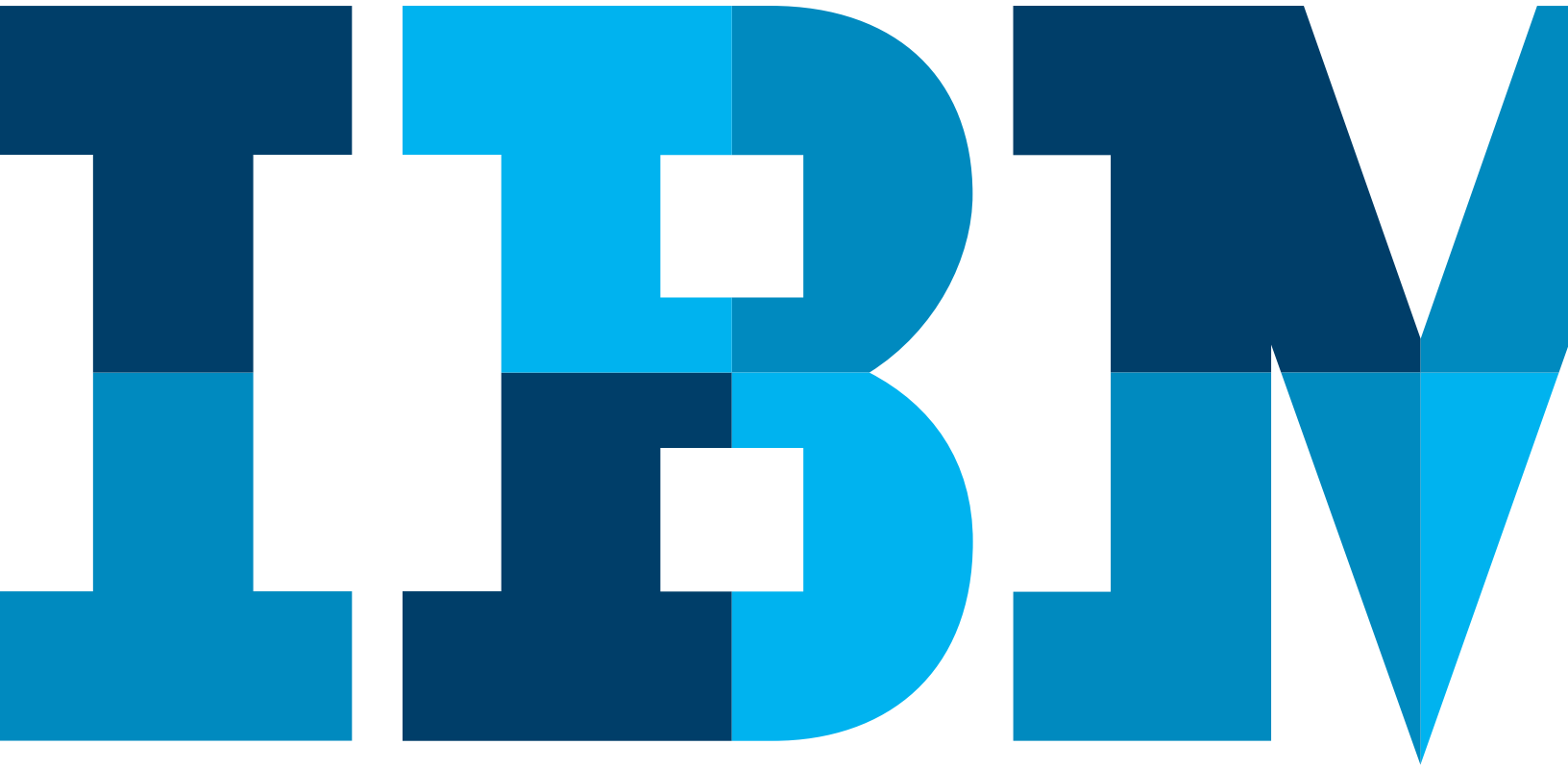
IBM Data Science Experience
White paper

SparkR

Transforming R into a tool for big data analytics



IBM

Executive summary

This white paper introduces SparkR, a package for the R statistical programming language that enables programmers and data scientists to access large-scale in-memory data processing. The R runtime is single-threaded and can therefore normally only run on a single computer, processing data sets that fit within that machine's memory. By providing a bridge to Spark's distributed computation engine, SparkR enables large R jobs to run across multiple cores within a single machine or across nodes in massively parallel clusters, with access to all the memory in the cluster.

Using SparkR, programmers and data scientists can transform R into a tool for big data analytics, taking advantage of parallel processing and near-linear scaling to tackle much larger challenges than would normally be possible. For example, when training a machine-learning model on a very large initial set of data, using SparkR would enable highly parallelized parameter tuning to accelerate the process of finding the best-fit model for the given dataset. With the R processing split across potentially thousands of computational nodes in a cluster, and with access to terabytes of data, this can result in orders-of-magnitude improvements in performance.

Requiring little or no change to existing code, SparkR is an easy way to augment the power of R and scale it up to the challenge of large datasets. Users can keep their existing programs and workflows, seamlessly flexing up to use Spark when distributed processing or large amounts of data are involved, then flexing back down to local resources. In this way, data scientists can easy balance performance and cost, and keep a single language for both local and distributed analysis.

The paper concludes with an invitation for readers to try IBM Data Science Experience, which provides an all-in-one web-based platform for data scientists, enabling the easy use of SparkR and other open-source tools running on robust cloud resources.

Spark — what it is, and why it's great news for data scientists

Apache Spark is an open-source processing engine designed for robust, in-memory machine learning, graph and streaming analytics. Developed to utilize distributed, in-memory data structures to improve data processing speeds for most workloads, Spark can perform up to 100 times faster than Hadoop MapReduce for iterative algorithms or interactive data mining. And while MapReduce requires the relatively low-level Java programming language, Spark also supports Scala, Python and R — which are easier to learn and more commonly used by data scientists. What's more, Spark jobs will run on any Spark environment, from a laptop right up to the very largest clusters, whereas MapReduce jobs are often specifically written for a particular cluster configuration, and are significantly less portable.

Spark provides a single architecture capable of seamlessly handling a wide range of data processing scenarios from complex analytics through algorithmic processing to stream computing. This removes the cost and complexity of maintaining different technology stacks, ensures that metrics are consistent, and makes it faster and easier to share data.

PySpark can work with data in a distributed storage system — for example, HDFS — and it can also take local data and parallelize it across the cluster to accelerate computations. With its ability to compute in real-time, Spark can enable faster decisions — for example, identifying why a transactional website is running slowly so that it can be fixed before financial losses are incurred. Similarly, for queries on streaming data, Spark's real-time processing can help organizations detect financial fraud or distributed denial of service (DDoS) attacks.

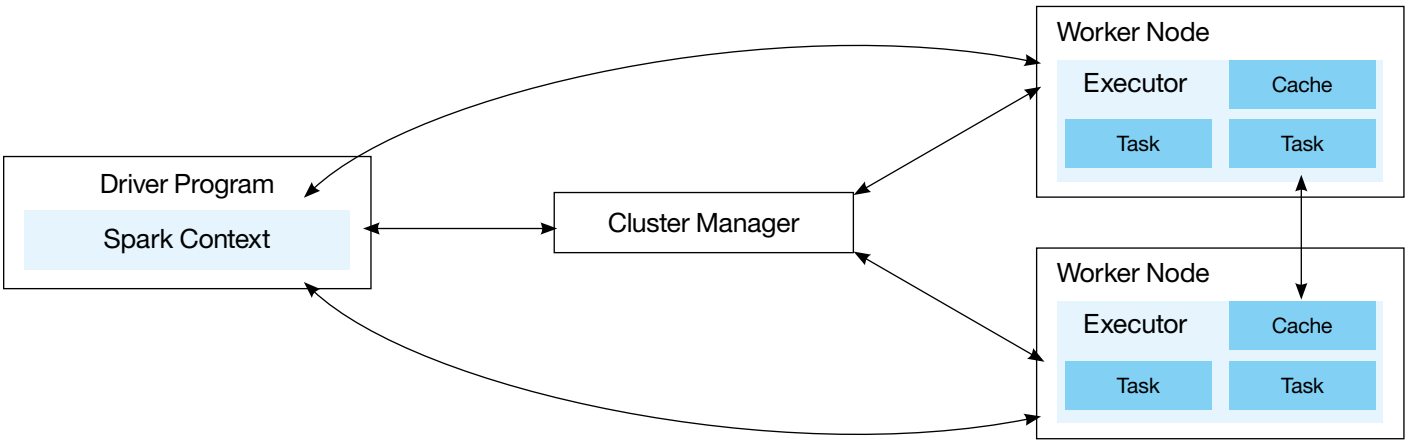


Figure 1: Spark architecture. Spark applications run as independent sets of processes on a cluster, coordinated by the SparkContext object in your main program (called the driver program). From https://spark.apache.org/docs/latest/cluster-overview.html.

What is SparkR, and what benefits does it offer?

SparkR is an R package that provides a light-weight front-end for using Apache Spark directly from R. Interactive data analysis in R is usually limited by the fact that its runtime is single-threaded. In practical terms, this means that it cannot take advantage of multiple cores—let alone multiple machines in a cluster—and it can only process datasets that fit in a single computer's memory. By providing direct access to Spark's distributed computation engine from the R shell, SparkR enables much faster analysis of much larger datasets by splitting the work across multiple cores or cluster nodes, and caching in memory across all the available memory.

As figure 2 shows, SparkR's architecture consists of two main components: an R-to-JVM binding on the driver that allows R programs to submit jobs to a Spark cluster, and support for running R on the Spark executors. The central component of SparkR is a distributed implementation of the standard R data frame. Spark DataFrames (note the different capitalization) are distributed collections of data organized into named columns; using them from SparkR allows multiple Spark nodes to run R operations such as selection, filtering and aggregation in parallel on large datasets. Spark DataFrames can be constructed from a wide array of sources, including structured data files, Hive tables, JSON files, external

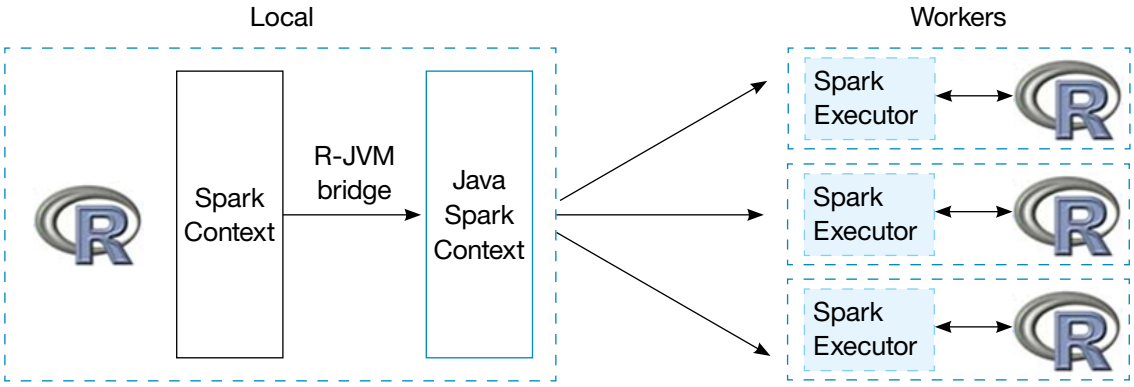


Figure 2: Architectural overview of SparkR.

databases, or existing local R data frames. SparkR also supports distributed machine learning using MLlib, making it easy for R programmers to take advantage of pre-built methods, in particular for solving complex mathematical optimization problems.

Using SparkR, programmers and data scientists can access Spark's scale-out parallel runtime, leverage its input and output formats, and make calls directly into Spark SQL. This means that, rather than being restricted to working only on a single computer, R jobs can now run across multiple cores in single machines or across massively parallel clusters. Spark DataFrames automatically inherit all optimizations made to the computation engine in terms of code generation and memory management.

SparkR exposes the API for the Spark resilient distributed dataset (RDD) as distributed lists in R, enabling input files to be read from HDFS and processed using *lapply* on an RDD. In addition to lapply, SparkR also allows closures to be applied on every partition using *lapplyWithPartition*. Other supported RDD functions include operations such as reduce, *reduceByKey*, *groupByKey* and *collect*.

SparkR automatically serializes the necessary variables to execute a function on the cluster. For example, if global variables are used in a function passed to *lapply*, SparkR automatically captures these variables and copies them to the cluster. SparkR enables easy use of existing R packages inside closures. The *includePackage* command can be used to indicate packages that should be loaded before every closure is executed on the cluster.

To improve performance during the analysis of large datasets, SparkR performs lazy evaluation on DataFrame operations—whereby transformations (for example: *map*, *filter*, *reduceByKey*) are not executed until required by an Action (for example: *count*, *reduce*, *saveAs*). This avoids the materialization of large intermediate results, and minimizes the number of passes over the data during complex multi-stage analyses.

Typical use cases for SparkR

SparkR will appeal to any R users who are constrained by single-threading or lack of memory on local resources. For some analytical tasks, the analysis may be relatively simple but the initial set of data may be too large. In such scenarios, users can use SparkR to perform data cleaning, aggregation and sampling using cluster resources rented in the cloud, then bring the downsampled dataset back to the local resources for building models or performing statistical analysis.

For ensemble learning, parameter tuning or bootstrap aggregation, users will likely need to execute a particular function in parallel across different partitions and then combine the results from each partition using an aggregation function. Using SparkR to run partition aggregate workflows could dramatically increase the speed in such scenarios. Even where the amount of input data is relatively small, using SparkR to parallelize the workflow will help in cases where the data is evaluated with a large number of parameter values.

Finally, when using machine learning algorithms it may be vital to use as much raw data as possible rather than a downsampling—when training a machine learning system, or when building and training a recommendation engine, for example. In such scenarios, SparkR can handle the pre-processing of the data to generate the training features, giving the labels as input to a machine learning algorithm (for example, one of those accessible to Spark via the MLlib library). The algorithm will then fit the data to a model, typically creating a much smaller entity suitable for running back on local resources to serve predictions. Performing the required cleaning, filtering and aggregating on a large volume of data would be very difficult in R. Equally, model training is slow on R, given its single-threaded nature. Model training involves applying a number of parameters to learn which parameter value works best. SparkR enables this parameter tuning to run in parallel across multiple nodes in a cluster, potentially improving performance by several orders of magnitude when the task is to determine the best model for a given dataset.

In practical terms, IBM Data Science Experience, which provides an all-in-one web-based platform for data scientists, makes it easy to use SparkR on cloud resources without needing to set up clusters manually. This avoids all the potential headaches involved in getting corporate IT functions to set up the hardware with the correct software and tools—IBM Data Science Experience will automatically provide fully functional remote clusters so that data scientists can simply fire up the resources they want when they need them.

The seamless binding between R and Spark makes it possible to have a hybrid approach to analytics: using local resources when the datasets and the computational tasks are appropriately sized, and flexing up to use distributed processing and large memory resources in a rented cluster whenever required. This approach also keeps control of the costs, which are naturally higher for a large computational cluster than for a single local machine.

Using SparkR

The Spark shell provides an interactive way to quickly prototype operations without having to develop a full program, package it and then deploy it. Users can choose to connect R programs to Spark clusters from Rstudio, the R shell, Rscript and other R IDEs.

The entry point into SparkR is the SparkSession, which connects R programs to Spark clusters and enables users to add options such as the application name, dependent spark packages and so on. The SparkContext is what actually allows the Spark application to use computational resources in a cluster (either via a resource manager such as YARN, or using Spark's own cluster manager). To create a SparkContext, users must first create a SparkConf, which stores the configuration parameters that the Spark driver application will pass to SparkContext. Some of these parameters define properties of the Spark driver application and some are used by Spark to allocate resources on the cluster. Spark also allows users to create empty SparkConf files so that the configuration can be defined dynamically.

For structured data processing, users can take advantage of the Spark SQL module, which uses Spark to distribute SQL queries across nodes in a cluster. Unlike the standard Spark RDD API, the interfaces in Spark SQL include information about the structure of the data and the computation, and this information is used to perform additional optimizations.

Spark uses the same execution engine regardless of the API or language used to express the computation. This provides enormous flexibility, enabling developers to choose their preferred way to express each transformation, and potentially mix and match different approaches within the same project.

Get up and running with SparkR

To get started with SparkR, please try IBM Data Science Experience, which provides a single point of access and control across multiple open-source data-transformation and analytics tools running on robust cloud resources. For more information, please visit datascience.ibm.com

IBM Corporation
Route 100
Somers, NY 10589

Produced in the United States of America
December 2016

CDW12367-USEN-00