Accelerating R analytics with Spark and Microsoft R Server for Hadoop

Bill Jacobs
Microsoft Advanced Analytics Product Marketing

July 2016

Microsoft

Abstract

Analysts predict that the Hadoop market will reach $50.2 billion USD by 2020.[1] Applications driving these large expenditures are some of the most important workloads for businesses today including:

- Analyzing clickstream data, including site-side clicks and web media tags.
- Measuring sentiment by scanning product feedback, blog feeds, social media comments, and Twitter streams.
- Analysis of behavior and risk by capturing vehicle telematics.
- Optimizing product performance and utilization by gathering data from built-in sensors.
- Tracking and analyzing people and material movement with location-aware systems.
- Identifying system performance and intrusion attempts by analyzing server and network log.
- Enabling automatic document and speech categorization.
- Extracting learning from digitized images, voice, video, and other media types.

Predictive analytics on large data sets provides organizations with a key opportunity to improve a broad variety of business outcomes, and many have embraced Apache Hadoop as the platform of choice.

In the last few years, large businesses have adopted Apache Hadoop as a next-generation data platform, one capable of managing large data assets in a way that is flexible, scalable, and relatively low cost. However, to realize predictive benefits of big data, organizations must be able to develop or hire individuals with the requisite statistics skills, then provide them with a platform for analyzing massive data assets collected in Hadoop "data lakes."

As users adopted Hadoop, many discovered performance and complexity limited Hadoop's use for broad predictive analytics use. In response, the Hadoop community has focused on the Apache Spark platform to provide Hadoop with significant performance improvements. With Spark atop Hadoop, users can leverage Hadoop's big-data management capabilities while achieving new performance levels by running analytics in Apache Spark.

What remains is a challenge—conquering the complexity of Hadoop when developing predictive analytics applications.

In this white paper, we'll describe how Microsoft R Server helps data scientists, actuaries, risk analysts, quantitative analysts, product planners, and other R users to capture the benefits of Apache Spark on Hadoop by providing a straightforward platform that eliminates much of the complexity of using Spark and Hadoop to conduct analyses on large data assets.

Summary: accelerating predictive analytics with R, Hadoop, and Spark

Microsoft R Server provides R users with the performance and scale needed when tackling statistical analyses on big-data assets. Microsoft R Server for Hadoop enables R users to conduct the full range of data exploration, transformation, feature engineering, and predictive modeling on large data assets stored in Hadoop, without becoming Hadoop experts themselves. With the latest edition of R Server for Hadoop, users can now multiply these benefits with R, Hadoop, and Spark, which together deliver:

- A startling performance improvement over MapReduce.
  - Six-fold performance improvement over R Server with YARN MapReduce
  - Allocate Spark resources to your analytics workloads more flexibly than MapReduce
- New capabilities for R users.
  - Analyze terabyte-class data sets without open-source R's memory limitations
  - Multiple orders-of-magnitude performance increases over open-source R algorithms
  - Achieve speed gains without manual parallel programming using R Server's Parallel External Memory Algorithms (PEMAs)
  - Simplify resource management and workload allocation for multiuser teams
- New capabilities for Spark users.
  - Run R analytics on Spark just as they run on other supported platforms including Windows and Linux
  - Nearly double the performance of the Spark MLlib algorithms accessible from R
  - Expand your analytical capabilities far beyond MLlib, entirely accessible from R
- Minimization of future disruptions for data science users.
  - Stabilize your data science development ecosystems across architectures including Hadoop, SQL Server, Windows, Linux, and others
  - Operationalize R analytics by making analytic services accessible to other platforms
  - Assure continuity via a commercial support team specializing in R

Background: why R?

Roughly coincident with the adoption of Hadoop as a big-data platform, the R language has emerged as the industry standard for open-source predictive analytics. Recent measures of R usage, such as the 2016 KDnuggets Software Poll,[2] show that 49 percent of survey respondents use the R language as their primary analytics language.

There are many capabilities in R that account for much of its growing popularity. Flexibility, rich graphics, and statistics-oriented capabilities are certainly of note. But it is R's broad open-source ecosystem of more than 8,000 freely available software packages and R's worldwide user community of over 2.5 million users that account for much of its growth. The R community, supported by many academic institutions that teach R, aided by an open-source business model, has brought businesses a way to rapidly grow their analytics capabilities by tapping an already large and growing talent pool of R users.

Early integration of R and Hadoop

In 2011, Revolution Analytics introduced one of several popular early methods for integrating R and Hadoop. The RHadoop open-source project, a popular project on GitHub, enabled R users to run R workloads in Hadoop by "injecting" R code into mappers and reducers using Hadoop Streaming.

While powerful, RHadoop required developers of R scripts to design their scripts to manage parallelization of computations. As a result, most users of RHadoop applied it to "embarrassingly parallel" tasks like transformation and model scoring. In the minority were those users who deployed RHadoop to parallelize modeling computations due to the difficulty of parallelizing their modeling algorithms across multiple nodes.

Recognizing the opportunity to apply the parallelism of Hadoop to achieve new levels of data scale, Revolution Analytics introduced enhanced versions of its flagship product, Revolution R Enterprise (RRE), for use within Hadoop. RRE enabled data scientists to conduct R analyses including transformation, exploration, visualization, and, most importantly, modeling in parallel using Hadoop MapReduce. This brought R users ease of use—no longer requiring them to design their own parallelized algorithms to run in Apache Hadoop.

Microsoft, having long recognized the potential of R-based analytics for both on-premises and cloud-based systems, was well along in using the R language to support Bing, Xbox, Office 365, and Azure Machine Learning. In 2015, Microsoft purchased Revolution Analytics and hired nearly its entire staff to drive forward on Microsoft's increasingly rich vision for on-premises and cloud-based advanced analytics using the open-source R language.

Simplifying R on Hadoop

By 2015, Revolution Analytics had pioneered what has now become Microsoft R Server for Hadoop. Originally designed for MapReduce, R Server for Hadoop has been deployed into a number of customer analytics scenarios, including:

- A life-sciences company that delivers recommended treatments to its customers based on user-specified details of planned usage.
- A leading US bank that has developed an innovative prediction engine.
- A major card issuer and transaction processor that is moving large portions of its analytics to R from another platform.
- A marketing-sciences company that analyzes billions of cookies and session records in a matter of hours instead of days.

Microsoft R Server does not simply connect to Hadoop—it transparently parallelizes R analytical computations inside of Hadoop on behalf of R users.

But the Hadoop community moves fast. Attention has shifted to Apache Spark, which pioneered in memory computing for clustered systems.

Even faster R: R analytics in Spark on Hadoop

Claiming thousands of contributions from hundreds of companies, the Apache Spark project enjoys one of the widest bases of adoption of any open-source project since Linux. As attention has shifted to Spark, so has the opportunity to run R analytics inside of Spark.

Approximately two years ago, Revolution Analytics began experimenting with Spark with strong results. Microsoft R Server for Hadoop has been upgraded to support Apache Spark to bring Spark's performance benefits to R users. Microsoft R Server version 8.0.5, released June 2016, makes support of Apache Spark on Hadoop generally available to R users via four different Apache Spark platforms:

- Hortonworks Hadoop
- Cloudera Hadoop
- MapR Hadoop
- HDInsight Hadoop in the Azure Cloud

The remainder of this white paper explains in detail the architecture we selected for combining the power of Microsoft R Server and Apache Spark, so that you may engage with your R user population and your IT staff in a discussion about the use of Apache Spark as an even faster platform for big-data analytics.

Microsoft R Server delivers speed, scale, and stability for R

Microsoft R Server provides big-data analytics built with open-source R at its core. R Server extends the capabilities and enhances the performance of R with:

- Performance-enhanced, CRAN-compatible distribution of open-source R, called Microsoft R Open.
- Fast, scalable analyses with the ScaleR Parallel External Memory Algorithm library.
- Multiple-platform portability with DistributedR platform integration.
- Integration of diverse data types with ConnectR data connectors.
- Application integration using the DeployR integration gateway.

Key to the performance and scale capabilities of Microsoft R Server is the ScaleR package of high-performance Parallel External Memory Algorithms (PEMAs). PEMAs scale and accelerate R analyses on data sets far larger than available memory through a combination of block-by-block analysis, distribution of processing across multiple cores, sockets and nodes, compiled code, and mathematically optimized algorithms.

Exploiting analytical parallelism for big data

To easily explain how Microsoft R Server scales R to large data assets, let's look first at the realities of the big-data world in which we live:

- *Parallel systems are here to stay.* In order to harness the power of massively parallel systems, analytics must adapt to be easily sealed using parallelism. Additionally, these systems are large shared assets and must be accessible by many users at once.
- *More memory is not a panacea.* Memory prices and capacities are not dropping fast enough to serve the needs of big-data analytics. As a result, analytical algorithms must be refactored and redesigned to operate on entire data sets, but do so with only a fraction of the subject data set in memory at any given time.
- *Data growth will exceed bandwidth growth.* With data growing several times faster than available network bandwidth, avoiding bottlenecks requires architectures that can move statistical computation to the data, rather than moving the data itself.
- *R excels at ease of use, others excel at speed.* While R brings broad adoption by data scientists, it is not as fast as other computer science languages. To maximize compute speed, algorithms need to do the bulk of big-data processing using computations rewritten into faster languages.

- *Portability is critical to long-term value.* No one platform will serve all purposes or last forever. We must anticipate that many R-based analyses will produce scripts that outlast the platforms on which they were built. We must also accommodate platform diversity. Users will need to explore and model using small data sets on individual workstations, retraining models on big-data platforms and deploying models to large servers and server clusters.

With these things in mind, Revolution Analytics pioneered a framework and a set of algorithms to run within that framework that provide for these changes. We call these algorithms Parallel External Memory Algorithms, or PEMAs. PEMAs share five characteristics:

- *Parallelism:* PEMAs are rewritten to compute results on chunks of data, using multiple threads, cores, and sockets and nodes.
- *External memory:* PEMAs act only on chunks of data at a time, keeping the entire corpus of a large data set in main storage and bringing only needed "chunks" into memory at any one time.
- *Remote capable:* PEMAs are capable of computing an algorithm using local resources on a local data set, or shipping the request to another remote system to compute on a data set resident at that system.
- *Language independent:* PEMAs are written for use by R scripts, but are not themselves written in R. Most PEMAs at the core are written in C++, maximizing computation through use of a compiled language.
- *Write Once Deploy Anywhere portability:* PEMAs are platform independent. They utilize resources available from, but do not depend on, features of any one platform. Parallelism is simply a collection of abstract tasks, whether implemented as threads on a Windows machine, multiple tasks in SQL Server, Table Functions in Teradata Database, or Spark Executors on a large cluster.

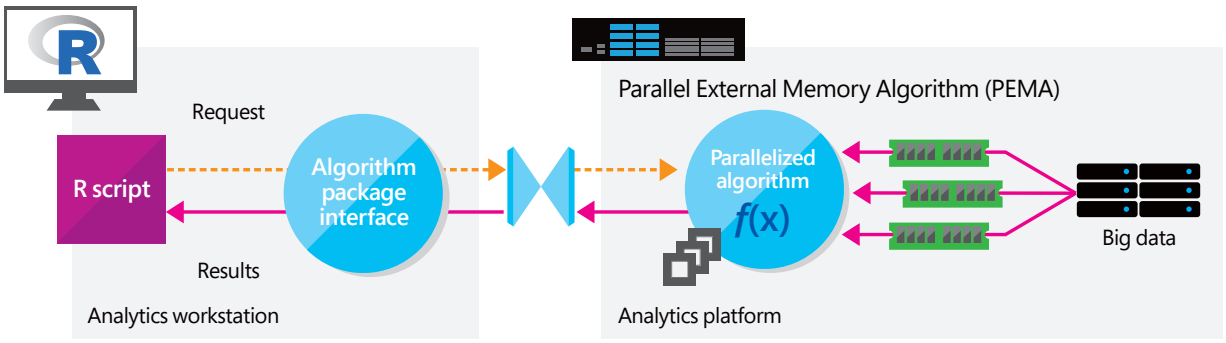

Figure 1: Parallelized remote execution of statistical algorithms using PEMAs

Figure 1 diagrams three of the five features of PEMAs: parallelism, external memory for "chunk-wise" data ingest, and remote execution. The two additional features not shown are portability, allowing a PEMA to work the same whether hosted in Windows workstations or Apache Hadoop clusters or another system, and language independence, where the actual algorithm can be written in languages other than R.

Operationalizing R analytics

Microsoft R Server for Hadoop reduces the time and complexity of deploying R analytics into production environments. Microsoft R Server includes DeployR, an enterprise deployment framework that allows authenticated users to publish R functions (such as those for model execution, script execution, and data visualizations) as services for wide accessibility. Via these services, developers can integrate real-time R analytics into a broad array of tools and platforms. DeployR enables two-way web services–based integration with popular tools including Excel, Qlik, Tableau, Power BI, and many others.

In brief, Microsoft R Server for Hadoop offers a scalable, high-performance platform for the rich capabilities of R. It offers true cross-platform integration together with a wide choice of user interfaces and deployment options. And now, it offers the very significant performance and capability advantages of Apache Spark.

Microsoft R Server affords users investment protection

R Server's Write Once Deploy Anywhere (WODA) architecture helps preserve investments in R analytics by avoiding potential disruptions as fast-changing platforms like Hadoop and Spark evolve.

With R Server's WODA architecture, users can develop scripts and predictive models on one platform and then deploy them to any other platform supported by Microsoft R products, such as SQL Server R Services or R Server running on Windows and Linux-based workstations or servers, data warehouse appliances, and Hadoop clusters running on-premises or in the cloud.

Over the past two years, Revolution Analytics and now Microsoft have enhanced and expanded the Microsoft R Server product to stabilize, accelerate, and operationalize R analytics on all of the following platforms:

- Microsoft Windows and Microsoft Windows Server
- Red Hat, SUSE, and CentOS Linux
- Microsoft SQL Server
- Teradata Database
- Hadoop MapReduce and YARN MapReduce from Cloudera, Hortonworks, and MapR

From 20,000 feet…

Before diving into the implementation details linking Microsoft R Server with Apache Spark, let's talk first about the goals of the project.

- *Preserve R Server's ease of use:* R Server simplifies big-data analytics by parallelizing analytical algorithms internally. This removes the burden of parallel algorithm design from the R user, focusing their attentions back on the problem of using, not designing and parallelizing, analytical and statistical algorithms.
- *Preserve R Server's portability:* R Server users enjoy portability between dissimilar platforms today, and have a greater likelihood of moving to new platforms in the future because their R scripts are portable. R Server's new Spark capability continues this portability, enabling R scripts written for other platforms to be easily run on Apache Spark.
- *Scale R Server's performance:* The key promise of Apache Spark is significant speed improvement over Hadoop MapReduce through the use of in-memory execution. R Server's new Spark capability uses Spark's in-memory computational speeds to greatly improve R computations without changing R scripts.

Details of R Server integration with Apache Spark

Microsoft has added Spark support to maximize the speed of analytics by porting R Server's Parallel External Memory Algorithms (PEMAs) to function compatibly within the Spark environment. The remainder of this paper describes the details of the integration architecture that brings R users continued ease of programming, portability, and even greater speed by running in Spark.

Let's first look at how Spark works. In Figure 2, Spark's main elements are Spark Driver processes and Spark Executor processes. Together they load information into a new memory-based construct called Resilient Distributed Data sets, or RDDs. RDDs are a means of accessing data very quickly on many nodes by storing portions of the data set in RAM spread across many nodes.

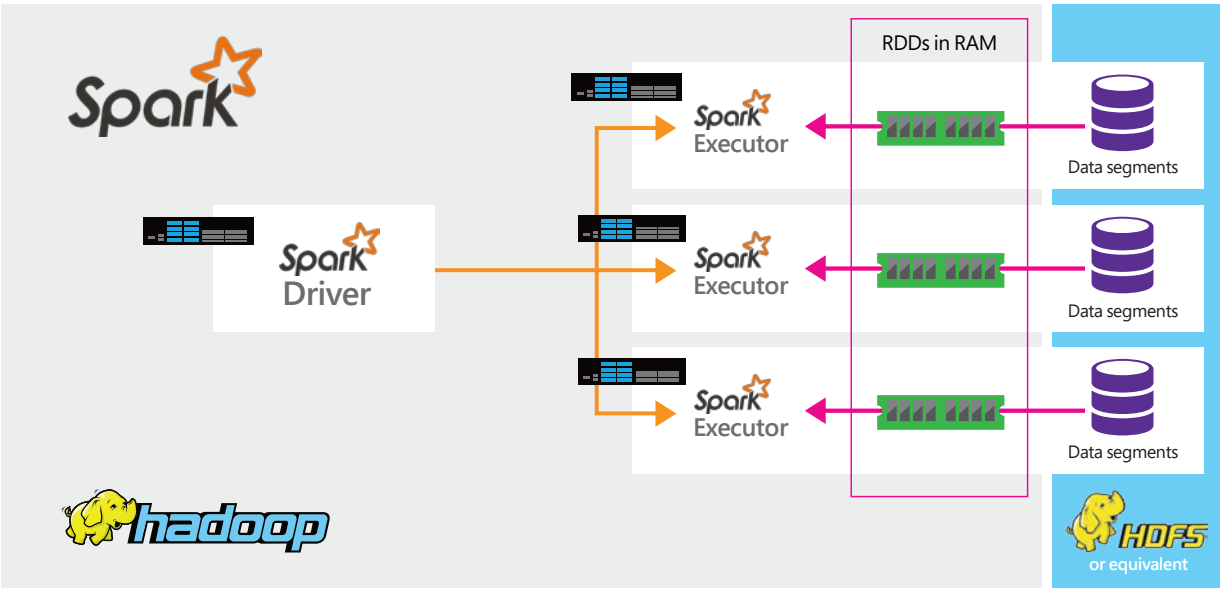

Figure 2: Generalized Spark parallel computation

With data sets to be analyzed loaded into large numbers of RAM blocks in many nodes, processing steps to access, transform, and analyze data are run on each node, called Spark Executors. Upon completion, results from multiple nodes are combined for return to the Spark Driver and thence to the calling process. In this way, Spark distributes work to many nodes like MapReduce, but holds the data to be analyzed in RAM instead of on disk, for far greater performance.

Other differences between Spark and MapReduce further accelerate work and include techniques like "lazy execution," which combines multiple tasks to be performed on a single pass for optimal efficiency. Spark also provides programmers with access to persistency, reusing Spark Executors where possible to avoid the many seconds of delay required if they must be restarted.

Integrating R Server's ScaleR PEMAs with Spark

Bringing the inherent parallelization of Microsoft R Server to Spark was accomplished by:

- Enhancing the R Server for Hadoop master process to schedule work to run in Spark on YARN in addition to the YARN MapReduce engine.
- Extending the DistributedR abstraction layer that supports the ScaleR algorithm individual processing steps so that ScaleR algorithms can run within Spark Executors.

The resulting architecture is shown in Figure 3.

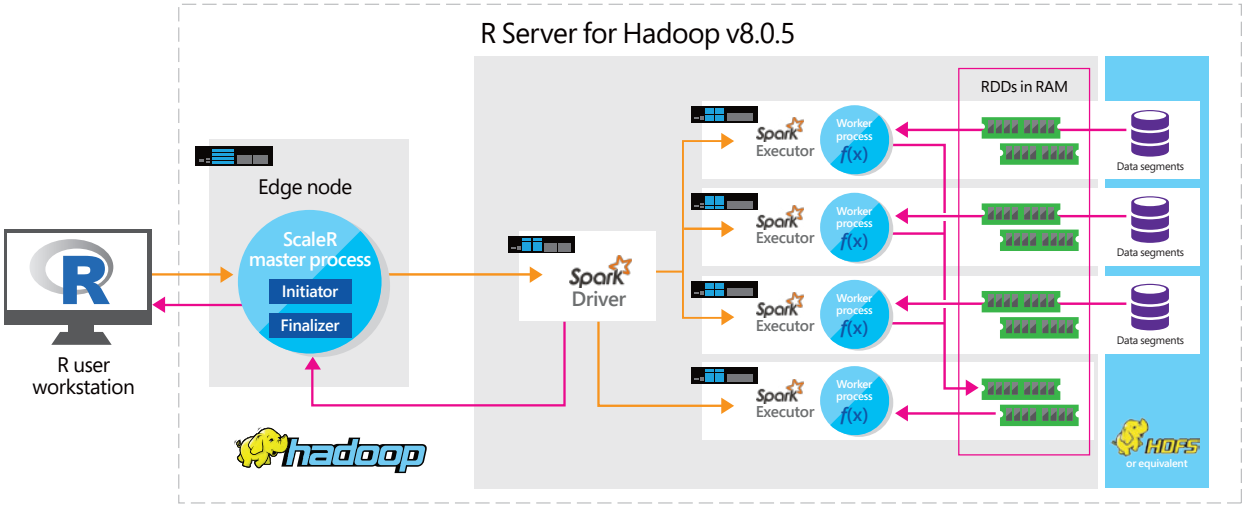

Figure 3: Combining R Server's ScaleR algorithms with Spark Drivers and Executors

In Figure 3, Microsoft R Server for Hadoop is installed on edge and worker nodes. The ScaleR master process is typically installed on an edge node of the Hadoop cluster for all but the smallest development clusters. The master process coordinates parallelization of work across the Spark cluster.

Microsoft R Server's master process can be configured to run on any node of the cluster. However, for production or performance-critical systems, users should configure a separate edge node to host the master process in order to maintain the workload balance across the cluster.

Microsoft R Server ScaleR algorithms can be run in Spark from a workstation using Microsoft R Client or from a local R instance started on the edge node itself.

For execution of an R Server algorithm from either a remote R Client session or from a local R instance, processing follows the following steps.

1. Within the user's R script, three actions are taken to initiate Spark-based analytics:
    a. Specify the data source to be ingested.
    b. Set the remote execution context to "RxSpark," identifying the target Spark on Hadoop instance for R Server to use.
    c. Call the desired ScaleR algorithm.
2. Once the ScaleR algorithm is called by the script, the local algorithm "stub" checks the execution context setting and directs execution to the specified local or remote platform.
3. When running analyses in the RxSpark remote context, the algorithm stub packages input parameters and input file specification passed by the R script and ships them to the ScaleR master process.

4. The master process unpacks the parameters and input file specification provided by the writer of the R script.

5. Processing of the algorithm within Spark proceeds as follows:

    a. The master process creates a Resilient Distributed Data set (RDD) using Spark, into which the subject data set is loaded by Spark.

    b. The master process schedules a Spark job to execute the initial step of the requested ScaleR algorithm.

6. To parallelize algorithm computation, Spark schedules Spark Executors that each consume one or more RDD segments and then invoke the ScaleR worker process on affected nodes, passing the RDD segment.

7. Each ScaleR worker process consumes its inbound RDD segment, applies the logic step described by the processing instructions, and produces:

    a. An Intermediate Results Object (IRO) containing interim results from processing a single data segment.

    b. In the case of transformation or scoring, output data is written to HDFS-compatible storage during segment processing.

    Once all segments of the RDD have been processed by ScaleR worker processes and IROs produced, Spark schedules one or more Executors to run the "reduce" component of the ScaleR algorithm selected, which consolidates individual IROs into a Final Results Object (FRO).

8. The ScaleR master process evaluates and acts upon the FRO:

    a. For single-step algorithms (for example, linear regression), the master process prepares the FRO for return to the calling R script and releases RDDs and other resources held on behalf of the algorithm.

    b. For iterative algorithms (for example, logistic regression), the master process tests the FRO for convergence. If not sufficiently converged and if the maximum number of iterations hasn't been reached, the master process prepares a new instructions object and starts another round of Spark Executors to iterate over the data, repeating steps 5 through 7.

    c. For multiple-step algorithms (for example, clustering), the master process prepares a new instructions object for the next step in processing the data, and schedules Spark to execute the next step, repeating steps 5 through 7.

9. The ScaleR master process, following completion, iteration, or multiple-step execution, packages the Final Results Object for return to the calling R script, whether on a user workstation or locally on the edge node.

10. The results of the algorithm operation are put into the correct response format for the calling script and the algorithm completes.

R Server augments speed with portability, reliability, and ease of use

The design of R Server for Hadoop and its Spark support maximize the effectiveness of the R users because:

- *R Server for Hadoop is compatible with other R Server products.* R Server is designed for compatibility across versions and across future editions. R scripts written for other platforms can be easily moved to modify, explore, transform, model, and score data sets in Hadoop HDFS-compatible storage using Spark, just as R Server does on Windows or Linux.
- *R Server is designed to be easy for R users.* R scripts written for R Server on other platforms can easily be used to run analytics in Spark and the reverse. To accomplish this, R Server for Hadoop transparently manages parallelization of PEMA algorithms and the creation and deletion of all RDDs needed.
- *Reducing data movement speeds processing and reduces security exposures.* During processing with R Server and Hadoop with Spark, subject data remains in place in HDFS or another storage subsystem. As users perform analyses, computations are conducted on the node closest to the data, dramatically reducing the time needed to build and deploy predictive models, and reducing security issues from data movement or replication beyond the Hadoop cluster.
- *Data sets larger than available memory can be analyzed.* When a data set exceeds available memory for RDDs, R Server and Spark transparently manage block-wise loading so that user coding needn't change.

Frequently asked questions

Q. *What input sources does Microsoft support for use with Microsoft R Server in Spark?*

A. Microsoft R Server for Hadoop ingests data sets from Hadoop HDFS-compatible storage into Spark Resilient Distributed Data sets (RDDs) transparently.

Q. *What data input formats can we use?*

A. For Hadoop users, Microsoft R Server supports text files in CSV or a fast proprietary format called XDF. XDF is an efficient binary-serialization format used by Microsoft R products to accelerate data access and manipulation. For data residing in other systems, Microsoft R Server supports many other formats, including both SAS and SPSS files.

Q. *Where can data be output?*

A. Data scored or transformed in R Server is written to HDFS. Models and aggregates produced are returned to the calling script where they can be run in R, deployed to the DeployR gateway, or exported in PMML.

Q. *What formats can be used to return results?*

A. Data created such as transformed columns or scored columns can be written to CSV or proprietary XDF files in HDFS. Trained predictive models are returned to the R script as R objects that can be saved, serialized, or exported to PMML.

Q. *How many Spark Executor tasks are run?*

A. Microsoft R Server for Hadoop schedules each pass through the data using one Spark Driver request, in turn scheduling enough Spark Executor tasks to consume all blocks of the subject data set. The number of blocks and therefore Executors is a product of system settings, in particular the segment size for the subject HDFS file.

Q. *How many reducer tasks does Spark run?*

A. Microsoft R Server for Hadoop schedules one or more reducer tasks as needed if consolidated results—models or aggregates—are to be returned. Additional reducer tasks are scheduled only for very large files. When scoring data, no reducer is required as the first pass of scoring writes resulting scores directly to HDFS-compatible storage.

Q. W*hat happens if a Spark Executor fails?*

A. Failure of a Spark Executor is handled by Spark itself—another Spark Executor will be scheduled, and any results objects and/or RDD segments produced will be recreated.

Q. *How is data scored using a model?*

A. Microsoft R Server offers users two options for model scoring. Most commonly, users run scoring natively using the rxPredict function. This function uses model objects produced by any of the ScaleR predictive modeling functions to score each data record in an input file. rxPredict then writes the scores to a file in HDFS. In addition, users can export model objects in Predictive Model Markup Language (PMML) for deployment into third-party PMML-based scoring engines.

Endnotes

[1] "World Hadoop Market - Opportunities and Forecasts, 2020." Allied Market Research. March 2014. http://www.alliedmarketresearch.com/hadoop-market.

[2] "R holds top ranking in KDnuggets software poll." Revolutions Blog. June 13, 2016. http://blog.revolutionanalytics.com/2016/06/r-holds-top-ranking-in-kdnuggets-software-poll.html.