Journal of Electrical and Electronic Engineering
2016; 4(3): 51-56
http://www.sciencepublishinggroup.com/j/jeee
doi: 10.11648/j.jeee.20160403.12
ISSN: 2329-1613 (Print); ISSN: 2329-1605 (Online)

Fast Type Recognition of Missive Insulator Leakage Current Data Using Spark

Song Yaqi

Department of Computer Science, North China Electric Power University, Baoding, China

Email address:
bdsyq@163.com

To cite this article:
Song Yaqi. Fast Type Recognition of Missive Insulator Leakage Current Data Using Spark. *Journal of Electrical and Electronic Engineering*. Vol. 4, No. 3, 2016, pp. 51-56. doi: 10.11648/j.jeee.20160403.12

Received: April 14, 2016; **Accepted**: May 17, 2016; **Published**: May 24, 2016

Abstract: Performance is the key issue in power big data applications. One of main challenges is how to exploit these technologies in building power big data processing platform and facilitating science discoveries such as those in electric power systems. This paper explores how Spark and Cloud computing can accelerate performance of missive insulator leak current data pattern recognition. We have designed and implemented the Parallel KNN(k-NearestNeighbor) algorithm using Spark and then deployed onto the Aliyun E-MapReduce cloud computing platform. The results from experiments shows the performance and scalability can be enhanced through these advanced technologies.

Keywords: Insulator Leakage Current, Electric Power Big Data, Spark

1. Introduction

Electric power equipment monitoring are developing from single-parameter monitoring to all-round and multi-parameter monitoring. The monitoring data showed exponential growth. Remote monitoring center of power equipments is faced with heavy tasks, such as data collection, processing, storage and analysis when large amounts of monitoring data flock in.

The traditional single-machine environment which using a single task is only applicable to small amount of data and will be difficult to finish data processing tasks on time or even unable to deal with when facing large volume of data [1].

[2] stores and manages massive neuroimaging data by integrating database management systems (DBMS) with Grid computing. [3] propose a dynamic programming algorithm for pattern-based time series classification on GPUs. Given the industrial real-time demand, [4] propose a parallelized method to model the Elman network, which shifts the computational intensive tasks of network training on GPU. [5] study a large scale EEG (electroencephalogram) distributed data storage method on Hadoop [6]. Cloudera used Hadoop to store and manage around 1.5 trillion points of time-series data in 15TB of PMU archive files at TVA [7]. Although the Hadoop MapReduce [8] can effectively deal with large amounts of data, it will frequently access disks and the task need to take up several minutes of even hours. In view of complex iterative computing tasks in electric power equipment condition evaluation, the MapReduce can not finish analysis and pattern recognition tasks for large amounts of alarm monitoring data in a short limited time and performance is difficult to meet the requirements [9].

Apache Spark is a fast and general engine for large-scale data processing [10]. It has an advanced DAG execution engine that supports cyclic data flow and in-memory computing. In some computing tasks, Spark run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk. Spark offers over 80 high-level operators that make it easy to build parallel apps. And one can use it interactively from the Scala, Python and R shells. Spark has been widely used in seismic data analysis [11], data analysis in smart grid [12], GATK DNA analysis [13], data reduction [14], etc.

This paper studied fast pattern recognition of electric power equipment monitoring data using Spark. Spark-based K-Nearest Neighbor algorithm (KNN) is designed and implemented in Aliyun E-MapReduce platform and used for insulator leakage current data type identification. The results from experiments show that Spark-KNN runs up to 2.97x faster than MapReduce-based one.

2. Distributed Storage of Monitoring Data in RDD

Resilient Distributed Dataset (RDD) is the core concept in Spark framework. It can hold any type of data. Spark stores data in RDD on different partitions. It helps with rearranging the computations and optimizing the data processing. It also has fault tolerance because an RDD know how to recreate and recompute the datasets. RDDs are immutable and can be modified with a transformation with the result of a new RDD returned, whereas the original RDD remains the same. RDD provides a rich set of operations to manipulate data, including map, flatMap, filter, join, group By, reduce By Key, etc which facilitate distributed data processing.

The waveform or extracted features of power equipment monitoring data is stored and organized as RDDs. RDD can be understood as a large array, but the array is distributed on the cluster. A RDD logically is composed of multiple partitions which are corresponding to physical block in data node memory. The process of executing analysis includes a series of transformations and actions for RDDs. Monitoring data is stored in the RDD as shown in Figure 1.

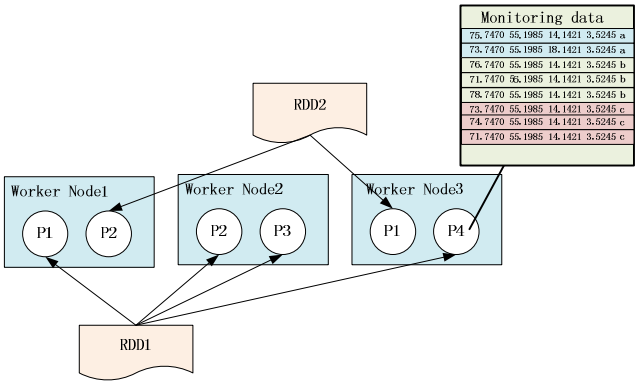


Figure 1. *Distributed storage of monitoring data in RDD.*

In Figure 1, RDD1 contains four partition (P1, P2, P3, P4) and are distributed stored in three nodes (Worker Node1, Worker Node2 and Worker Node3). RDD2 contains two partitions (P1 and P2) and are stored in 2 nodes (Worker Node3 and Worker Node1).

3. The Spark-KNN Algorithm for Fast Pattern Recognition

In pattern recognition, the k-Nearest Neighbors algorithm (k-NN) is a non-parametric method used for classification and regression [15]. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance. The k-NN algorithm is among the simplest of all machine learning algorithms.

This paper studied parallel KNN algorithm using Spark, called Spark-KNN. The input and output can be from local file system, HDFS, or OSS, etc. Spark-KNN algorithm is described in Table 1.

Table 1. *Spark-KNN algorithm.*

Spark-KNN algorithm
Input: TrainSet file(cvs format);TestSet file(cvs format); ResultSet file path; Parameter k; Output: ResultSet file(cvs format); Procedure: 1: Initialize SparkContext environment parameters; 2: Load TrainSet to RDD using SparkContext.textFile( ); Do format conversion using RDD.map( ); map(line => {vardatas = line.split(" ") (datas(0), datas(1), datas(2))}) 3: Executing RDD.collect( ) to get a scala Array for TrainSet in Driver node named TrainSet_Array; 4: Broadcast the TrainSet_Array to every node using SparkContext.broadcast( ); 5: Broadcast the parameter k to every node using SparkContext.broadcast( ); 6: Load TestSet to RDD using SparkContext.textFile( ); Do format conversion using RDD.map( ), as described at step 2; 7: Get the result set using RDD.map( ): 7.1: Parsing a test sample tuple, extracting the characteristics; 7.2: distance_set= trainDatas.foreach(trainData =>(characteristics，distance，category)}); 7.3: Sorting the distance_set according to the ascending order of distance; 7.4 Get the first k points and their categories; 8: ResultSet.saveAsTextFile(ResultSet file path);

The flowchart for Spark-KNN is shown in Figure 2.

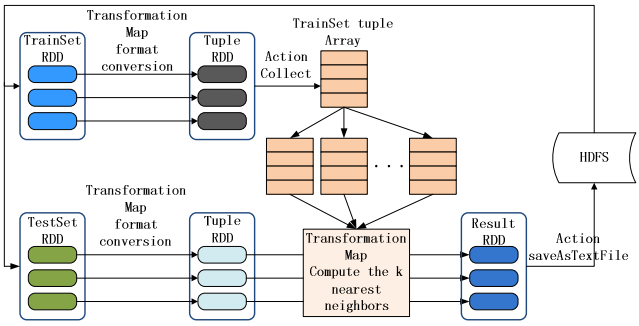


Figure 2. *Data processing flow in Spark-KNN.*

In Figure 2, Input data are loaded from HDFS using textFile function of SparkContext class. Then input data are organized as RDDs. Format conversion operations are conducted by the map transformation. A new RDD will be produced after map transformation. Colletc is a kind of Action and return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data. Broadcast is a kind of Action and allows the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. They can be used, for example, to give every node a copy of a large input dataset in an efficient manner. Spark also attempts to distribute broadcast variables using efficient broadcast algorithms to reduce communication cost. 'saveAsTextFile' is

a kind of Action and can write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call 'toString' on each element to convert it to a line of text in the file.

4. Experimental Evaluation in the Cloud

4.1. The Experiment Environment

We have conducted the experiments based on the Cloudcomputing platform. We deploy our prototype in the AliyunE-Map Reduce platform by renting virtual computer nodes (ECS.S3. Large type) from Aliyun ECS. The configuration of each machine is described in Table 2.

Table 2. *Configuration of a virtual machine fromAliyun ECS.*

Item	Configuration
CPU	Intel Xeon CPU,4 cores
Memory	8GB memory
Storage	80GB SSD cloud disk
Network bandwidth	8MB
Operating system	Federa Core 8 (2.6.21.7-2.ec2.v1.2.fc8xenLinux Kernel)
Platform major version	EMR 1.0.0
Software	hive 1.0.1; ganglia 3.7.2; Spark 1.4.1; yarn 2.6.0; pig 0.14.0

We deploy the Spark cluster using Spark on YARN mode. The ganglia (3.7.2) is a scalable distributed monitoring system for high-performance computing systems such as clusters and Grids. We use it to monitor the CPU and memory utilization so as to make adjustment and optimization of parallel tasks configuration parameters, such as 'number-exector', etc. The following 4 parameters are very important as shown in Table 3.

Table 3. *Parameter Configuration of Spark job.*

Parameter	Description	Default value	Our value
--executor-memory	Memory per executor	1GB	2GB
--driver-memory	Memory for driver	512MB	1GB
--num-executors	Number of executors to launch	2	4
--executor-cores	Number of cores per executor	1 in YARN mode	4

4.2. Experimental Data

This paper focus on the insulator leakage current data pattern recognition intransmission line monitoring system. We select the widely used four features (maximum leakage current, the 50-Hz, 150-Hz and 250-Hz amplitudes after Fourier transform) to form a 4-dimension vector for pattern recognition using Spark-KNN. Some samples selected from training set are shown in table 4.

Table 4. *Samples in training set.*

Type	maximum (mA)	50Hzamplitude (mA)	150Hzamplitude (mA)	250Hzamplitude (mA)
Stage A	14.8936	12.4707	0.1082	0.1016
Stage A	18.0136	14.7075	0.8962	0.1175
Stage A	59.1919	44.0040	11.5511	2.7286
Stage B	87.6251	63.2405	15.7299	3.7481
Stage B	92.7320	68.5759	17.1612	4.2756
Stage C	138.9287	102.3116	20.6031	5.8952
Stage E	20781	1603	348	161

Insulator leakage current types [16] are described in table 5.

Table 5. *Category of iced insulator leakage current samples.*

Type	Type description
Stage A	Faint discharges, a subtle audible sound, no visible signal
Stage B	Some visible point discharges, a continuous sound
Stage C	Liner weak local arcs
Stage D	Intermittent, stronger local arcs
Stage E	Flash over

Table 6. *Train set.*

Train Set ID	Sample size (piece)
T1	50
T2	500
T3	1000

Experimental data is from artificial experiments and real-measured insulator leakage current. We make several replications to simulate large scale concurrent alarm data. The experiment simulates 6 million monitor points, and by setting up the fault rate(0—100%), simulates the different size of alarm data due to bad weather. In a short period of time, the alarm data size needed to deal with is in the range of zero to 6 million pieces of data. The data set includes raining set and test set as shown in Table 6 and Table 7.

Table 7. *Testset.*

Test Set ID	failure rate	Sample size (kilo pieces)
C1	10%	60
C2	30%	180
C3	50%	300
C4	80%	480
C5	100%	600

4.3. Performance Evaluation

We use the data set in table 5 to test the performance of Spark-KNN. We run the KNN program respectively in single computer, Hadoop cluster and Spark cluster, which named KNN, MR-KNN and Spark-KNN respectively. The MR-KNN and Spark - KNN run at the same hardware environment as shown in table 1.

In a single computer, the run time of KNN changes along with the data size variation as shown in Figure 3.

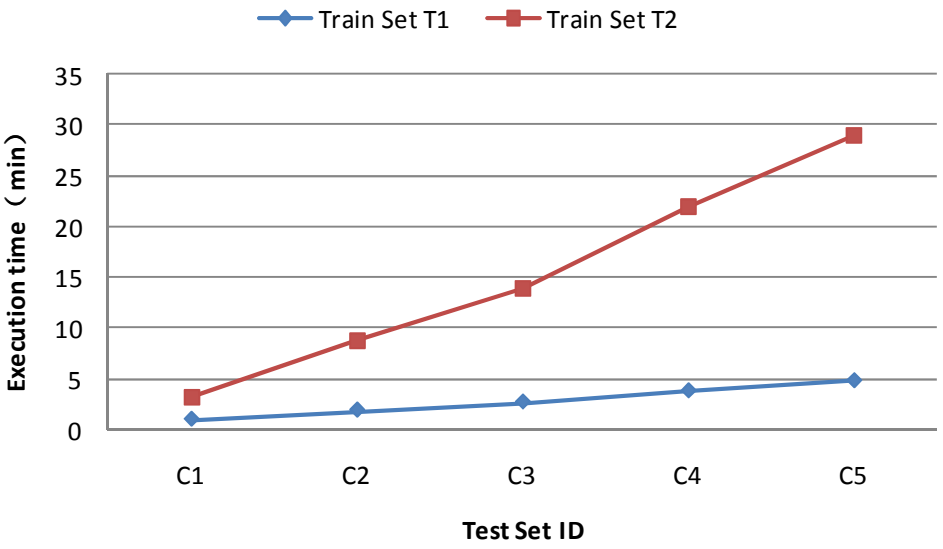
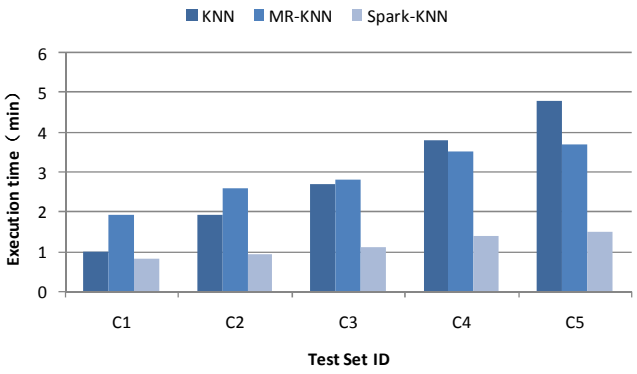


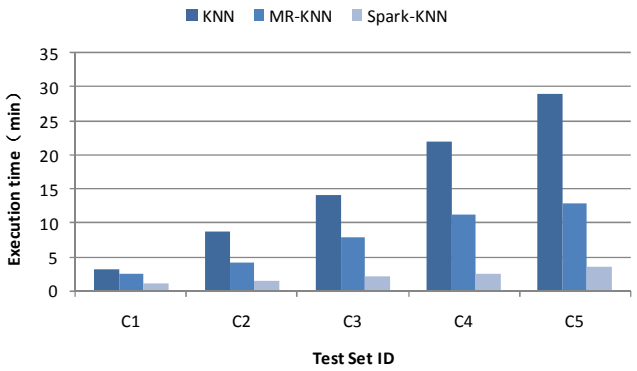
Figure 3. *Execution time of KNN on a single computer.*

As can be seen from the Figure 3, the execution time is close to half an hour when training set is T2 and test set is C5. While choosing training set T3, the execution time is so long that the computer is 'died'. As a result, T3 curve is not drawn in figure 3. The experimental results show that the single computer environment is not up to the fast pattern recognition task of large-scale alarm data.

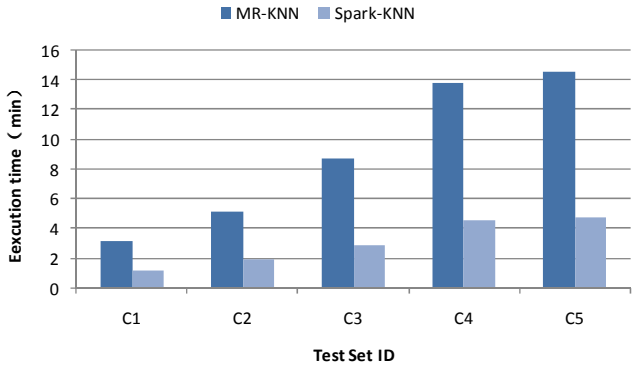
We have measured the execution time of MR-KNN and Spark-KNN by varying the training set, as shown in Figure 4.



a) Training set T1



b) Training set T2



c) Training set T3

Figure 4. *Execution time comparison between Spark-KNN and MR-KNN.*

As shown in Figure 4, the Spark - KNN performance is superior to MR-KNN under various training set. Spark-KNN runs up to 2.97x faster than MR-KNN.

The execution time trend of Spark-KNN under different training sets is shown in Figure 5.

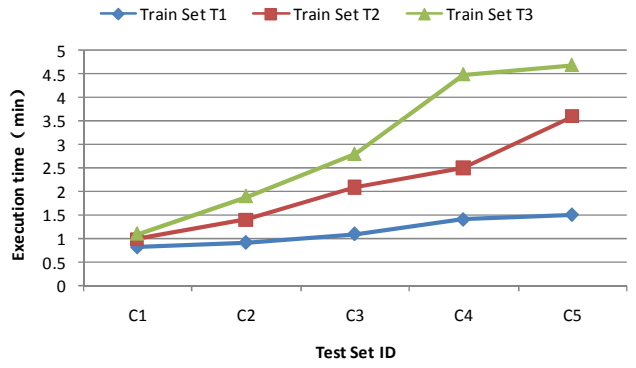


Figure 5. *Execution time trend of Spark-KNN.*

As can be seen from Figure 5, the programexecution time grows slowly as the growth of the test set size.

Speedup is a metric for improvement in performance between two systems processing the same problem. Speedup is calculated by formula (1). T_s denotes the execution time with a single CPU core. T_h denotes the execution time with h CPU cores.

$$S_{peedup} = T_s / T_h \quad (1)$$

We calculate the speedup for Spark-KNN by varying core number of the cluster and using various training set and test set, as shown in Figure 6.

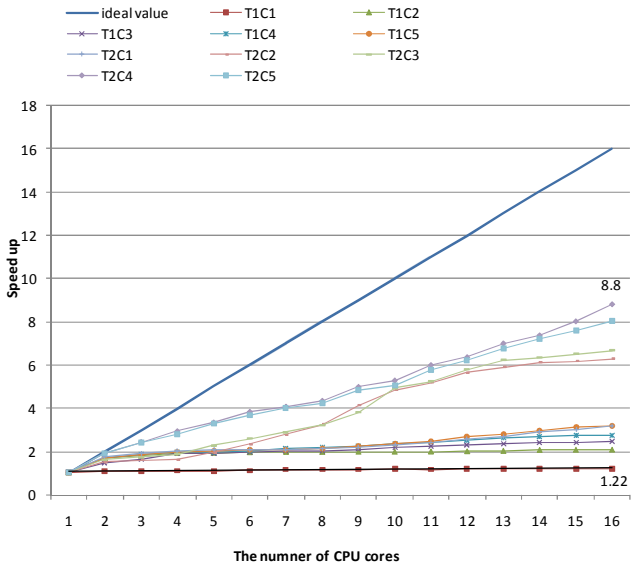


Figure 6. *Speedup of Spark-KNN.*

In Figure 6, the speedup increases with the growth of data scale. The maximum is 8.8 when using the dataset T2 and C4 with 16 CPU cores. The minimum is 1.22 when using the dataset T1 and C1. The speedup does not increase when using dataset T1 and C1 even if add more CPU cores, while increases almost linearly when using dataset T2C4 and T2C5.

5. Conclusion

In this paper, we have investigated how to apply Spark and Cloud computing to insulator leak current data pattern recognition use case in order to understand how well these advanced technologies can accelerate the performance in supporting data-intensive applications. We have adapted the KNN pattern recognition task to Spark program. The prototype was deployed on the Aliyun E-MapReduc cloud computing platform for experimental evaluation. We have used the speed up as the standard metric.

The results from the experiment show that the performance can be improved by using Spark. We have also compared execution time of the Spark and the Hadoop MapReduce. The result shows that Spark-KNN is much faster than MR-KNN and more suitable for real-time data processing for electric power equipment monitoring system.

Acknowledgements

This work was supported by the Fundamental Research Funds for the Central Universities (2016MS117, 2016MS116).

References

[1] Zhou, G., Zhu, Y., Wang, G., & Song, Y. “Real-time big data processing technology application in the field of state monitoring”. Diangong Jishu Xuebao/transactions of China Electrotechnical Society, vol.29, pp. 432-437.

[2] Uri Hasson, Jeremy I Skipper, Michael J Wilde, Howard C Nusbaum, and Steven L Small. Improving the analysis, storage and sharing of neuroimaging data using relational databases and distributed computing. NeuroImage, 39(2):693–706, 2008.

[3] Kai-Wei Chang, Deka, B Hwu, W.-M.W, etc. Efficient Pattern-based Time Series Classification on GPU[C]. 2012 IEEE 12th International Conference on Data Mining (ICDM 2012). Los Alamitos, CA, USA, 2012: 131-40.

[4] Zhao Jun, Zhu Xiaoliang, Wang Wei, etc. Extended Kalman filter-based Elman networks for industrial time series prediction with GPU acceleration [J]. Neurocomputing, 2013, 118: 215-224.

[5] Haimonti Dutta, Alex Kamil, Manoj Pooleery, Simha Sethumadhavan, and John Demme. Distributed Storage of Large-Scale Multidimensional Electroencephalogram Data Using Hadoop and HBase [J]. Grid and Cloud Database Management.2011.9.

[6] White T. Hadoop: The definitive guide [M]. O'Reilly Media, Inc, 2012:260-261.

[7] Christophe Bisciglia. The smart grid: Hadoop at the Tennessee Valley Authority (TVA) [EB/OL]. 2009.6 [2013.2]. http://www.cloudera.com/blog/2009/06/smart-grid-hadoop-tennessee-valley-authority-tva/

[8] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[9] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing [A]. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation[C]. USENIX Association, 2012: 2-2

[10] Zaharia M, Chowdhury M, Das T, et al. Fast and interactive analytics over Hadoop data with Spark[J]. USENIX; login, 2012, 37(4): 45-51

[11] Yan Y, Huang L, Yi L. Is Apache Spark scalable to seismic data analytics and computations? [C]// IEEE International Conference on Big Data. IEEE, 2015.

[12] Shyam R, Bharathi Ganesh H. B, Sachin Kumar S, et al. Apache Spark a Big Data Analytics Platform for Smart Grid[J]. Procedia Technology, 2015, 21:171-178.

[13] Mushtaq H, Al-Ars Z. Cluster-based Apache Spark implementation of the GATK DNA analysis pipeline[C]// Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on. IEEE, 2015:1471-1477.

[14] Ram&#, Rez-Gallego S, Garc&#, et al. Distributed Entropy Minimization Discretizer for Big Data Analysis under Apache Spark [C]// IEEE Trustcom/bigdatase/ispa. IEEE Computer Society, 2015.

[15] Cover, T., Hart, P. Nearest neighbor pattern classification [J]. IEEETrans. Inf. Theory, 1967, 30(1): 21–27

[16] Suda T. Frequency characteristics of leakage current waveforms of an artificially polluted suspension insulator [J]. Dielectrics & Electrical Insulation IEEE Transactions on, 2001, 8(4): 705-709.