Journal of Korea Multimedia Society Vol. 23, No. 2, February 2020(pp. 241-246)
https://doi.org/10.9717/kmms.2020.23.2.241

Spark 기반에서 Python과 Scala API의 성능 비교 분석

지경엽†, 권영미††

Performance Comparison of Python and Scala APIs in Spark Distributed Cluster Computing System

Keung-yeup Ji†, Youngmi Kwon††

ABSTRACT

Hadoop is a framework to process large data sets in a distributed way across clusters of nodes. It has been a popular platform to process big data, but in recent years, other platforms became competitive ones depending on the characteristics of the application. Spark is one of distributed platforms to enable real-time data processing and improve overall processing performance over Hadoop by introducing in-memory processing instead of disk I/O. Whereas Hadoop is designed to work on Java and data analysis is processed using Java API, Spark provides a variety of APIs with Scala, Python, Java and R. In this paper, the goal is to find out whether the APIs of different programming languages affect the performances in Spark. We chose two popular APIs: Python and Scala. Python is easy to learn and is used in AI domain in a wide range. Scala is a programming language with advantages of parallelism. Our experiment shows much faster processing with Scala API than Python API. For the performance issues on AI-based analysis, further study is needed.

Key words: Spark, Scala, Python, Master Node, Worker Node

1. 서 론

기업의 주요 비즈니스용 시스템이 웹기반으로 B2B나 B2C를 시행함으로서 데이터 규모가 급격히 증가하고 있고, 타 기업에 대한 경쟁력 향상과 제품의 품질향상을 위해서 적재된 데이터에 대한 분석의 필요성이 요구되고 있다. 또한 스토리지 구입 및 유지비용이 낮아짐에 따라 기업은 대규모 데이터를 이용한 영업 전략의 구현 및 빅데이터 분석을 통한 마케팅 경쟁력을 강화하고 있다. 데이터 분석 성능 향상을 위한 방법으로는 기계학습을 통하여서 이상적인 데이터 군집화를 함으로써 검색 성능향상을 달성할 수도 있으나[1], 본 논문에서는 빅데이터 처리 성능 향상을 위한 방안으로 API 방식을 실험적인 과정을 통해 확인하려고 한다. 실험 목적은 Spark로 구성된 동일한 분산 플랫폼에서 Scala와 Python의 서로 다른 API로 작업을 수행할 때 전체 성능에 어떤 영향을 미치는지 실험을 통해 분석하였다. 실험 환경은 같은 작업을 worker node의 개수를 1개 및 2개, 3개로 변화시켜가면서 각 node별로 20회씩의 실험을 통해 성능에 어떤 영향을 미치는지도 측정하였다. Spark는 다양한 프로그래밍언어로 구현된 API를 지원한다. 본 논문은 다른 연구에서는 아직 수행하지 않은 동일한 Spark 플랫폼 환경에서 Scala와 Python의 서

※ Corresponding Author : Youngmi Kwon, Address: (305-764) 79 Daehangno, Yuseong-gu, Daejeon, Korea, TEL : +82-42-821-6890, FAX : +82-42-823-5586, E-mail : ymkwon@cnu.ac.kr
Receipt date : Nov. 19, 2019, Revision date : Jan. 3, 2020
Approval date : Jan. 13, 2020

† Dept. of Radio and Info. Communications Eng., Chungnam National University
(E-mail : kyglimins@kaist.ac.kr)
†† Dept. of Radio and Info. Communications Eng., Chungnam National University
※ This work was supported by research fund of Chungnam National University

로 다른 API의 사용이 성능에 어떤 영향을 미치는지에 대한 연구는 아직 없었기 때문에 복수의 API에 대해서 실험을 하였다. 따라서 본 논문에서는 Spark 기반의 플랫폼을 직접 구축해서 Python과 Scala방식으로 application을 구현하여 각각의 성능을 비교 측정 해보고자 한다. worker node의 개수별로 각각 어느 정도의 성능 차이가 나는지를 파악하여 최적의 클러스터를 구성하여 빅 데이터 처리를 하고자 했다.

본 논문의 구성은 다음과 같다. 2장에서는 본 논문과 관련된 Spark 프레임워크 구성 및 RDD(Resilient Distributed Dataset)에 대해 좀 더 자세히 알아본다. 3장에서는 Spark 기반에서 Python과 Scala API 에 따른 성능을 비교하기 위한 실험 환경과 내용을 설명한다. 4장에서는 API에 따른 처리시간의 차이에 대한 실험 결과를 보이고, 5장에서 결론 및 향후 계획을 기술한다.

2. Spark 개요

2.1 Spark 개념

Spark는 빅 데이터 처리를 위한 인 메모리 기반의 대용량 데이터 고속처리 엔진을 보유하고 있는 확장성이 뛰어난 분산데이터 분석 시스템으로서, 기존의 디스크 I/O 방식이었던 빅 데이터 엔진들에 비해 데이터를 메모리에 저장하고 처리함으로써 데이터 처리 속도가 높다. 또한 범용의 분산 클러스터 컴퓨팅 프레임워크 형태를 취하고 있어서 Spark SQL, Spark stream, Spark machine learning library, Spark GraphX와 같은 라이브러리를 통해 기존의 분산처리 시스템의 한계였던 실시간 데이터 처리를 용이하게 하고, 요즘 모든 분야에 각광받는 머신러닝 알고리즘을 수용할 수 있는 모델로 개발되어 있다. 또한 Fig. 1과 같이 Java, Python, Scala, R 등의 언어로 개발할 수 있으므로 개발자의 선택의 폭이 넓고, 생산성과 효율성이 높다[2].

Spark의 발전과정은 2009년 UC Berkeley lab에서 시작된 오픈소스 프로젝트로서, 2013년 6월 아파치 프로젝트로 채택되고 2014년 2월에 아파치 최상위 프로젝트가 되었으며 현재는 AMPlabs에서 독립한 스타트업인 데이터브릭스(Databricks)등이 주축이 되어 전 세계 100여명의 전문가가 개발에 참여하고 있다[3].

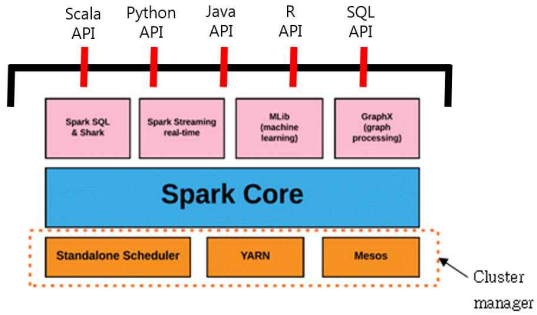

Fig. 1. Apache Spark framework and APIs.

2.2 Spark 구조 및 framework

Spark는 Fig. 1과 같은 프레임워크로 구성되어 있다. Spark core는 작업스케줄링, 메모리관리, 장애복구, 저장장치와 연동되어 동작하는 핵심 코드로 구성되어 있으며 핵심 컴포넌트로 RDD(Resilient Distributed Dataset)에 연산기능을 추가해서 데이터를 신속하게 처리할 수 있는 기능을 지원한다. Spark SQL은 Spark 환경 하에서 SQL처럼 데이터를 조회할 수 있게 해주고, 정형데이터를 처리하기 위한 Spark 라이브러리이다. Spark Streaming은 Spark에서 확장가능하고 높은 처리율과 fault-tolerant한 기능을 지원해주며, 다양한 형태의 데이터를 처리하고 처리된 데이터는 실시간으로 HDFS, DB, dashboard등으로 보내진다. MLlib(Machine Learning library)은 확장 가능한 머신러닝 알고리즘을 지원하는 라이브러리이며 Java, Python, Scala 및 R의 인터페이스를 지원해준다. GraphX는 분산형 그래프 프로세싱이 가능하게 지원해주는 라이브러리이다. 마지막으로 executor를 실행하는 역할을 하는 Cluster Manager에는 Standalone Scheduler, Hadoop의 YARN, Apache Mesos로 구성된다.

Cluster manager는 Fig. 2와 같이 Spark context와 Spark worker node 사이에 중계자 역할을 하여서 여러 worker node가 협동으로 작업을 수행하게 해준다[4].

Fig. 2에 보인 바와 같이 Spark는 클러스터 상에서 동작하는 Spark context와 Spark worker node와의 인터액션을 줄여 데이터 송수신으로 인한 오버헤드를 줄이고, 로컬 메모리에서 많은 일을 수행함으로써 분산 데이터 처리에 기존의 하둡보다 빠른 처리 속도를 보인다.

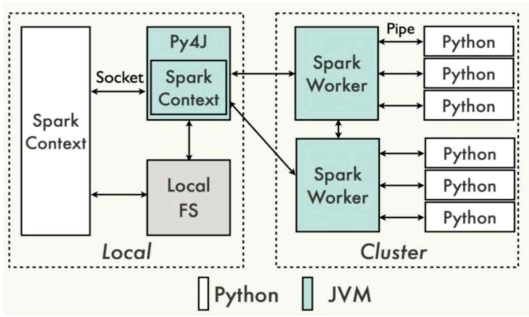

Fig. 2. Structure of Spark cluster.

2.3 RDD(Resilient Distributed Dataset)

RDD(Resilient distributed Dataset)는 Spark에서 중요한 개념으로서, 병렬로 수행될 수 있는 element의 집합이다. 즉 여러 node에 분산 저장되어 있는 변경 불가능한 데이터의 집합이다. Spark에서 모든 작업은 새로운 RDD를 생성하거나 존재하는 RDD를 변형하거나, 결과 계산을 위해 RDD를 연산하는 것이다[5, 6].

2.4 Py4J와 Pyspark

Py4J는 Python 인터프리터에서 실행되는 Python 프로그램이 JVM(Java Virtual Machine)의 Java 객체에 동적으로 액세스가 가능하게 한다. 또한 Py4J는 자바 프로그램이 Python 객체와 연동할 수 있게 하며, BSD 라이센스하에 배포된다[7]. Method는 Python interpreter에 있는 Java 객체와 Java collection이 표준 Python collection method를 통해 액세스 할 수 있는 것처럼 호출된다. PySpark는 Spark의 Java API 위에 구축되었으며 데이터를 Python에서 처리하고 JVM에서caching 및 shuffling되게 한다. Shuffling은 partition 간에 데이터를 재분배하는 프로세스이다[8].

3. Spark 기반의 Python vs. Scala

본 논문의 목적은 Spark 기반의 동일한 플랫폼 환경에서 Python과 Scala방식으로 구현된 application의 수행 시간을 비교함으로써 서로 다른 API가 Spark의 전체 성능에 미치는 영향을 분석하려는 것이다. 이 기본 분석은 향후 빅 데이터 및 인공지능 처리 어플리케이션을 개발할 때 성능 저하 문제를 고려해 개발자가 API를 선택하게 하는 데 참고가 될 수 있을 것이다. 실험 환경 구성은 Fig. 2의 Spark 구성에서 1개의 master node와 다수의 worker node를 구축해 동작하게 하였으며, worker node를 1개에서 3개까지 다르게 구성하면서 각 상황에서의 수행 시간을 측정하였다. 사용한 데이터는 미국 항공사의 항공 자료로 120 GB를 사용하였으며[9], 항공 지연이 일어난 달이나 요일 등을 분석하기 위한 작업내용을 주고 결과를 얻기까지 걸리는 시간을 측정하였다. Spark가 제공하는 API 중 Python과 Scala를 선택한 이유는, Python이 현재 머신러닝 응용을 개발하는 주된 언어이고, Scala는 동시처리성(concurrency) 면에서 우수한 언어로 알려져 있기 때문이다[10].

3.1 실험 환경

Master node의 사양은 Table 1과 같이 구성하였다. Intel core i7-7700의 CPU환경이며 worker node의 사양보다 더 좋은 프로세서를 사용하였다. 또한 master node와 worker node를 1Gbps의 유선 랜으로 별도의 네트워크 환경을 구축하였다. 하둡과 Map Reduce 기반 하에 dataset 처리를 위해서 Spark를 이용하였다.

Worker node의 사양은 다음과 같다. Table 2와 같이 Intel Core i5-7500의 CPU환경에서 동일한 로직의 알고리즘을 각각 Python과 Scala로 작성하고 실행하여 120 GB의 실험 데이터를 사용하여 성능을

Table 1. Master node specification

Item	Specification
CPU	Core i7-7700 @ 3.6GHz
Number of core	Quad-Core
OS	CentOS 6.6
Memory	DDR4 8GB
Network environment	1Gbps Wire LAN

Table 2. Worker node specification

Item	Specification
CPU	Core i5-7500 @ 3.4GHz
Number of core	Quad-Core
OS	CentOS 6.6
Memory	DDR4 8GB
Network environment	1Gbps Wire LAN

No	Algorithm steps
1:	import csv file, Spark package
2:	Parse a row(128 MB) and returns a named tuple
3:	Load csv file of the airlines
4:	Broadcast the csv file to cluster
5:	Read the CSV data into RDD
6:	Declare Map/Reduce
7:	Configure Spark
8:	Compute processing time

Fig. 3. Python Code Steps

No	Algorithm steps
1:	Import Spark context and Spark configuration
2:	Configure Spark
3:	Upload CSV file and declare Map function
4:	Compute processing time for dataset

Fig. 4. Scala Code Steps.

비교하였다. 또한 각각의 API마다 worker node의 개수를 1개, 2개 및 3개로 증가시켜가며 실험 하였다.

3.2 실험 내용

두 API의 성능 실험을 위해 동일한 목적을 지닌 알고리즘의 코드를 각각 Python과 Scala로 작성하였다. 알고리즘의 내용은 미국 교통부의 실시간 비행 데이터를 기반으로 하였으며, 각 실험마다 120 GB의 데이터를 가지고 시행하였다. 실험 데이터는 미국 내에서의 모든 상업용 비행기의 출발과 도착에 관련된 데이터로서 1987년 10월부터 2008년 4월까지의 내용이다.

Fig. 3은 Python code로 구성하기 위한 알고리즘이다. 실험 데이터를 loading해서 RDD에 저장한후 Map/Reduce 기능을 활용하여 Spark 환경에서 dataset loading 처리를 하며 처리 시간을 worker node 별로 비교한다[11].

Fig. 4는 Scala로 구현하기 위한 알고리즘으로서 Spark 환경을 구성하고 실험 데이터 CSV 파일을

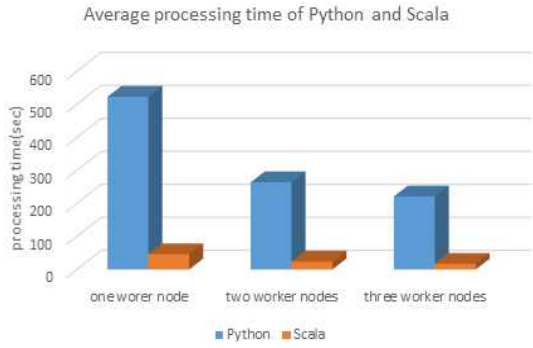

Fig. 5. Processing time of Python and Scala per worker node.

loading한 후 대용량 데이터 처리를 위한 Map 기능으로 수행하며 worker node 별로 실험 데이터에 대한 처리 시간을 계산한다.

4. 실험 결과

4.1 Worker node 별 Python과 Scala 성능 비교 결과

120 GB의 실험 데이터를 활용하여 20회 실험을 반복하였고 처리시간 단위는 초(sec)이다. Table 3은 worker node의 개수를 1개~3개로 변화시켜가며 Scala와 Python 코드를 20회 수행하였을 때의 평균 수행 시간을 보인 것이며, Python과 Scala의 평균 성능 비교값을 마지막 열에 보였다.

Fig. 5는 Table 3의 데이터를 그래프로 보인 것이다. Python과 Scala 코드 모두 worker node의 개수가 늘어날수록 전체 수행시간이 단축되므로, worker node의 개수를 늘리는 것이 빅데이터 처리의 수행시간 단축에 기여함을 알 수 있다.

4.2 결과 분석

Table 4는 worker node의 수가 1개에서 2개로, 2개에서 3개로 늘어났을 때의 수행시간 단축을 각각 나타낸 것이고 Fig. 6과 Fig. 7은 Table 4의 데이터를 그래프로 보인 것이다. Fig. 6을 보면worker node의

Table 3. Average processing time of Python vs. Scala

type	Python (sec)	Scala (sec)	average ratio (Python/Scala)
one worker node	523	46	11
two worker nodes	264	24	11
three worker nodes	221	17	13

Table 4. Performance improvement time and ratio

# of worker nodes	Python improvement		Scala improvement	
	Reduced time (sec)	Reduced time ratio (%)	Reduced time (sec)	Reduced time ratio (%)
one->two nodes	523 - 264 = 259	49.5%	46 - 24 = 22	47.8%
two-> three nodes	264 - 221 = 43	16.3%	24 - 17 = 7	29.2%

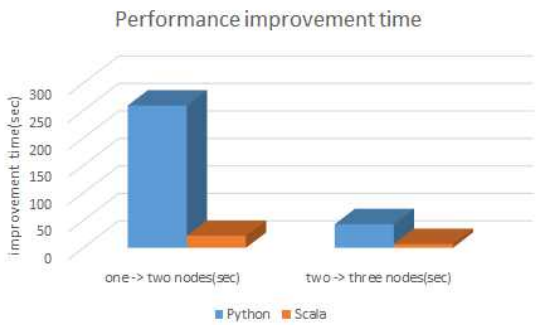

Fig. 6. Performance improvement time.

Fig. 7. Performance improvement ratio.

수를 하나에서 두 개로 증가시켰을 때 Python은 평균 259초가 단축되었고, Scala는 평균 22초가 단축되었다. 그러나 worker node를 두 개에서 세 개로 증가시켰을 때는 Python은 평균 43초가 단축되었고 Scala는 평균 7초가 단축되었다. 즉 worker node 개수의 증가와 비례하여 성능이 개선되는 것은 아니다. 따라서, 모든 분석 응용에 필요한 worker node의 개수는 하나로 정해져 있지 않으며, 각 application의 특성과 데이터의 규모에 따라 정하는 것이 좋다.

Fig. 7에서 worker node가 하나에서 두 개로 증가하였을 때, Python이 49.5%의 성능 개선을 보였고, Scala는 평균 47.8%의 성능 개선을 나타냈다. Worker node가 두 개에서 세 개로 증가했을 때는 Python이 평균 16.3%의 성능 개선을, Scala는 29.2%의 성능 개선을 보였다. 1개에서 2개로 worker node가 늘어났을 때, 두 언어의 코드 수행 시간이 모두 거의 50%의 감소를 보여, 단일 worker node의 구성이 전체 수행시간 단축에 매우 비효율적임을 볼 수 있다. 그 이상의 worker node 개수의 증가는 Python처럼 수행시간이 큰 API의 경우는 여전히 크게 시간을 단축할 수 있는 요인이지만, Scala같이 전체 수행시간이 작은 경우에는 이미 전체 수행 시간이 작기 때문에 그 시간을 단축시키는 데 효과가 아주 크지는 않음을 알 수 있다. 다만, Python이 인공지능 기능으로 빅데이터를 분석하는 데 유용하게 사용되는 API이고, Scala가 병렬처리에 장점을 보이는 API라는 점은 분석에 제외하였다. 쿼리와 같이 단순한 빅 데이터처리를 Spark 환경에서 할 때, 어떤 API가 어떤 성능을 보이는 가를 비교 분석한 결과이다.

5. 결론 및 향후 계획

본 논문에서는 Spark의 worker node를 1개에서 3개로 증가시켜가면서 Python과 Scala API의 성능 비교를 하였다. 측정된 실험 결과는 Scala가 Python보다 성능이 11배에서 13배까지 빠른 결과가 나왔다. 이렇게 성능의 차이가 나는 이유는 RDD(Resilient distributed Dataset)가 JVM(Java Virtual Machine)에 저장되기 때문에 JVM 위에서 동작하는 Scala가 그렇지 않은 Python보다 더 좋은 성능을 가져왔기 때문이다. 성능보다 데이터 분석측면을 우선하는 경우에는 배우기 쉽고 많이 보급된 Python을 사용해도 무방할 것이며, 실시간 분석과 짧은 시간 내에 분석할 필요가 있는 경우에는 Scala를 사용하여 개발하는 것이 좋다 하겠다.

향후 연구 계획으로는 Java와 Python사이의 Interface 역할을 하는 Py4J(Python for Java)를 이용하여 Docker와 Container 기반의 PySpark(Spark와 Python간의 API)환경에서, machine learning 기

능을 활용해서 정확도를 높이는 데이터 분석을 위한, 성능이 개선된 Python 기반의 application을 개발할 계획이다. 이때 상대적으로 느린 Python의 속도 개선을 위해서 내장식(Built-in expression)을 가급적 많이 활용하고자 한다.

REFERENCES

[1] H. Lee, "Performance Improvement of Deep Clustering Networks for Multi Dimensional Data," *Journal of Korea Multimedia Society*, Vol. 21, No. 8, pp. 952-959, 2018.

[2] Spark 2.4.4 API Documentation, https://spark.apache.org/docs/latest/api.html (accessed November 1, 2019).

[3] IT World Glossary|Spark, http://www.itworld.co.kr/news/92835 (accessed April 9, 2015).

[4] Real-world Python Workloads on Spark: Standalone Clusters, https://becominghuman.ai/real-world-python-workloads-on-spark-standalone-clusters-2246346c7040 (accessed November 1, 2019).

[5] Tuning Spark Serialization, https://spark.apache.org/docs/latest/tuning.html (accessed April 4, 2017).

[6] Understanding Sparking Caching, https://spark.rstudio.com/guides/caching/ (accessed April 4, 2017).

[7] Py4J-A Bridge between Python and Java, https://www.py4j.org (accessed October 21, 2018).

[8] Spark/Wiki Homepage/Spark Internals/PySpark Internals, https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals (accessed November 22, 2016).

[9] ASA Sections on Statistical Computing Statistical Graphics/Data Expo Airline On-time Performance, http://stat-computing.org/data-expo/2009/ (accessed April 4, 2017).

[10] Spark: Python or Scala?, https://brunch.co.kr/@abrahamsong/43 (accessed November 1, 2019).

[11] B. Bengfort and J. Kim, *Data Analytics with Hadoop: An Introduction for Data Scientists*, O'Reilly Media, California, 2016.

지 경 엽

1989년 광운대학교, 전자계산학과 학사
2016년 충남대학교, 전자정보통신공학과 석사
2017년 충남대학교, 전자정보통신공학과 박사과정

2006년~현재 한국과학기술원 정보화전략팀, 책임기술원
관심분야 : Data Mining, DW, Hadoop, Spark

권 영 미

1986년 서울대학교, 컴퓨터공학과 학사
1988년 서울대학교, 컴퓨터공학과 석사
1996년 서울대학교, 컴퓨터공학과 박사
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