Spark: the Next-generation Processing Engine for Big Data

By Dihui Lai and Richard Xu

The concept of "information explosion" was formed more than 70 years ago and the world of big data has evolved ever since. As pointed out by Eric Schmidt (Google CEO), every two days we are creating as much information as we did since the dawn of civilization till 2003. The ever increasing information size has changed the way we store and process data.

Until recently, Hadoop has almost been a paraphrase of big data. The system, famous for its HDFS (Hadoop Distributed File) storage and Map-Reduce processing, has been widely adopted as a tool for big data in IT, health care, financial services, telecommunication and life sciences.

However, the Map-reduce paradigm is not designed for data processing that requires cyclic data sharing, e.g., iterative data processing and interactive data analysis.[1] The invention of Spark, an in-memory data processing engine, seems to bring a solution to the problem.[2] In 2014, Spark was announced as the top-level project of the apache software foundation. Cloudera, a major vendor of Hadoop, considers Spark as a replacement for MapReduce for data execution in their data management system.[3] Spark is also embraced by many big data solution vendors, e.g., Hortonworks, IBM, MapR, etc.

This article gives a general introduction to Spark and shows evidences that Spark could be potentially used as a data processing engine for the insurance industry as well.

DATA STRUCTURES IN SPARK

The core abstraction upon which Spark is built is called Resilient Distributed Dataset (RDD). Basically, RDD is an immutable collection of partitioned records that can be distributed across the nodes of a cluster and operated in parallel. Each partition is a subset of the data it is created from and RDD contains the information on how the partitions are created. If some partitions get lost during a process, they can still be recreated from the original dataset. Therefore, if any nodes in a cluster go down during a large data process, a reconstruction process will be triggered for the lost partition to ensure a successful completion.

Despite its beauty in processing big data, RDD is still a little distal from the data structures that people are familiar with, e.g., SQL schema, data frame. The recent release of Spark introduces DataFrame[4] into its ecosystem. The columnar organized data structure is conceptually similar to a data-frame in R and it also offers relational data operations like SQL. The following are spark codes (in Scala) that create DataFrame from a csv file and perform aggregation on claim counts by states. The 3rd statement appends the claim to the data set by join operations.

```
val df = sqlContext.read.format("com.databricks.spark.
csv").option("header", "true").load("claim_data.csv")

val df_claim_state = df.groupBy("State").agg(count("CLAIM_
CNT"))

val df = df.join (df_claim_state, df("State") === df_
claim_state ("State"), "inner")
```

Equivalently, one can also register the data frame as a SQL table and use SQL-like syntax to do the joining operations, as shown below.

```
df. registerTempTable("claim_data")

sqlContext.sql ("select * from claim_data JOIN
df_claim_state WHERE State claim_data.State = df_

claim_state.State")
```

Both RDD and DataFrame use lazy execution, which means all the operations above will not be executed until some special commands are made, e.g., save, show. The laziness of spark reduces the communication overhead and allows optimization across operations.

SPARK DEPLOYMENT

Spark allows different modes of deployment. It could be deployed with a cluster manager system, e.g., Hadoop YARN, Amazon EC2 or Apache Mesos. Spark also allows a standalone mode by which it can work independent of any cluster management system. It is also possible to run spark on a laptop as a single-node cluster.

The standalone mode is ideal for users to dive into Spark without worrying about the setup of a complicated cluster system. Actually, the standalone mode itself provides a quite powerful tool in dealing with large data of reasonable size. Powerful big data storage systems like HDFS are not necessary for Spark to work. A shared file system, e.g., network file system (NFS) works well for processing data of a large size, e.g., gigabytes as long as the data can fit into a single disk. Spark also integrates well with various databases, including the ones from the NoSQL family, e.g., Cassandra, Hbase and also the ones from the relational database family, e.g., MySQL.

MACHINE LEARNING AND ANALYTICS

As mentioned before, one major limitation of Hadoop's Map-reduce method is that it is not designed for analytics such as machine learning (ML). The in-memory architecture of Spark introduces a nice solution to the problem. By keeping data in memory, Spark allows the users to query data repeatedly and speed up the iterative ML algorithms to a large extent.

Moreover, Spark provides the users with a built-in machine learning library (MLlib). The library covers quite a list of popular ML algorithms, which includes regression (linear/logistic), classification (SVM, naïve Bayes) and clustering (k-means), etc. If one needs an algorithm that is beyond the scope of MLlib, Sparkling-water would be a nice package to add. Sparkling-water is created upon the integration of spark platform and H2O. H2O provides scalable predictive analytics in a wide spectrum, including Generalized Linear Models (GLM), tree algorithms, Gradient Boosting Machine (GBM), Deep Learning, etc.

SPEED AND SCALABILITY

Existing analytical tools such as R or Python do not provide parallel computation for free and are not scalable inherently. Revolution R provides parallelized algorithms but could be very expensive to deploy in a cluster environment. Spark provides an open source platform for analytics and can process large data that could be otherwise hard to handle. Moreover, Spark provides an API for Java, Scala, Python and R. Data scientists who are more familiar with R/Python can dive into the system without much pain.

Generalized linear model (GLM) is widely used in the insurance industry. To understand the potential usage of Spark in insurance, we built GLMs for data of varying sizes and compared the performance difference using terminal server and spark cluster. The terminal server and the spark cluster (7 nodes) had comparable memory sizes in our test. Revolution R showed better performance than regular R regarding the processing speed due to optimized algorithm and parallelism (Table 1). The spark cluster reduced the processing time of the model further due to the involvement of more CPUs. In processing a large data set, the terminal server experienced a memory overflow in processing data 70GB in size while the spark cluster finished the model in about three minutes.

Table 1. Processing data on terminal server and cluster. The processing time of different data on terminal server and spark cluster. Generalized Linear Model is built within two environments on data of different sizes. * we used a GLM routine from the H2O package in the spark cluster.

	Proc Time (Data 1.5 GB)	Proc Time (Data 70 GB)
R (TS)	480.19 s	Memory Overflow
Revolution R(TS)	33 s	Memory Overflow
Spark* (Cluster)	6 s	184 s

To further test the scalability of the spark cluster, we built a GLM model on the same data set while changing the size of the cluster. The processing speed changed depending on the data type and the complexity of the model. Under the given test environment, the spark-cluster showed a close-to-linear scalability where the increase of processing speed was almost proportional to the size of the cluster (Table 2). A 2-nodes cluster failed the task due to memory overflow.

Table 2. Scalability. Comparison between processing time on clusters of different sizes.

Cluster	2-nodes (8 cores)	4-nodes (16 Cores)	7-nodes (28 Cores)
Proc Time	Memory Overflow	300 s	180 s

In summary, spark provides a fast and scalable platform for handling big data. Its in-memory architecture makes it a nice fit for big data analytics. The APIs for multiple languages make it easy to dive-in from various backgrounds. The various deployment modes make it easy to implement into existing big data environments. ■

REFERENCES

1 Zaharia M., Chowdhury M., Franklin M.J., Shenker S. and Stoica I., Spark: Cluster Computing with Working Sets. In *HotCloud*, 2010.

2 Zaharia M., Chowdhury M., Das T., Dave A., Ma J., McCauley M., Franklin M.J., Shenker S. and Stoica I.; Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing; In NSDI, 2012.

3 *http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/spark.html*

4 Bradley J.K., Meng X., Kaftan T., Franklin M.J., Ghodsi A. and Zaharia M. Spark SQL: Relational data processing in Spark; In Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '15), 2015.

Dihui Lai, Ph.D., is a data scientist analyst at RGA Reinsurance Company in Chesterfield, Mo. He can be reached at *dlai@rgare.com.*

Richard Xu, FSA, Ph.D., is VP and actuary, head of Data Science at RGA Reinsurance Company in Chesterfield, Mo. He can be reached at *rxu@rgare.com.*