의 고속 처리를 지원하는 오픈소스 프레임워크로서 메모리 기반 분산 데이터 처리를 지원함에 따라 저장장치 및 네트워크 I/O를 감소시킴으로서 그 성능을 높이는 특징이 있다. 스파크는 독자적인 분산 클러스터에서 동작하는 것은 물론 하둡 얀(Yarn), 아파치 메소스(Apache Mesos), 쿠버네티스(Kubernetes) 및 아마존 EC2와 같은 기존 분산 클러스터를 지원한다. 그리고, HDFS, 카산드라(Cassandra), HBase, 아파치 하이브(Apache Hive) 등의 다양한 분산 데이터 저장소를 지원한다.

그림 1은 분산 클러스터 환경에서의 스파크의 구성요소를 도시하고 있다. 사용자 애플리케이션은 드라이버 프로그램 내 SparkContext를 통해 관리되며 클러스터 매니저를 통해 각 워커 노드에 있는 Executor를 할당받은 후 각 Executor에 Task를 전달하여 실행한다[9]. 이때 클러스터 매니저가 얀인 경우 드라이버 프로그램과 Executor는 각각 애플리케이션 마스터와 노드 매니저 내의 컨테이너로서 동작하게 된다.

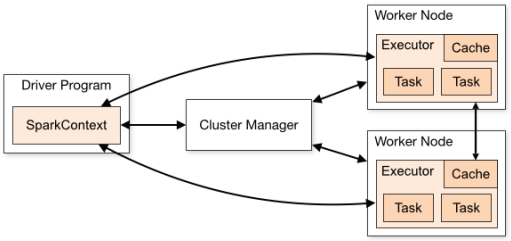

그림 1. 클러스터에서의 스파크 구성 요소[9]
Fig. 1 Components of spark in the cluster[9]

MLlib은 스파크에서 제공하는 머신러닝 라이브러리로서 자바, 스칼라(Scala), 파이썬 및 R의 다양한 프로그래밍 언어를 지원하고 다양한 클러스터 매니저를 지원하므로 개발의 용이성 및 규모 확장성이 뛰어나다. 회귀분석, 의사결정나무, K-Means 클러스터링 등 대부분의 머신러닝 알고리즘을 지원하고 있으며, 딥러닝은 DeepSpark, SparkNet 등의 별도 라이브러리를 통해 지원한다[10].

아파치 스파크의 성능과 관련한 연구로서 [11]은 랜덤 포레스트 알고리즘과 로지스틱 회귀분석 알고리즘을 캐시와 코어수를 변화하면서 성능을 비교하였으며, [12]는 스파크 내 파티셔닝 최적화 기법을 제시하였으나 두 연구 모두 하나의 가상머신 또는 제한된 클러스터를 이용하였으므로 규모 확장성을 검증하는 데에는 한계가 있었다.

III. 스파크의 분산 성능 요인

본 장에서는 아파치의 분산처리에 영향을 주는 주요 요소를 하드웨어, 소프트웨어, 데이터의 측면에서 분석하고자 한다. 먼저 하드웨어는 워커 노드의 수, 코어의 수, 메모리 용량, 네트워크의 속도 등이 있다. 클러스터의 규모를 의미하는 워커 노드의 수는 노드 간 병렬화 수준을 결정하는 요소이며, 코어의 개수는 노드 내 병렬화 수준을 결정하는 요소이다. 이때, 네트워크 속도는 워커 노드의 수가 많아질수록 전체 처리시간에 영향을 미친다. 메모리 용량은 인-메모리

처리를 수행하는 Executor의 성능에 직접적인 영향을 미치며 태스크 수행과정에서 메모리 용량이 부족한 경우 디스크 I/O가 추가로 발생하는 특징이 있다.

소프트웨어 측면에서 사용자가 설정 가능한 분산 성능 요인은 Executor의 수(spark.executor.instances), Executor 메모리(spark.executor.memory) 및 코어 수(spark.executor.cores)이다. Executor는 단일 워커 노드에서 응용프로그램 태스크를 수행하는 자바가상머신(JVM) 프로세스를 의미하며 스파크 애플리케이션 실행시 사용자가 Executor의 수를 지정할 수 있다. Executor 메모리 및 코어 수는 하나의 Executor가 사용하는 메모리 용량 및 코어의 수를 각각 의미한다. 이때 Executor의 수는 응용 프로그램 실행 시 병렬처리의 수준을 결정할 수 있으며, 메모리 용량 및 코어의 수는 하나의 노드에 한정되므로 하드웨어 사양에 의존적인 특징이 있다.

머신러닝과 같은 알고리즘은 저장소에서 데이터를 읽은 후 이를 반복해서 계산하는 특징이 있는데 데이터의 I/O도 성능의 주요 요소가 된다. 하둡 분산 파일 시스템(HDFS)에서는 데이터의 배치 처리를 위해 블록 단위로 저장하고 이를 중복 계수(replication factor)에 따라 여러 노드에 나누어 저장한다. 분산 저장소를 활용하는 경우 데이터의 지역성(Data locality)이 스파크의 처리 성능에 영향을 미치게 된다. 즉, 데이터가 저장된 노드에서 연산 처리를 수행하는 것이 불필요한 네트워크 I/O를 줄일 수 있다.

위 요인들을 종합하여 볼 때, 분석 대상의 데이터 크기에 따라 최적화된 클러스터의 설정을 계산 가능하다. 데이터가 저장된 블록 개수를 N_B라고 하면 Executor의 개수(N_E)는 N_B와 동일하게 설정할 때 데이터 접근을 동시에 이용할 수 있으므로 분산 처리가 최대화될 것으로 생각할 수 있다. 하지만, 이는 데이터 지역성과 연계되어 있는데 Executor가 특정 노드에 집중되는 경우 노드의 처리 부하가 발생하는 문제가 있다. 또한, 클러스터의 규모가 큰 경우 유휴 노드가 발생하므로 시스템 자원을 모두 활용하지 못하는 문제가 있다. 클러스터 규모가 기 설정되어 있는 경우 최적의 N_E는 N_B와 코어의 개수(N_C)에 영향을 받는다. N_E는 N_C를 Executor당 코어 수로 나눈 값이 동시 수행의 최대값으로 설정할 수 있는데 만일 N_B가 노드의 수(N_N)보다 작은 경우에는 불필요한 병렬화로 인한 네트워크 I/O가 발생할 수 있다.

이와 같이 최적의 클러스터 설정은 하드웨어, 소프트웨어, 데이터 측면에서의 변수들이 서로 영향을 미치므로 다음 장에서 실험을 통해 변수들 간의 관계를 검증하고자 한다.

Ⅳ. 성능 평가

4.1 실험 환경

이 장에서는 분산 클러스터에서 머신러닝을 수행할 때 성능을 결정하는 주요 요인인 노드의 개수와 분산 노드 각각에서 Job을 처리하는 프로세스 단위인 Executor의 개수가 변화할 때 발생하는 아파치의 성능 변화를 분석한다.

실험을 위한 분산 클러스터는 2 코어 인텔 펜티엄 프로세서, 4GB 메모리와 500GB 하드디스크를 장착한 최대 9대의 PC를 클러스터로 구성하였고 이중 1대의 노드를 마스터 노드로 구성하고 워커 노드는 분산 성능을 측정하기 위해 1대부터 8대까지 가변적으로 구성하였다. 소프트웨어 구성으로 분산 클러스터에는 우분투 16.04와 하둡 2.7.4 버전을 이용하여 구성하였다. 아파치는 3.0.1버전을 설치하였으며 클러스터 매니저는 YARN을, 그리고 분산 파일시스템은 HDFS를 이용하였다.

실험을 위한 데이터셋은 국민건강보험공단에서 제공하는 2018년 진료내역정보[1]를 이용하였다. 본 데이터는 국민건강보험 가입자 중 진료이력이 있는 수진자 100만명의 기본정보 및 진료내역으로 구성된 개방데이터로서 약 1297만 건의 진료내역이 저장되어 있다. 본 실험에서는 데이터 규모에 따른 성능 비교를 위해 데이터셋1(100만명), 데이터셋2(50만명) 및 데이터셋3(25만명)으로 데이터셋을 세분화하였다. 이때 데이터 용량은 약 2GB, 1GB, 500MB으로 구성하였다.

실험 환경에서의 스파크와 하둡의 설정은 다음과 같다. HDFS에서 데이터 블록의 사본 수를 의미하는 복제 계수(dfs.replication)은 기본값인 3으로 설정하였으며, 데이터 블록의 크기도 기본값인 128MB을 유지하였다. 이에 각 데이터셋1, 2, 3의 블록의 개수는 각

1) 국민건강보험공단 공공데이터 개방서비스, 홈페이지: https://nhiss.nhis.or.kr/op/it/index.do

각 18개, 9개, 5개로 HDFS에 저장되었다.

본 데이터셋을 이용한 머신러닝 프로그램은 MLLib에 포함되어 있는 ml.regression 패키지의 LinearRegression을 이용하였다. 실험에 사용된 코드는 사전에 생성한 libsvm 포맷의 데이터를 로드하고 선형회귀 모델을 학습하는 데 까지 소요되는 시간을 측정하도록 구현하였다. 성능 측정은 데이터셋에 대해 워커 노드 개수 및 Executor 개수를 달리하면서 5회씩 실행 후 평균 실행 시간을 측정하였다.

표 1. 실험 환경 변수
Table 1. Configuration of the experiment

System	Variable	Value
HDFS	replication factor	3
	block size	128MB
YARN	memory size of node manager	3072MB
Spark	core of executor	1
	memory size of executor	512MB
	the number of executors	1 to 10

4.2. 실험 결과

그림 2는 2GB 데이터셋에 대해 프로그램을 실행하였을 때 노드 수와 Executor 수에 따른 실행시간의 추이를 도시한 그림이다. 이 데이터셋은 18개의 데이터 블록을 가지고 있으므로 클러스터 규모를 6개의 노드로 확장하여도 N_C가 12에 제한됨에 따라 N_B가 N_C보다 항상 큰 특성이 있다. 처리 시간은 N_E가 N_N일 때까지 빠른 폭으로 감소하며 N_C와 동일해질 때까지 지속적으로 감소하지만, N_C보다 증가하면 시간이 소폭 증가함을 확인하였다.

예를 들어, N_N이 3인 경우 N_E가 6이상이 되면 실행시간이 더 이상 감소하지 않으므로 적정 N_E는 N_C와 동일할 때로 확인되었다. 본 실험에서는 N_N과 N_E 두 값이 각각 6, 10일 때 가장 높은 성능을 보였으나 각각 5, 10일 때와 큰 차이를 보이지는 않았다. N_N이 6일 때 N_E가 12까지 증가한다면 추가적인 성능 향상이 있을 것으로 예상된다.

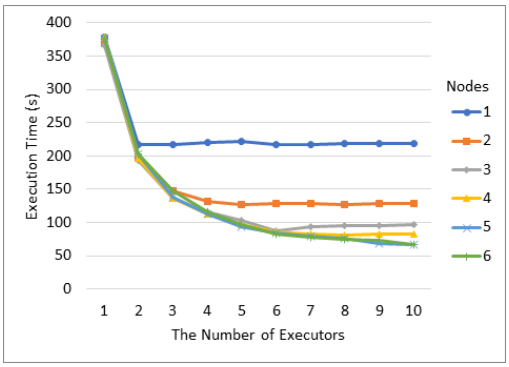

그림 2. 워커 노드 및 Executor 수에 따른 데이터셋1(2GB)의 수행 성능
Fig. 2 Performance of dataset 1 according to the number of worker nodes and executors

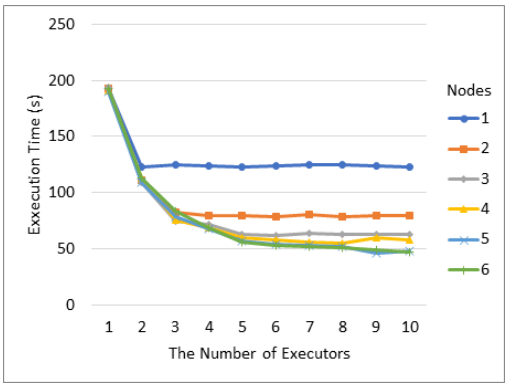

그림 3. 워커 노드 및 Executor 수에 따른 데이터셋2(1GB)의 수행 성능
Fig. 3 Performance of dataset 2 according to the number of worker nodes and executors

그림 3은 1GB 데이터셋, 즉 N_B가 9개일 때의 실행결과이다. 전체적으로 그림 1과 유사한 결과를 보이고 있으며, 실험에서는 N_N과 N_E 두 값이 각각 6, 10일 때 가장 높은 성능을 보였다. 다만 N_N이 5일 때에도 성능의 변화는 크지 않았는데, 이때 N_E가 N_B와 동일한 9일 때보다 Nc와 동일한 10일 때 성능이 오히려 하락하였다. 즉, 적정 N_E의 최대값은 Nc보다 N_B가 더 영향을 미친다는 것을 추정할 수 있다.

그림 4는 N_B가 5인 500MB 데이터셋에 대한 실행결과이다. N_N이 2일 경우 N_E가 4를 초과하면 오히려 성능이 떨어지는데 이는 코어 수보다 많은 Executor

로 인해 발생하는 컨텍스트 스위칭 비용이 분산으로 인한 네트워크보다 더 높아서 발생한 것으로 판단된다. N_N이 6인 경우는 노드 수가 블록 수보다 많은 경우로서 N_E를 N_B와 동일하게 설정할 때 보다 N_N과 동일하게 설정하였을 때 더 높은 성능을 보였다.

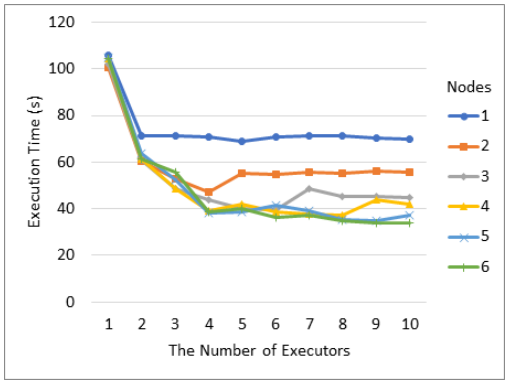

그림 4. 워커 노드 및 Executor 수에 따른 데이터셋1(500MB)의 수행 성능
Fig. 4 Performance of dataset 1 according to the number of worker nodes and executors

그림 5는 동일한 500MB 데이터셋에 대해 로딩 시간과 학습 시간을 나누어서 표시한 그림이다. N_N이 2일 때 N_E가 2~4인 구간과 N_N이 3일 때 N_E가 3~6인 구간에서 보는 바와 같이 N_E가 N_C만큼 증가하여도 로딩 시간에 큰 차이는 없으나 학습시간은 줄어드는 것을 확인할 수 있다. 그리고, N_N이 7 이상인 경우는 실험결과에는 도시되어 있지 않았는데, 데이터셋 2와 3의 경우 N_E를 늘려가면서 수행한 5회의 반복실험의 편차가 매우 크게 나와서 실험결과의 신뢰성이 부족하였다. 데이터셋 대비 클러스터의 규모가 매우 큰 경우는 데이터 지역성으로 인한 네트워크 I/O가 예측하기 어려운 수준으로 증가한 것으로 추정할 수 있다.

실험결과를 정리해 볼 때 머신러닝과 같이 CPU 자원을 많이 사용하는 응용 프로그램을 스파크에서 실행하는 경우 적정 Executor 수는 클러스터의 노드 수와 코어 수(Executor당 코어가 1인 경우) 이하의 범위 내에서 데이터 블록의 개수만큼 지정하는 것이 효과적임을 확인할 수 있다.

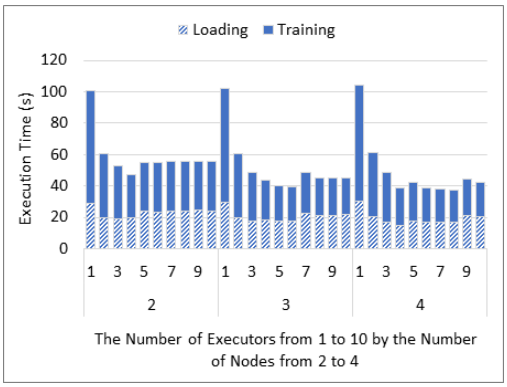

그림 5. 워크 노드 및 Executor 수에 따른 데이터셋1(500MB)의 로딩시간 및 학습시간 비교
Fig. 5 Comparison of loading time and training time of dataset 1 according to the number of worker nodes and executors

V. 결 론

본 논문에서는 아파치 스파크를 이용하여 분산 클러스터 환경에서 머신러닝을 수행할 때 클러스터의 규모 및 Executor의 설정에 따른 처리 시간을 실험을 통해 검증하고 최적의 Executor 설정을 제시하였다. 스파크에서 병렬 분산 처리의 단위가 되는 Executor는 데이터셋의 블록 개수에 영향을 받는데, 클러스터 규모보다 데이터셋의 개수가 더 큰 경우에는 태스크 전환, 네트워크 I/O에 따른 오버헤드를 줄이기 위해 클러스터 전체 코어의 수를 넘지 않도록 제한하는 것이 필요하다. 클러스터의 규모가 충분히 큰 경우에는 노드의 개수만큼 Executor를 설정함으로써 병렬성을 최대화하는 것이 효과적이라는 사실을 실험을 통해서 검증하였다. 또한, 분석할 데이터셋의 크기가 고정된 경우에는 해당 데이터셋의 규모에 맞게 클러스터의 규모를 역으로 산정할 수 있다.

본 연구에서는 워커 노드를 최대 6대로 설정하여 실험하였는데, 추후 연구로서 본 연구를 구글 및 아마존 클라우드로 확대하여 데이터에 따른 최적의 클라우드 클러스터 규모를 산정할 수 있는 클러스터 규모 모델을 산정하는 것이 필요하다.

감사의 글

이 논문은 2018년도 부산가톨릭대학교 교내연구비에 의하여 연구되었음

References

[1] Y. Jeong and K. Choi, "For Gene disease Analysis using Data Mining Implement MKSV system," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 14, no. 4, Aug. 2019, pp. 781-786.

[2] N. Shahid, T. Rappon, and W. Berta, "Applications of artificial neural networks in health care organizational decision-making: A scoping review," *PLOS ONE*, vol. 14, no. 2, Feb. 2019, pp. 1-22.

[3] Y. Bae and D. Hwang, "The prediction of bidding price using deep learning in the electronic bidding," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 15 no. 1, Feb. 2020, pp. 147-152.

[4] I. Mavridis and H. Karatza, "Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark," *J. of Systems and Software*, vol. 125, Mar. 2017, pp. 133-151.

[5] M. Zaharia, R. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. Franklin, and A. Ghodsi, " Apache spark: a unified engine for big data processing," *Communications of the ACM,* vol. 59, no. 11, Oct. 2016, pp. 56-65.

[6] J. Jo, "Performance Comparison Analysis of AI Supervised Learning Methods of Tensorflow and Scikit-Learn in the Writing Digit Data," *J. of the Korea Institute of Electronic Communication Sciences*, vol. 14, no. 4, Aug. 2019, pp. 701-706.

[7] R. Anil, G. Capan, I. Drost-Fromm, T. Dunning, E. Friedman, T. Grant, S. Quinn, P. Ranjan, S. Schelter, and Ö. Yılmazel, "Apache Mahout: Machine Learning on Distributed Dataflow Systems," *J. of Machine Learning Research*, vol. 21, no. 127, Jan. 2020, pp. 1-6.

[8] M. Assefi, E. Behravesh, G. Liu, and A. P. Tafti, "Big data machine learning using apache spark MLlib," In *2017 IEEE Int. Conf. on Big Data (Big Data)*, Boston, MA, U.S.A., 2017, pp. 3492-3498.

[9] M. Frampton, *Mastering Apache Spark.* Birmingham: Packt Publishing Ltd, 2015.

[10] J. Jang, J. Park, H. Kim, and S. Yoon, "A Comparative Performance Analysis of Spark-Based Distributed Deep-Learning Frameworks," *KIISE Trans. Computing Practices*, vol. 23, no. 5, May 2017, pp. 299-303.

[11] A. Garate-Escamilla, A. Hassani, and E. Andres, "Big data scalability based on Spark Machine Learning Libraries," In *Proc. the 3rd International Conference on Big Data Research*, Cergy-Pontoise, France, Nov. 2019, pp. 166-171.

[12] R. Myung, H. Yu, and S. Choi, "Performance Optimization Strategies for Fully Utilizing Apache Spark," *KIPS Trans. Computer and Communication Systems*, vol. 7, no. 1, Jan. 2018, pp. 9-18.

저자 소개

류우석(Woo-Seok Ryu)

1997년 부산대학교 컴퓨터공학과 졸업 (공학사)

1999년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사)

2012년 부산대학교 대학원 컴퓨터공학과 졸업(공학박사)

2013년~현재 부산가톨릭대학교 병원경영학과 부교수

※ 관심분야 : 의료정보, 빅데이터, 병렬분산 처리, 머신러닝