Process your data with Apache Pig

Get the info you need from big data sets with Apache Pig

M. Tim Jones
Independent author
Consultant

28 February 2012

Apache Pig is a high-level procedural language for querying large semi-structured data sets using Hadoop and the MapReduce Platform. Pig simplifies the use of Hadoop by allowing SQL-like queries to a distributed dataset. Explore the language behind Pig and discover its use in a simple Hadoop cluster.

Connect with Tim

Tim is one of our most popular and prolific authors. Browse all of Tim's articles on developerWorks. Check out Tim's profile and connect with him, other authors, and fellow developers in the developerWorks community.

It's not surprising with the popularity of Hadoop to find its ecosystem growing. One area in particular that is advancing is programming Hadoop applications. Although programming Map and Reduce applications is not overly complex, doing so does require some experience with software development. Apache Pig changes this by creating a simpler procedural language abstraction over MapReduce to expose a more Structured Query Language (SQL)-like interface for Hadoop applications. So instead of writing a separate MapReduce application, you can write a single script in Pig Latin that is automatically parallelized and distributed across a cluster.

Pig Latin example

Let's start with a simple example of Pig and dissect it. One interesting use of Hadoop is searching a large data set for records that satisfy a given search criterion (otherwise known in Linux® as `grep`). Listing 1 shows the simplicity of this process in Pig. Given the three lines shown, only one is the actual search. The first line simply reads the test data set (the messages log) into a bag that represents a collection of tuples. You filter this data (the only entry in the tuple, represented as `$0`, or field 1) with a regular expression, looking for the character sequence `WARN`. Finally, you store this bag, which now represents all of those tuples from messages that contain `WARN` into a new file called *warnings* in the host file system.

Listing 1. A simple Pig Latin script

```
messages = LOAD 'messages';
warns = FILTER messages BY $0 MATCHES '.*WARN+.*';
STORE warns INTO 'warnings';
```

As you can see, this simple script implements a simple flow but would require considerably more code if implemented directly in the traditional MapReduce model. This makes it considerably easier to learn Hadoop and begin working with the data, compared to raw development.

Let's now dig a bit further into the Pig language, and then look at some other examples of the language's capabilities.

The basics of Pig Latin

Pig Latin is a relatively simple language that executes statements. A *statement* is an operation that takes input (such as a bag, which represents a set of tuples) and emits another bag as its output. A *bag* is a relation, similar to table, that you'll find in a relational database (where tuples represent the rows, and individual tuples are made up of fields).

A script in Pig Latin often follows a specific format in which data is read from the file system, a number operations are performed on the data (transforming it in one or more ways), and then the resulting relation is written back to the file system. You can see this pattern in its simplest form (with one transformation) in Listing 1.

Pig has a rich set of data types, supporting not only high-level concepts like bags, tuples, and maps, but also simple data types such as `int`s, `long`s, `float`s, `double`s, `chararray`s, and `bytearray`s. With the simple types, you'll find a range of arithmetic operators (such as `add`, `subtract`, `multiply`, `divide`, and `module`) in addition to a conditional operator called `bincond` that operates similar to the `C ternary` operator. And as you'd expect, a full suite of comparison operators, including rich pattern matching using regular expressions.

All Pig Latin statements operate on relations (and are called *relational operators*). As you saw in Listing 1, there's an operator for loading data from and storing data in the file system. There's a means to `FILTER` data by iterating the rows of a relation. This functionality is commonly used to remove data from the relation that is not needed for subsequent operations. Alternatively, if you need to iterate the columns of a relation instead of the rows, you can use the `FOREACH` operator. `FOREACH` permits nested operations such as `FILTER` and `ORDER` to transform the data during the iteration.

The `ORDER` operator provides the ability to sort a relation based on one or more fields. The `JOIN` operator performs an inner or outer join of two or more relations based on common fields. The `SPLIT` operator provides the ability to split a relation into two or more relations based on a user-defined expression. Finally, the `GROUP` operator groups the data in one or more relations based on some expression. Table 1 provides a partial list of relational operators in Pig.

Table 1. Incomplete list of Pig Latin relational operators

Operator	Description

`FILTER`	Select a set of tuples from a relation based on a condition.
`FOREACH`	Iterate the tuples of a relation, generating a data transformation.
`GROUP`	Group the data in one or more relations.
`JOIN`	Join two or more relations (inner or outer join).
`LOAD`	Load data from the file system.
`ORDER`	Sort a relation based on one or more fields.
`SPLIT`	Partition a relation into two or more relations.
`STORE`	Store data in the file system.

Although this isn't an exhaustive list of operators within Pig Latin, this table provides an extremely useful set of operations to process large data sets. You can learn about the complete Pig Latin language through Resources, as Pig has a nice set of online documentation. Now try your hand at writing some Pig Latin scripts to understand how these operators work in practice.

Getting access to Pig

In earlier articles on Hadoop, I took the approach of installing and configuring Hadoop as a package. But Cloudera has made it even easier to use Hadoop by packaging it with Linux as a virtual appliance. And although it's a large download, the virtual machine (VM) is pre-built and configured with not just Hadoop but also Apache Hive and Pig. So, with one download and a freely available type-2 hypervisor (VirtualBox or Kernel-based Virtual Machine [KVM]), you have an entire Hadoop environment that's preconfigured and ready to go.

Getting Hadoop and Pig up and running

After downloading your specific VM file, you'll need to create a VM for your particular hypervisor. In Resources, you'll find step-by-step instructions for how to do this.

Cloudera VM memory

I found that the VM would not work properly with 1GB of memory assigned to it. Doubling or even tripling that memory allotment results in normal operation (that is, no Java™ heap space issues).

Once you create your VM, you can start it through VirtualBox, which boots the Linux kernel and starts all of the necessary Hadoop daemons. With boot complete, begin by creating a terminal in which to communicate with Hadoop and Pig.

You can use Pig in either of two modes. The first is Local mode, which has no reliance on Hadoop or the Hadoop Distributed File System (HDFS), where everything is executed in a single Java virtual machine (JVM) in the context of the local file system. The other is Mapreduce mode, which uses a Hadoop file system and cluster.

Pig in Local mode

For Local mode, simply start Pig and specify Local mode with the `exectype` option. Doing so brings you to the Grunt shell, which allows you to interactively enter Pig statements:

```
$ pig -x local
...
grunt>
```

From here, you can interactively code your Pig Latin script, seeing the result after each operator. Return to Listing 1 and try this script out (see Listing 2). Note in this case that instead of storing your data to a file, you simply dump it as a set of relations. You'll note in the modified output that each log line (that matches the search criteria defined by the `FILTER`) is itself a relation (bounded by parentheses [`()`]).

Listing 2. Using Pig interactively in Local mode

```
grunt> messages = LOAD '/var/log/messages';
grunt> warns = FILTER messages BY $0 MATCHES '.*WARN+.*';
grunt> DUMP warns
...
(Dec 10 03:56:43 localhost NetworkManager: <WARN> nm_generic_enable_loopback(): error ...
(Dec 10 06:10:18 localhost NetworkManager: <WARN> check_one_route(): (eth0) error ...
grunt>
```

If you had specified the `STORE` operator, it would have generated your data within a directory of the name specified (not a simple regular file).

Pig in Mapreduce mode

For Mapreduce mode, you must first ensure that Hadoop is running. The easiest way to do that is to perform a file list operation on the root of the Hadoop file system tree, as in Listing 3.

Listing 3. Testing Hadoop availability

```
$ hadoop dfs -ls /
Found 3 items
drwxrwxrwx   - hue    supergroup          0 2011-12-08 05:20 /tmp
drwxr-xr-x   - hue    supergroup          0 2011-12-08 05:20 /user
drwxr-xr-x   - mapred supergroup          0 2011-12-08 05:20 /var
$
```

As shown, this code will result in a listing of one or more files, if Hadoop is running successfully. Now, let's test Pig. Begin by starting Pig, and then changing the directory to your hdfs root to determine whether you can see what you saw externally in HDFS (see Listing 4).

Listing 4. Testing Pig

```
$ pig
2011-12-10 06:39:44,276 [main] INFO  org.apache.pig.Main - Logging error messages to...
2011-12-10 06:39:44,601 [main] INFO  org.apache.pig.... Connecting to hadoop file \
system at: hdfs://0.0.0.0:8020
2011-12-10 06:39:44,988 [main] INFO  org.apache.pig.... connecting to map-reduce \
job tracker at: 0.0.0.0:8021
grunt> cd hdfs:///
grunt> ls
hdfs://0.0.0.0/tmp     <dir>
hdfs://0.0.0.0/user    <dir>
hdfs://0.0.0.0/var     <dir>
grunt>
```

So far, so good. You can see your Hadoop file system from within Pig, so now, try to read some data into it from your local host file system. Copy a file from local to HDFS through Pig (see Listing 5).

Listing 5. Getting some test data

```
grunt> mkdir test
grunt> cd test
grunt> copyFromLocal /etc/passwd passwd
grunt> ls
hdfs://0.0.0.0/test/passwd<r 1> 1728
```

Next, with your test data now safely within Hadoop's file system, you can try another script. Note that you can `cat` the file within Pig to see its contents (just to see if it's there). In this particular example, identify the number of shells specified for users within the passwd file (the last column within passwd).

To begin, you need to load your passwd file from HDFS into a Pig relation. You do this before using the `LOAD` operator, but in this case, you want to parse the fields of the password file down to their individual fields. In this example, specify the `PigStorage` function, which allows you to indicate the delimiter character for the file (in this case, a colon [`:`] character). You also specify the individual fields (or the schema) with the `AS` keyword, including their individual types (see Listing 6).

Listing 6. Reading your file into a relation

```
grunt> passwd = LOAD '/etc/passwd' USING PigStorage(':') AS (user:chararray, \
passwd:chararray, uid:int, gid:int, userinfo:chararray, home:chararray, \
shell:chararray);
grunt> DUMP passwd;
(root,x,0,0,root,/root,/bin/bash)
(bin,x,1,1,bin,/bin,/sbin/nologin)
...
(cloudera,x,500,500,,/home/cloudera,/bin/bash)
grunt>
```

Next, use the `GROUP` operator to group the tuples in this relation based on their shell (see Listing 7). Dump this again, just to illustrate the result of the `GROUP` operator. Note here that you have tuples grouped (as an inner bag) under their particular shell being used (with the shell specified at the beginning).

Listing 7. Grouping the tuples as a function of their shell

```
grunt> grp_shell = GROUP passwd BY shell;
grunt> DUMP grp_shell;
(/bin/bash,{(cloudera,x,500,500,,/home/cloudera,/bin/bash),(root,x,0,0,...), ...})
(/bin/sync,{(sync,x,5,0,sync,/sbin,/bin/sync)})
(/sbin/shutdown,{(shutdown,x,6,0,shutdown,/sbin,/sbin/shutdown)})
grunt>
```

But your desire is a count of the unique shells specified within the passwd file. So, you use the `FOREACH` operator to iterate each tuple in your group to `COUNT` the number that appear (see Listing 8).

Listing 8. Grouping the results with counts for each shell

```
grunt> counts = FOREACH grp_shell GENERATE group, COUNT(passwd);
grunt> DUMP counts;
...
(/bin/bash,5)
(/bin/sync,1)
(/bin/false,1)
(/bin/halt,1)
(/bin/nologin,27)
(/bin/shutdown,1)
grunt>
```

Note: To execute this code as a script, simply type your script into a file, and then execute it as `pig myscript.pig`.

Diagnostic operators

Pig supports a number of diagnostic operators that you can use to debug Pig scripts. As you saw in the prior script examples, the `DUMP` operator is invaluable for viewing not only data but the schema of the data itself. You can also use the `DESCRIBE` operator to generate a detailed format of a relation's schema (field and type).

The `EXPLAIN` operator is quite a bit more complex but useful. For a given relation, you can use `EXPLAIN` to view how the physical operators are grouped into map and reduce jobs (that is, how the data was derived).

Table 2 provides a list of diagnostic operators in Pig Latin and their description.

Table 2. Pig Latin diagnostic operators

Operator	Description
`DESCRIBE`	Return the schema of a relation.
`DUMP`	Dump the contents of a relation to the screen.
`EXPLAIN`	Display the MapReduce execution plans.

User-defined functions

Although Pig is powerful and useful in the context explored in this article, it can be made even more powerful through user-defined functions (UDFs). Pig scripts can use functions that you define for things such as parsing input data or formatting output data and even operators. UDFs are written in the Java language and permit Pig to support custom processing. UDFs are the way to extend Pig into your particular application domain. You can learn more about the development of UDFs in Resources.

Pig users

As you can see from this short article, Pig is a powerful tool for querying data in a Hadoop cluster. It's so powerful that Yahoo! estimates that between 40% and 60% of its Hadoop workloads are

generated from Pig Latin scripts. With 100,000 CPUs at Yahoo! and roughly 50% running Hadoop, that's a lot of pigs running around.

But Yahoo! isn't the only organization taking advantage of Pig. You'll find Pig at Twitter (processing logs, mining tweet data); at AOL and MapQuest (for analytics and batch data processing); and at LinkedIn, where Pig is used to discover people you might know. Ebay is reportedly using Pig for search optimization, and adyard uses Pig in about half of its recommender system.

Going forward

Nothing short of a book can enumerate the power behind Pig for processing big data. Pig makes it easy, even for non-developers, to perform big data processing on a Hadoop cluster. Pig was originally developed by Yahoo! in 2006 and shortly thereafter moved into the Apache Software Foundation so that it could be used globally. That move was a result of Yahoo! researchers' recognition of the power that Pig provided to non-developers. As Hadoop grows in popularity as an infrastructure, it's the Hadoop ecosystem that will change the face of big data and its ever-growing use.

Resources

Learn

- The Apache website is the source for information on Pig, including current news, the latest software, how to get started, and how to get involved.
- The Hadoop Demo VM is by far the simplest way to get a Hadoop instance running. The VM contains everything you need, including Hadoop, Hive, and Pig on a CentOS Linux operating system.
- The Cloudera Training VM is a great way to get started with Hadoop. It minimizes the amount of configuration needed and allows you to easily get started processing data sets with Hadoop and Pig.
- Virtual appliances and the Open Virtualization Format (M. Tim Jones, developerWorks, October 2009) explores the use of virtual appliances as a new delivery form for software. Virtual appliances permit the distribution of preconfigured software (with operating system) as a VM.
- Pig has a large number of online resources, ranging from reference manuals, cookbooks, and other resources. Pig also has two well-written manuals (part 1 and part 2), a script cookbook, and a UDF guide.
- Pig has a large user population consisting of prominent web properties. Learn about the variety of web companies that use Pig on Apache's PoweredBy page.
- IBM InfoSphere BigInsights Basic Edition offers a highly scalable and powerful analytics platform that can handle incredibly high data throughput rates that can range to millions of events or messages per second.
- In the developerWorks Linux zone, find hundreds of how-to articles and tutorials, as well as downloads, discussion forums, and a wealth of other resources for Linux developers and administrators.
- The Open Source developerWorks zone provides a wealth of information on open source tools and using open source technologies.
- Stay current with developerWorks technical events and webcasts focused on a variety of IBM products and IT industry topics.
- Attend a free developerWorks Live! briefing to get up-to-speed quickly on IBM products and tools, as well as IT industry trends.
- Watch developerWorks on-demand demos ranging from product installation and setup demos for beginners, to advanced functionality for experienced developers.
- Follow Tim on Twitter. You can also follow developerWorks on Twitter, or subscribe to a feed of Linux tweets on developerWorks.

Get products and technologies

- IBM InfoSphere BigInsights Basic Edition -- IBM's Hadoop distribution -- is an integrated, tested and pre-configured, no-charge download for anyone who wants to experiment with and learn about Hadoop.
- 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.

Discuss

- Get involved in the developerWorks community. Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

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, Colo.