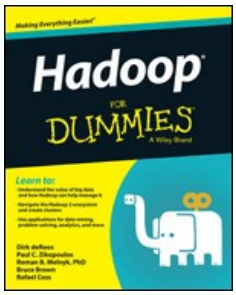

Hadoop for Dummies

by Dirk deRoos et al.

Chapter 9: Statistical Analysis in Hadoop

In This Chapter

- Scaling out statistical analysis with Hadoop
- Gaining an understanding of Mahout
- Working with R on Hadoop

Big data is all about applying analytics to more data, for more people. To carry out this task, big data practitioners use new tools — such as Hadoop — to explore and understand data in ways that previously might not have been possible (problems that were "too difficult," "too expensive," or "too slow"). Some of the "bigger analytics" that you often hear mentioned when Hadoop comes up in a conversation revolve around concepts such as machine learning, data mining, and predictive analytics. Now, what's the common thread that runs through all these methods? That's right: they all use good old-fashioned statistical analysis.

In this chapter, we explore some of the challenges that arise when you try to use traditional statistical tools on a Hadoop-level scale — a *massive* scale, in other words. We also introduce you to some common, Hadoop-specific statistical tools and show you when it makes sense to use them.

Pumping up Your Statistical Analysis

Statistical analytics is far from being a new kid on the block, and it is certainly old news that it depends on processing large amounts of data to gain new insight. However, the amount of data that's traditionally processed by these systems was in the range between 10 and 100 (or hundreds of) gigabytes — not the terabyte or petabyte ranges seen today, in other words. And it often required an expensive *symmetric multi-processing* (SMP) machine with as much memory as possible to hold the data being analyzed. That's because many of the algorithms used by the analytic approaches were quite "compute intensive" and were designed to run in memory — as they require multiple, and often frequent, passes through the data.

The Limitations of Sampling

Faced with expensive hardware and a pretty high commitment in terms of time and RAM, folks tried to make the analytics workload a bit more reasonable by analyzing only a sampling of the data. The idea was to keep the mountains upon mountains of data safely stashed in data warehouses, only moving a statistically significant sampling of the data from their repositories to a statistical engine.

While sampling is a good idea in theory, in practice this is often an unreliable tactic. Finding a statistically significant sampling can be challenging for sparse and/or skewed data sets, which are quite common. This leads to poorly judged samplings, which can introduce outliers and anomalous data points, and can, in turn, bias the results of your analysis.

Factors That Increase the Scale of Statistical Analysis

As we can see above, the reason people sample their data before running statistical analysis is that this kind of analysis often requires significant computing resources. This isn't just about data volumes: there are five main factors that influence the scale of statistical analysis:

- This one's easy, but we have to mention it: the volume of data on which you'll perform the analysis definitely determines the scale of the analysis.
- The number of transformations needed on the data set before applying statistical models is definitely a factor.
- The number of pairwise correlations you'll need to calculate plays a role.
- The degree of complexity of the statistical computations to be applied is a factor.
- The number of statistical models to be applied to your data set plays a significant role.

Hadoop offers a way out of this dilemma by providing a platform to perform massively parallel processing computations on data in Hadoop. In doing so, it's able to flip the analytic data flow; rather than move the data from its repository to the analytics server, Hadoop delivers analytics directly to the data. More specifically, HDFS allows you to store your mountains of data and then bring the computation (in the form of MapReduce tasks) to the slave nodes.

The common challenge posed by moving from traditional symmetric multi-processing statistical systems (SMP) to Hadoop architecture is the locality of the data. On traditional SMP platforms, multiple processors share access to a single main memory resource. In Hadoop, HDFS replicates partitions of data across multiple nodes and machines. Also, statistical algorithms that were designed for processing data in-memory must now adapt to datasets that span multiple nodes/racks and could not hope to fit in a single block of memory.

Running Statistical Models in MapReduce

Converting statistical models to run in parallel is a challenging task. In the traditional paradigm for parallel programming, memory access is regulated through the use of *threads* — sub-processes created by the operating system to distribute a single shared memory across multiple

processors. Factors such as race conditions between competing threads — when two or more threads try to change shared data at the same time — can influence the performance of your algorithm, as well as affect the precision of the statistical results your program outputs — particularly for long-running analyses of large sample sets.

A pragmatic approach to this problem is to assume that not many statisticians will know the ins and outs of MapReduce (and vice-versa), nor can we expect they'll be aware of all the pitfalls that parallel programming entails. Contributors to the Hadoop project have (and continue to develop) statistical tools with these realities in mind. The upshot: Hadoop offers many solutions for implementing the algorithms required to perform statistical modeling and analysis, without overburdening the statistician with nuanced parallel programming considerations. We'll be looking at the following tools in greater detail:

- Mahout — and its wealth of statistical models and library functions
- The R language — and how to run it over Hadoop (including Big R)

Machine Learning with Mahout

Machine learning refers to a branch of artificial intelligence techniques that provides tools enabling computers to improve their analysis based on previous events. These computer systems leverage historical data from previous attempts at solving a task in order to improve the performance of future attempts at similar tasks. In terms of expected outcomes, machine learning may sound a lot like that other buzzword "data mining"; however, the former focuses on prediction through analysis of *prepared* training data, the latter is concerned with knowledge discovery from *unprocessed* raw data. For this reason, machine learning depends heavily upon statistical modelling techniques and draws from areas of probability theory and pattern recognition.

Mahout is an open source project from Apache, offering Java libraries for distributed or otherwise scalable machine-learning algorithms. (See Figure 9-1 for the Big Picture.) These algorithms cover classic machine learning tasks such as classification, clustering, association rule analysis, and recommendations. Although Mahout libraries are designed to work within an Apache Hadoop context, they are also compatible with any system supporting the MapReduce framework. For example, Mahout provides Java libraries for Java collections and common math operations (linear algebra and statistics) that can be used without Hadoop.

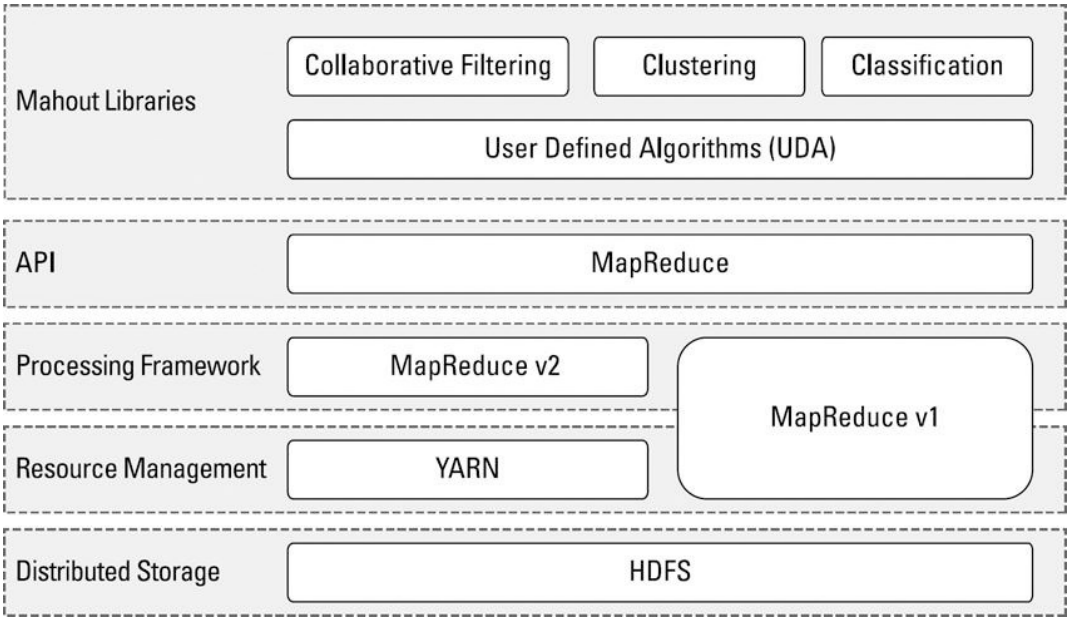

Figure 9-1: High-level view of a Mahout deployment over the Hadoop framework

As you can see in Figure 9-1, the Mahout libraries are implemented in Java MapReduce and run on your cluster as collections of MapReduce jobs on either YARN (with MapReduce v2), or MapReduce v1.

REMEMBER Mahout is an evolving project with multiple contributors. By the time of this writing, the collection of algorithms available in the Mahout libraries is by no means complete; however, the collection of algorithms implemented for use continues to expand with time.

There are three main categories of Mahout algorithms for supporting statistical analysis: collaborative filtering, clustering, and classification. The next few sections tackle each of these categories in turn.

Collaborative Filtering

Mahout was specifically designed for serving as a recommendation engine, employing what is known as a *collaborative filtering* algorithm. Mahout combines the wealth of clustering and classification algorithms at its disposal to produce more precise recommendations based on input data. These recommendations are often applied against user preferences, taking into consideration the behavior of the user. By comparing a user's previous selections, it is possible to identify the nearest neighbors (persons with a similar decision history) to that user and predict future selections based on the behavior of the neighbors.

Consider a "taste profile" engine such as Netflix — an engine which recommends ratings based on that user's previous scoring and viewing habits. In this example, the behavioral patterns for a user are compared against the user's history — and the trends of users with similar tastes belonging to the same Netflix community — to generate a recommendation for content not yet viewed by the user in question.

Clustering

Unlike the supervised learning method described earlier for Mahout's recommendation engine feature, clustering is a form of *unsupervised* learning — where the labels for data points are unknown ahead of time and must be inferred from the data without human input (the *supervised* part). Generally, objects within a cluster should be similar; objects from different clusters should be dissimilar. Decisions made ahead of time about the number of clusters to generate, the criteria for measuring "similarity," and the representation of objects will impact the labelling produced by clustering algorithms.

For example, a clustering engine that is provided a list of news articles should be able to define clusters of articles within that collection which discuss similar topics. Suppose a set of articles about Canada, France, China, forestry, oil, and wine were to be clustered. If the maximum number of clusters were set to 2, our algorithm might produce categories such as "regions" and "industries." Adjustments to the number of clusters will produce different categorizations; for example, selecting for 3 clusters may result in pairwise groupings of nation-industry categories.

Classifications

Classification algorithms make use of human-labelled training data sets, where the categorization and classification of all future input is governed by these known labels. These classifiers implement what is known as *supervised learning* in the machine learning world. Classification rules — set by the training data, which has been labelled ahead of time by domain experts — are then applied against raw, unprocessed data to best determine their appropriate labelling.

These techniques are often used by e-mail services which attempt to classify spam e-mail before they ever cross your inbox. Specifically, given an e-mail containing a set of phrases known to commonly occur together in a certain class of spam mail — delivered from an address belonging to a known botnet — our classification algorithm is able to reliably identify the e-mail as malicious.

REMEMBER In addition to the wealth of statistical algorithms that Mahout provides natively, a supporting *User Defined Algorithms* (UDA) module is also available. Users can override existing algorithms or implement their own through the UDA module. This robust customization allows for performance tuning of native Mahout algorithms and flexibility in tackling unique statistical analysis challenges. If Mahout can be viewed as a statistical analytics extension to Hadoop, UDA should be seen as an extension to Mahout's statistical capabilities.

Traditional statistical analysis applications (such as SAS, SPSS, and R) come with powerful tools for generating workflows. These applications utilize intuitive graphical user interfaces that allow for better data visualization. Mahout scripts follow a similar pattern as these other tools for generating statistical analysis workflows. (See Figure 9-2.) During the final data exploration and visualization step, users can export to human-readable formats (JSON, CSV) or take advantage of visualization tools such as Tableau Desktop.

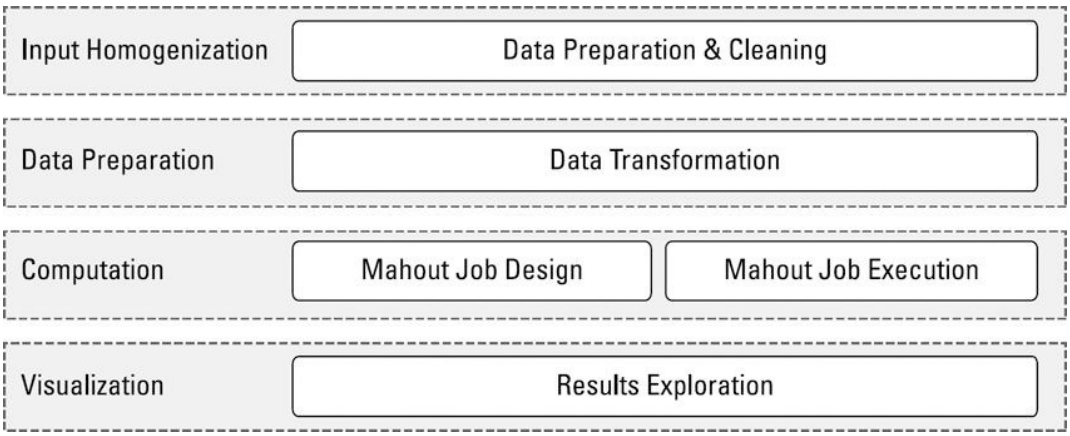

Figure 9-2: Generalized statistical analysis workflow for Mahout

Recall from Figure 9-1 that Mahout's architecture sits atop the Hadoop platform. Hadoop unburdens the programmer by separating the task of programming MapReduce jobs from the complex bookkeeping needed to manage parallelism across distributed file systems. In the same spirit, Mahout provides programmer-friendly abstractions of complex statistical algorithms, ready for implementation with the Hadoop framework.

R on Hadoop

The machine learning discipline has a rich and extensive catalogue of techniques. Mahout brings a range of statistical tools and algorithms to the table, but it only captures a fraction of those techniques and algorithms, as the task of converting these models to a MapReduce framework is a challenging one. Over time, Mahout is sure to continue expanding its statistical toolbox, but until then we advise all data scientists and statisticians out there to be aware of alternative statistical modelling software — which is where R comes in.

The R Language

The R language is a powerful and popular open-source statistical language and development environment. It offers a rich analytics ecosystem that can assist data scientists with data exploration, visualization, statistical analysis and computing, modelling, machine learning, and simulation. The R language is commonly used by statisticians, data miners, data analysts, and (nowadays) data scientists.

R language programmers have access to the *Comprehensive R Archive Network* (CRAN) libraries which, as of the time of this writing, contains over 3000 statistical analysis packages. These add-ons can be pulled into any R project, providing rich analytical tools for running classification, regression, clustering, linear modelling, and more specialized machine learning algorithms. The language is accessible to those familiar with simple data structure types — vectors, scalars, data frames (matrices), and the like — commonly used by statisticians as well as programmers.

Out of the box, one of the major pitfalls with using the R language is the lack of support it offers for running concurrent tasks. Statistical language tools like R excel at rigorous analysis, but lack scalability and native support for parallel computations. These systems are non-distributable and were not developed to be scalable for the modern petabyte-world of big data. Proposals for overcoming these limitations need to extend R's scope beyond in-memory loading and single computer execution environments, while maintaining R's flair for easily-deployable statistical algorithms.

Hadoop Integration with R

In the beginning, big data and R were not natural friends. R programming requires that all objects be loaded into the main memory of a single machine. The limitations of this architecture are quickly realized when big data becomes a part of the equation. In contrast, distributed file systems such as Hadoop are missing strong statistical techniques but are ideal for scaling complex operations and tasks. Vertical scaling solutions — requiring investment in costly supercomputing hardware — often cannot compete with the cost-value return offered by distributed, commodity hardware clusters.

To conform to the in-memory, single-machine limitations of the R language, data scientists often had to restrict analysis to only a subset of the available sample data. Prior to deeper integration with Hadoop, R language programmers offered a scale-out strategy for overcoming the in-memory challenges posed by large data sets on single machines. This was achieved using message-passing systems and paging. This technique is able to facilitate work over data sets too large to store in main memory simultaneously; however, its low-level programming approach presents a steep learning curve for those unfamiliar with parallel programming paradigms.

Alternative approaches seek to integrate R's statistical capabilities with Hadoop's distributed clusters in two ways: interfacing with SQL query languages, and integration with Hadoop Streaming. With the former, the goal is to leverage existing SQL data warehousing platforms such as Hive (see Chapter 13) and Pig (see Chapter 8). These schemas simplify Hadoop job programming using SQL-style statements in order to provide high-level programming for conducting statistical jobs over Hadoop data. For programmers wishing to program MapReduce jobs in languages (including R) other than Java, a second option is to make use of Hadoop's Streaming API. User-submitted MapReduce jobs undergo data transformations with the assistance of UNIX standard streams and serialization, guaranteeing Java-compliant input to Hadoop — regardless of the language originally inputted by the programmer.

Developers continue to explore various strategies to leverage the distributed computation capability of MapReduce and the almost limitless storage capacity of HDFS in ways that can be exploited by R. Integration of Hadoop with R is ongoing, with offerings available from IBM (Big R as part of BigInsights) and Revolution Analytics (Revolution R Enterprise). Bridging solutions that integrate high-level programming and querying languages with Hadoop, such as RHive and RHadoop, are also available. Fundamentally, each system aims to deliver the deep analytical capabilities of the R language to much larger sets of data. In closing this chapter, we briefly examine some of these efforts to marry Hadoop's scalability with R's statistical capabilities.

RHive

The RHive framework serves as a bridge between the R language and Hive. RHive delivers the rich statistical libraries and algorithms of R to data stored in Hadoop by extending Hive's SQL-like query language (HiveQL) with R-specific functions. Through the RHive functions, you can use HiveQL to apply R statistical models to data in your Hadoop cluster that you have cataloged using Hive.

RHadoop

Another open source framework available to R programmers is RHadoop, a collection of packages intended to help manage the distribution and analysis of data with Hadoop. Three packages of note — rmr2, rhdfs, and rhbase — provide most of RHadoop's functionality:

- **rmr2**: The rmr2 package supports translation of the R language into Hadoop-compliant MapReduce jobs (producing efficient, low-level MapReduce code from higher-level R code).
- **rhdfs**: The rhdfs package provides an R language API for file management over HDFS stores. Using rhdfs, users can read from HDFS stores to an R data frame (matrix), and similarly write data from these R matrices back into HDFS storage.
- **rhbase**: rhbase packages provide an R language API as well, but their goal in life is to deal with database management for HBase stores, rather than HDFS files.

Revolution R

Revolution R (by Revolution Analytics) is a commercial R offering with support for R integration on Hadoop distributed systems. Revolution R promises to deliver improved performance, functionality, and usability for R on Hadoop. To provide deep analytics akin to R, Revolution R

makes use of the company's ScaleR library — a collection of statistical analysis algorithms developed specifically for enterprise-scale big data collections.

ScaleR aims to deliver fast execution of R program code on Hadoop clusters, allowing the R developer to focus exclusively on their statistical algorithms and not on MapReduce. Furthermore, it handles numerous analytics tasks, such as data preparation, visualization, and statistical tests.

IBM BigInsights Big R

Big R offers end-to-end integration between R and IBM's Hadoop offering, BigInsights, enabling R developers to analyze Hadoop data. The aim is to exploit R's programming syntax and coding paradigms, while ensuring that the data operated upon stays in HDFS. R datatypes serve as proxies to these data stores, which means R developers don't need to think about low-level MapReduce constructs or any Hadoop-specific scripting languages (like Pig).

BigInsights Big R technology supports multiple data sources — including flat files, HBase, and Hive storage formats — while providing parallel and partitioned execution of R code across the Hadoop cluster. It hides many of the complexities in the underlying HDFS and MapReduce frameworks, allowing Big R functions to perform comprehensive data analytics — on both structured and unstructured data. Finally, the scalability of Big R's statistical engine allows R developers to make use of both pre-defined statistical techniques, as well as author new algorithms themselves.