Spark, an alternative for fast data analytics

M. Tim Jones
Consultant Engineer
Consultant

01 November 2011

Although Hadoop captures the most attention for distributed data analytics, there are alternatives that provide some interesting advantages to the typical Hadoop platform. Spark is a scalable data analytics platform that incorporates primitives for in-memory computing and therefore exercises some performance advantages over Hadoop's cluster storage approach. Spark is implemented in and exploits the Scala language, which provides a unique environment for data processing. Get to know the Spark approach for cluster computing and its differences from Hadoop.
16 Feb 2012 - *Added link to a Spark practice article to the introduction, Going further, and Resources sections*

Spark is an open source cluster computing environment similar to Hadoop, but it has some useful differences that make it superior in certain workloads—namely, Spark enables in-memory distributed datasets that optimize iterative workloads in addition to interactive queries.

Spark is implemented in the Scala language and uses Scala as its application framework. Unlike Hadoop, Spark and Scala create a tight integration, where Scala can easily manipulate distributed datasets as locally collective objects.

Experiment with Spark

In this practice session, Data analysis and performance with Spark, explore multi-thread and multi-node performance with Spark and Mesos and its tunable parameters.(M. Tim Jones, developerWorks, February 2012).

Although Spark was created to support iterative jobs on distributed datasets, it's actually complementary to Hadoop and can run side by side over the Hadoop file system. This behavior is supported through a third-party clustering framework called *Mesos.* Spark was developed at the University of California, Berkeley, Algorithms, Machines, and People Lab to build large-scale and low-latency data analytics applications.

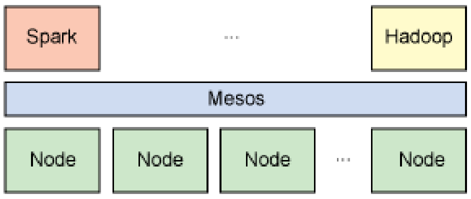
Spark cluster computing architecture

Although Spark has similarities to Hadoop, it represents a new cluster computing framework with useful differences. First, Spark was designed for a specific type of workload in cluster computing —namely, those that reuse a working set of data across parallel operations (such as machine learning algorithms). To optimize for these types of workloads, Spark introduces the concept of in-memory cluster computing, where datasets can be cached in memory to reduce their latency of access.

Spark also introduces an abstraction called *resilient distributed datasets* (RDDs). An RDD is a read-only collection of objects distributed across a set of nodes. These collections are resilient, because they can be rebuilt if a portion of the dataset is lost. The process of rebuilding a portion of the dataset relies on a fault-tolerance mechanism that maintains *lineage* (or information that allows the portion of the dataset to be re-created based on the process from which the data was derived). An RDD is represented as a Scala object and can be created from a file; as a parallelized slice (spread across nodes); as a transformation of another RDD; and finally through changing the persistence of an existing RDD, such as requesting that it be cached in memory.

Applications in Spark are called *drivers,* and these drivers implement the operations performed either on a single node or in parallel across a set of nodes. Like Hadoop, Spark supports a single-node cluster or a multi-node cluster. For multi-node operation, Spark relies on the Mesos cluster manager. Mesos provides an efficient platform for resource sharing and isolation for distributed applications (see Figure 1). This setup allows Spark to coexist with Hadoop in a single shared pool of nodes.

Figure 1. Spark relies on the Mesos cluster manager for resource sharing and isolation.



Spark programming model

A driver can perform two types of operations on a dataset: an action and a transformation. An *action* performs a computation on a dataset and returns a value to the driver; a *transformation* creates a new dataset from an existing dataset. Examples of actions include performing a Reduce operation (using a function) and iterating a dataset (running a function on each element, similar to the Map operation). Examples of transformations include the Map operation and the Cache operation (which requests that the new dataset be stored in memory).

We'll look at examples of these two operations shortly, but first, let's get acquainted with the Scala language.

Brief introduction to Scala

Scala may be one of the Internet's best-kept secrets. You can find Scala in production at some of the Internet's busiest websites, including Twitter, LinkedIn, and Foursquare (with its web application framework, called *Lift*). There's also evidence to suggest that financial institutions have taken an interest in the performance of Scala (such as EDF Trading's use for derivative pricing).

Scala is a multi-paradigm language in that it supports language features associated with imperative, functional, and object-oriented languages in a smooth and comfortable way. From the perspective of object-orientation, every value in Scala is an object. Similarly, from the functional perspective, every function is a value. Scala is also statically typed with a type system both expressive and safe.

In addition, Scala is a virtual machine (VM) language and runs directly on the Java™ Virtual Machine (JVM) using the Java Runtime Environment version 2 through byte codes that the Scala compiler generates. This setup allows Scala to run almost everywhere the JVM runs (with the requirement of an additional Scala run time library). It also allows Scala to exploit the vast catalog of Java libraries that exist, along with your existing Java code.

Finally, Scala is extensible. The language (which actually stands for *Scalable Language*) was defined for simple extensions that integrate cleanly into the language.

Scala's origins

The Scala language began at the Ecole Polytechnique Federale de Lausanne (one of the two Swiss Federal Institutes of Technology in Lausanne, Switzerland). It was designed by Martin Odersky following his work on a programming language called *Funnel* that integrated ideas from functional programming and Petri nets. In 2011, the Scala design team received a 5-year research grant from the European Research Council, and the new company (Typesafe) formed to commercially support Scala received a funding round to begin operations.

Scala illustrated

Let's look at some examples of the Scala language in action. Scala comes with its own interpreter, allowing you to experiment with the language in an interactive way. A useful treatment of Scala is beyond the scope of this article, but you can find links to more information in Resources.

Listing 1 begins our quick tour of the Scala language through its interpreter. After starting Scala, you're greeted with its prompt, through which you can interactively evaluate expressions and programs. Begin by creating two variables—one being immutable (`vals`, called *single assignment*) and one being mutable (`vars`). Note that when you try to change `b` (your `var`), you succeed, but an error is returned when you attempt the change to your `val`.

Listing 1. Simple variables in Scala

```
$ scala
Welcome to Scala version 2.8.1.final (OpenJDK Client VM, Java 1.6.0_20).
Type in expressions to have them evaluated.
Type :help for more information.

scala> val a = 1
a: Int = 1

scala> var b = 2
b: Int = 2

scala> b = b + a
b: Int = 3

scala> a = 2
<console>6: error: reassignment to val
       a = 2
         ^
```

Next, create a simple method that calculates and returns the square of an `Int`. Defining a method in Scala begins with `def`, followed by the method name and a list of parameters, then you set it to number of statements (in this example, one). No return value is specified, as it can be inferred from the method itself. Note how this is similar to assigning a value to a variable. I demonstrate this process on an object called `3` and a result variable called `res0` (which the Scala interpreter creates for you automatically). This is all shown in Listing 2.

Listing 2. A simple method in Scala

```
scala> def square(x: Int) = x*x
square: (x: Int)Int

scala> square(3)
res0: Int = 9

scala> square(res0)
res1: Int = 81
```

Next, let's look at the construction of a simple class in Scala (see Listing 3). You define a simple `Dog` class that accepts a `String` argument (your name constructor). Note here that the class takes the parameter directly (with no definition of the class parameter in the body of the class). A single method exists that emits a string when called. You create a new instance of your class, and then invoke your method. Note that the interpreter inserts the vertical bars: They are not part of the code.

Listing 3. A simple class in Scala

```
scala> class Dog( name: String ) {
     |   def bark() = println(name + " barked")
     | }
defined class Dog

scala> val stubby = new Dog("Stubby")
stubby: Dog = Dog@1dd5a3d

scala> stubby.bark
Stubby barked

scala>
```

When you're done, simply type `:quit` to exit the Scala interpreter.

Installing Scala and Spark

The first step is to download and configure Scala. The commands shown in Listing 4 illustrate downloading and preparing the Scala installation. Use the 2.8 version of Scala, because this is what Spark is documented as needing.

Listing 4. Installing Scala

```
$ wget http://www.scala-lang.org/downloads/distrib/files/scala-2.8.1.final.tgz
$ sudo tar xvfz scala-2.8.1.final.tgz --directory /opt/
```

To make Scala visible, add the following lines to your .bashrc (if you're using Bash as your shell):

```
export SCALA_HOME=/opt/scala-2.8.1.final
export PATH=$SCALA_HOME/bin:$PATH
```

You can then test your installation, as illustrated in Listing 5. This set of commands loads the changes to the bashrc file, and then does a quick test of the Scala interpreter shell.

Listing 5. Configuring and running interactive Scala

```
$ scala
Welcome to Scala version 2.8.1.final (OpenJDK Client VM, Java 1.6.0_20).
Type in expressions to have them evaluated.
Type :help for more information.

scala> println("Scala is installed!")
Scala is installed!

scala> :quit
$
```

As shown, you should now see a Scala prompt. You can exit by typing `:quit`. Note that Scala executes within the context of the JVM, so you'll need that also. I'm using Ubuntu, which comes with OpenJDK by default.

Next, get the latest copy of the Spark framework. To do so, use the script shown in Listing 6.

Listing 6. Downloading and installing the Spark framework

```
wget https://github.com/mesos/spark/tarball/0.3-scala-2.8/
mesos-spark-0.3-scala-2.8-0-gc86af80.tar.gz
$ sudo tar xvfz mesos-spark-0.3-scala-2.8-0-gc86af80.tar.gz
```

Next, set up the spark configuration in ./conf/spar-env.sh with the following line for the Scala home directory:

```
export SCALA_HOME=/opt/scala-2.8.1.final
```

The final step in setup is to update your distribution using the simple build tool (`sbt`). `sbt` is a build tool for Scala and has been used with the Spark distribution. You perform the update and compile step in the mesos-spark-c86af80 subdirectory as:

```
$ sbt/sbt update compile
```

Note that you'll need to be connected to the Internet when you perform this step. When complete, run a quick test of Spark, as shown in Listing 7. In this test, you request to run the SparkPi example, which calculates an estimation of pi (through random point sampling in the unit square). The format shown requests the sample program (spark.examples.SparkPi) and the host parameter, which defines the Mesos master (in this case, your localhost, because it's a single-node cluster) and the number of threads to use. Notice that in Listing 7, two tasks are executed, but they are serialized (task 0 starts and finishes before task 1 begins).

Listing 7. Performing a quick test of Spark

```
$ ./run spark.examples.SparkPi local[1]
11/08/26 19:52:33 INFO spark.CacheTrackerActor: Registered actor on port 50501
11/08/26 19:52:33 INFO spark.MapOutputTrackerActor: Registered actor on port 50501
11/08/26 19:52:33 INFO spark.SparkContext: Starting job...
11/08/26 19:52:33 INFO spark.CacheTracker: Registering RDD ID 0 with cache
11/08/26 19:52:33 INFO spark.CacheTrackerActor: Registering RDD 0 with 2 partitions
11/08/26 19:52:33 INFO spark.CacheTrackerActor: Asked for current cache locations
11/08/26 19:52:33 INFO spark.LocalScheduler: Final stage: Stage 0
11/08/26 19:52:33 INFO spark.LocalScheduler: Parents of final stage: List()
11/08/26 19:52:33 INFO spark.LocalScheduler: Missing parents: List()
11/08/26 19:52:33 INFO spark.LocalScheduler: Submitting Stage 0, which has no missing ...
11/08/26 19:52:33 INFO spark.LocalScheduler: Running task 0
11/08/26 19:52:33 INFO spark.LocalScheduler: Size of task 0 is 1385 bytes
11/08/26 19:52:33 INFO spark.LocalScheduler: Finished task 0
11/08/26 19:52:33 INFO spark.LocalScheduler: Running task 1
11/08/26 19:52:33 INFO spark.LocalScheduler: Completed ResultTask(0, 0)
11/08/26 19:52:33 INFO spark.LocalScheduler: Size of task 1 is 1385 bytes
11/08/26 19:52:33 INFO spark.LocalScheduler: Finished task 1
11/08/26 19:52:33 INFO spark.LocalScheduler: Completed ResultTask(0, 1)
11/08/26 19:52:33 INFO spark.SparkContext: Job finished in 0.145892763 s
Pi is roughly 3.14952
$
```

By increasing the number of threads, you can not only increase the parallelization of thread execution but also execute the job in less time (as shown in Listing 8).

Listing 8. Another quick test of Spark with two threads

```
$ ./run spark.examples.SparkPi local[2]
11/08/26 20:04:30 INFO spark.MapOutputTrackerActor: Registered actor on port 50501
```

```
11/08/26 20:04:30 INFO spark.CacheTrackerActor: Registered actor on port 50501
11/08/26 20:04:30 INFO spark.SparkContext: Starting job...
11/08/26 20:04:30 INFO spark.CacheTracker: Registering RDD ID 0 with cache
11/08/26 20:04:30 INFO spark.CacheTrackerActor: Registering RDD 0 with 2 partitions
11/08/26 20:04:30 INFO spark.CacheTrackerActor: Asked for current cache locations
11/08/26 20:04:30 INFO spark.LocalScheduler: Final stage: Stage 0
11/08/26 20:04:30 INFO spark.LocalScheduler: Parents of final stage: List()
11/08/26 20:04:30 INFO spark.LocalScheduler: Missing parents: List()
11/08/26 20:04:30 INFO spark.LocalScheduler: Submitting Stage 0, which has no missing ...
11/08/26 20:04:30 INFO spark.LocalScheduler: Running task 0
11/08/26 20:04:30 INFO spark.LocalScheduler: Running task 1
11/08/26 20:04:30 INFO spark.LocalScheduler: Size of task 1 is 1385 bytes
11/08/26 20:04:30 INFO spark.LocalScheduler: Size of task 0 is 1385 bytes
11/08/26 20:04:30 INFO spark.LocalScheduler: Finished task 0
11/08/26 20:04:30 INFO spark.LocalScheduler: Finished task 1
11/08/26 20:04:30 INFO spark.LocalScheduler: Completed ResultTask(0, 1)
11/08/26 20:04:30 INFO spark.LocalScheduler: Completed ResultTask(0, 0)
11/08/26 20:04:30 INFO spark.SparkContext: Job finished in 0.101287331 s
Pi is roughly 3.14052
$
```

Building a simple Spark application with Scala

To build a Spark application, you need Spark and its dependencies in a single Java archive (JAR) file. Create this JAR in Spark's top-level directory with `sbt` as:

```
$ sbt/sbt assembly
```

The result is the file ./core/target/scala_2.8.1/"Spark Core-assembly-0.3.jar"). Add this file to your CLASSPATH so that it's accessible. In this example, you won't use this JAR, because you'll run it with the Scala interpreter instead of compiling it.

For this example, use the standard MapReduce transformation (shown in Listing 9). The example begins with the necessary imports for the Spark classes. Next, you define your class (`SparkTest`), with its main method, which parses the arguments for later use. These arguments define the environment from which Spark will be executed (in this case, a single-node cluster). Next, create your `SparkContext` object, which tells Spark how to access your cluster. This object requires two parameters: the Mesos master name (passed in) and the name that you assign the job (`SparkTest`). Parse the number of slices from the command line, which tells Spark how many threads to use for the job. The last remaining item for setup is specifying the text file to use for the MapReduce operation.

Finally, you get to the real meat of the Spark example, which consists of a set of transformations. With your file, invoke the `flatMap` method to return an RDD (through the specified function to split up the text line into tokens). This RDD is then passed through the `map` method (which creates the key-value pairs) and finally through the `ReduceByKey` method, which aggregates your key-value pairs. It does this by passing the key-value pairs to the `_ + _` anonymous function. This function simply takes two parameters (the key and value) and returns the result by appending them together (a `String` and an `Int`). This value is then emitted as a text file (to the output directory).

Listing 9. MapReduce in Scala/Spark (SparkTest.scala)

```
import spark.SparkContext
import SparkContext._
```

```
object SparkTest {

  def main( args: Array[String]) {

    if (args.length == 0) {
      System.err.println("Usage: SparkTest <host> [<slices>]")
      System.exit(1)
    }

    val spark = new SparkContext(args(0), "SparkTest")
    val slices = if (args.length > 1) args(1).toInt else 2

    val myFile = spark.textFile("test.txt")
    val counts = myFile.flatMap(line => line.split(" "))
                       .map(word => (word, 1))
                       .reduceByKey(_ + _)

    counts.saveAsTextFile("out.txt")

  }

}

SparkTest.main(args)
```

To execute your script, simply request execution with:

```
$ scala SparkTest.scala local[1]
```

You can find the MapReduce test file in the output directory (as output/part-00000).

Other big data analytics frameworks

Since Hadoop was developed, a number of other big data analytics platforms have arrived that may be worthy of a look. These platforms range from simple script-based offerings to production environments similar to Hadoop.

One of the simplest is called `bashreduce`, which as the name suggests allows you to perform MapReduce-type operations across multiple machines in the Bash environment. `bashreduce` relies on Secure Shell (password-less) for the cluster of machines you plan to use, and then exists as a script through which you request jobs via UNIX®-style tools (`sort`, `awk`, `netcat`, and the like).

GraphLab is another interesting implementation of the MapReduce abstraction that focuses on parallel implementation of machine learning algorithms. In GraphLab, the Map stage defines computations that can be performed independently in isolation (on separate hosts), and the Reduce stage combines the results.

Finally, a newcomer to the big data scene is Storm from Twitter (through the acquisition of BackType). Storm is defined as the "Hadoop of real-time processing" and is focused on stream processing and continuous computation (stream results out as they're computed). Storm is written in Clojure (a modern dialect of the Lisp language) but supports applications written in any language (such as Ruby and Python). Twitter released Storm as open source in September 2011.

See Resources for more information.

Going further

Experiment with Spark

In this practice session, Data analysis and performance with Spark, explore multi-thread and multi-node performance with Spark and Mesos and its tunable parameters.(M. Tim Jones, developerWorks, February 2012).

Spark is an interesting addition to the growing family of big data analytics solutions. It provides not only an efficient framework for the processing of distributed datasets but does so in an efficient way (through simple and clean Scala scripts). Both Spark and Scala are under active development. However, with their adoption at key Internet properties, it appears that both have transitioned from interesting open source software to foundational web technologies.

Resources

Learn

- In this practice session, Data analysis and performance with Spark, explore multi-thread and multi-node performance with Spark and Mesos and its tunable parameters.(M. Tim Jones, developerWorks, February 2012).
- EDF Trading: Implementing a domain-specific language for derivative pricing with Scala: Scala has found adoption in a variety of industries, including stock trading. Learn about one example by watching this video.
- Application virtualization, past and future (M. Tim Jones, developerWorks, May 2011) presents an introduction to virtual machine languages and their implementations.
- Ceylon: True advance, or just another language? (M. Tim Jones, developerWorks July 2011) explores another interesting (work-in-progress) VM language that relies on the JVM.
- First Steps to Scala is a great introduction to the Scala language (written in part by Martin Odersky, the designer of Scala). This lengthy introduction from 2007 covers many aspects of the language. Another useful example is Code Examples for Programming in Scala, which provides Scala recipes for a large variety of code patterns.
- Distributed computing with Linux and Hadoop (Ken Mann and M. Tim Jones, developerWorks, December 2008) provides an introduction to the architecture of Hadoop, including the basics of the MapReduce paradigm for distributed processing of bulk data.
- Distributed data processing with Hadoop (M. Tim Jones, developerWorks 2010): Find a practical introduction to Hadoop, including how to set up and use a single-node Hadoop cluster, how to set up and use a multi-node cluster, and how to develop map and reduce applications within the Hadoop environment.
- developerWorks on Twitter: Follow us for the latest news. You can also follow this author on Twitter at M. Tim Jones.
- developerWorks Open source zone: Find extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.
- Events of interest: Check out upcoming conferences, trade shows, and webcasts that are of interest to IBM open source developers.
- developerWorks podcasts: Tune into interesting interviews and discussions for software developers
- developerWorks On demand demos: Watch our no-cost demos and learn about IBM and open source technologies and product functions.

Get products and technologies

- Spark introduces an in-memory data analytics solution written and supported by the Scala language.
- The simple build tool is the build solution adopted by the Scala language. It offers a simple method for small projects as well as advanced features for complex builds.
- Lift is the web application framework for Scala, similar to the Rails framework for Ruby. You can find Lift in action at Twitter and Foursquare.

- Mesos Project: Spark doesn't support distribution of workloads natively but instead relies on this cluster manager that provides resource isolation and sharing across a network for distributed applications.
- `bashreduce` (a Bash script-based implementation), GraphLab (focused on machine learning), and Storm (acquired by Twitter from BackType, a real-time distributed stream processing system written in Clojure): Hadoop kicked off a number of big data analytics platforms. Other than Spark, you can implement parallel computing architectures with these three offerings.
- Evaluate IBM products in the way that suits you best: Download a product trial, try a product online, use a product in a cloud environment, or spend a few hours in the SOA Sandbox learning how to implement Service Oriented Architecture efficiently. We have trial versions for several information management products. Since you are interested in data analytics you might want to look at IBM SPSS Text Analytics for Surveys, IBM Cognos Business Intelligence and Cognos Express.

Discuss

- developerWorks community: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis. Help build the Real world open source group in the developerWorks community.

About the author

M. Tim Jones

M. Tim Jones is an embedded firmware architect and the author of *Artificial Intelligence: A Systems Approach, GNU/Linux Application Programming* (now in its second edition), *AI Application Programming* (in its second edition), and *BSD Sockets Programming from a Multilanguage Perspective.* His engineering background ranges from the development of kernels for geosynchronous spacecraft to embedded systems architecture and networking protocols development. Tim is a platform architect with Intel and author in Longmont, Colorado.