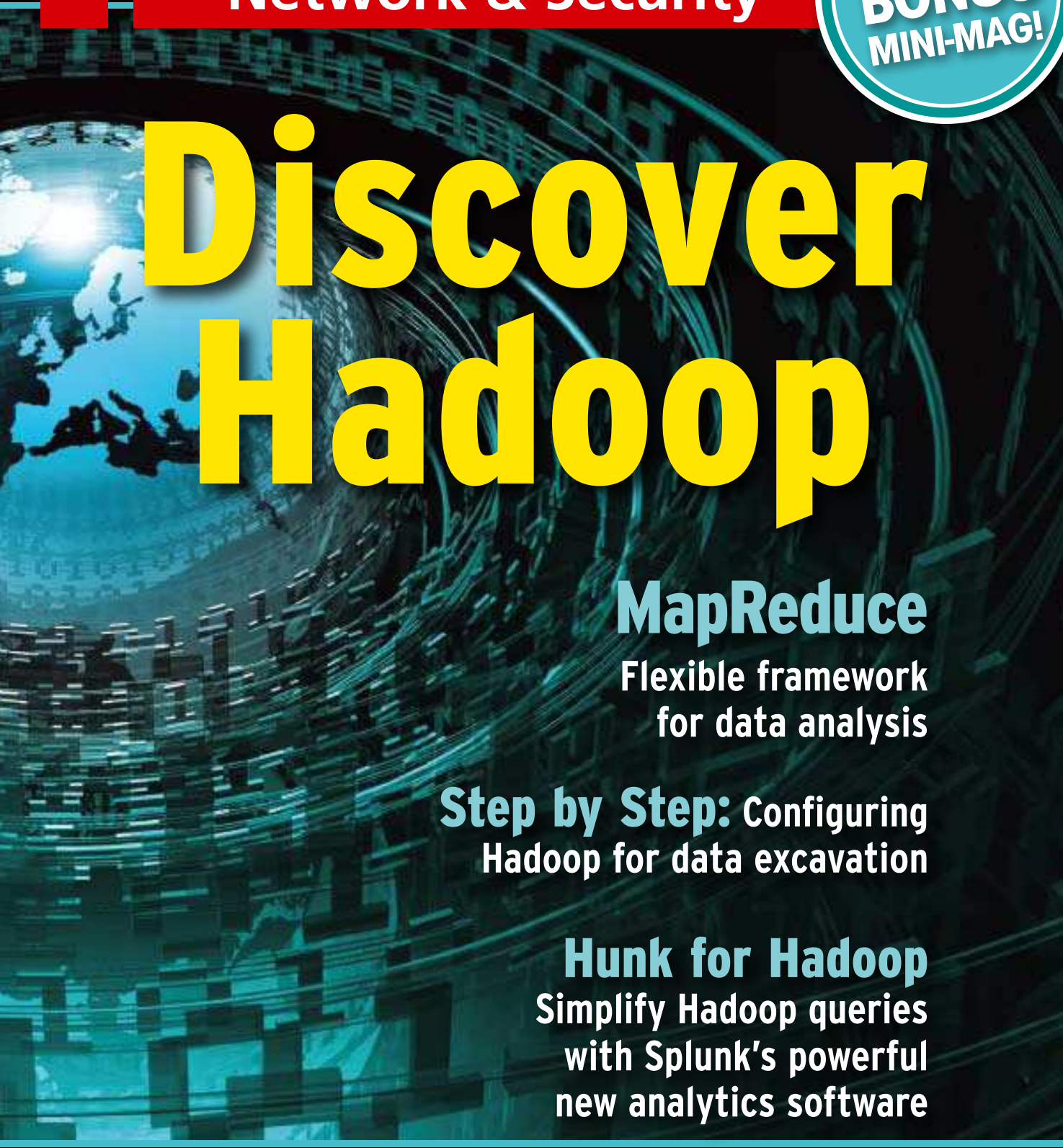
ADMIN

Network & Security

SPECIAL BONUS MINI-MAG!



Discover Hadoop

MapReduce
Flexible framework for data analysis

Step by Step: Configuring Hadoop for data excavation

Hunk for Hadoop
Simplify Hadoop queries with Splunk's powerful new analytics software

US$ 7.95

ADMIN Network & Security

Discover Hadoop

Dear Readers:

Inexpensive data storage, faster networks, and high-performance clusters are changing the way we envision and use computers. One area where the change is most evident is in the new paradigm for unstructured data storage, and the biggest headline for the unstructured data story is Hadoop – a free tool that is bringing the once-inaccessible vision of big data to practical problems on everyday networks.

This ADMIN Magazine special supplement describes how to get started with Hadoop and the big data revolution. We'll explore the features of the Hadoop environment, and we'll show you how to set up and use Hadoop on your own network. We'll also introduce you to Splunk's new Hunk: Splunk Analytics for Hadoop, innovative software that makes it fast and easy to access, analyze, and visualize Hadoop data.

ADMIN Special

Editor in Chief – Joe Casad
Managing Editor – Rita L Sooby
Layout / Design – Dena Friesen, Lori White
Advertising
Ann Jesse, ajesse@admin-magazine.com
Phone: +1-785-841-8834
Publisher – Brian Osborn
Product Management – Christian Ullrich
Customer Service / Subscription
For USA and Canada:
Email: cs@admin-magazine.com
Phone: 1-866-247-2802
(toll-free from the US and Canada)
www.admin-magazine.com

While every care has been taken in the content of the magazine, the publishers cannot be held responsible for the accuracy of the information contained within it or any consequences arising from the use of it.

Printed in Germany

ADMIN ISSN 2045-0702

ADMIN is published by Linux New Media USA, LLC, 616 Kentucky St, Lawrence, KS 66044, USA.

Table of Contents

MapReduce and Hadoop

Giant Data

Enterprises like Google and Facebook use the MapReduce approach to process petabyte-range volumes of data. Apache Hadoop is a powerful open source implementation.

By Thomas Hornung, Martin Przyjaciel-Zablocki, and Alexander Schätzle

Giant volumes of data are nothing unusual in our times of Google and Facebook. In 2010, Facebook sat on top of a mountain of data; just one year later it had grown from 21 to 30 petabytes. If you were to store all of this data on 1TB hard disks and stack them on top of one another, you would have a tower twice as high as the Empire State building in New York.

The height of this tower illustrates that processing and analyzing such data need to take place in a distributed process on multiple machines rather than on a single system. However, this kind of processing has always been very complex, and much time is spent solving recurring problems, like processing in parallel, distributing data to the compute nodes, and, in particular, handling errors during processing. To free developers from these repetitive tasks, Google introduced the MapReduce framework.

The idea behind MapReduce is based on the understanding that most data-intensive programs used by Google are very similar in terms of their basic concept. On the basis of these common features, Google developed an abstraction layer that splits the data flow into two main phases: the map phase and the reduce phase [1]. In a style similar to functional programming, computations can take place in parallel on multiple computers in the map phase. The same thing also applies to the reduce phase, so that MapReduce applications can be massively parallelized on a computer cluster. Automatic parallelization of large-scale computations does not explain the popularity of MapReduce in companies such as Adobe, eBay, Splunk, Twitter, and others. Of course, the Apache Hadoop open source implementation of MapReduce does help, but what is more likely to be important is that Hadoop can be installed on standard hardware and possesses excellent scaling characteristics, which means you can run it on a cluster and then extend the cluster dynamically by purchasing more computers. An equally attractive option is not having to operate your own MapReduce cluster but accessing cloud capacity instead. Amazon, for example, offers Amazon Elastic MapReduce clusters that adapt dynamically to customer requirements.

MapReduce Framework

The pillar of a MapReduce system is a distributed file system whose basic functionality is easily explained: Large files are split into blocks of equal size, which are distributed across the cluster for storage. Because you always need to consider the failure of the computer in a larger cluster, each block is stored multiple times (typically three times) on different computers.

In the implementation of MapReduce, the user applies an alternat-

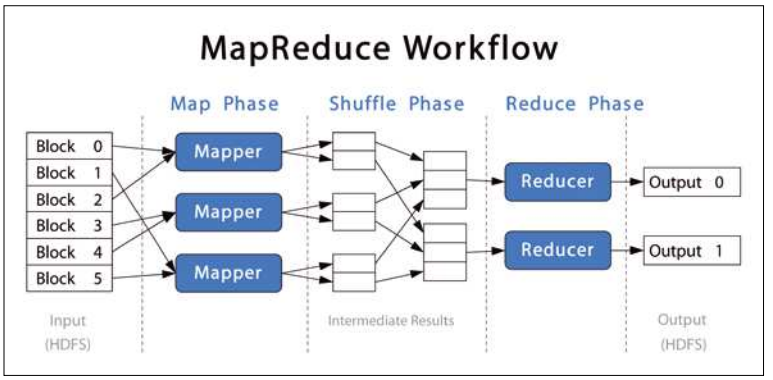


Figure 1: The MapReduce framework breaks down data processing into map, shuffle, and reduce phases. Processing is mainly in parallel on multiple compute nodes.

ing succession of map and reduce functions to the data. Parallel execution of these functions, and the difficulties that occur in the process, are handled automatically by the framework. An iteration comprises three phases: map, shuffle, and reduce (Figure 1).

Map Phase

The map phase applies the `map` function to all input. For this to happen, mappers are launched on all computers in the cluster, whose task it is to process the blocks in the input file that are stored locally on the computer. In other words, computations take place where the data is stored (data locality). Because no dependencies exist between the various mappers, they can work in parallel and independently of one another.

If a computer in the cluster fails, the last or not yet computed map results can be recalculated on another computer that possesses a replica of the corresponding block. A mapper processes the contents of a block line by line, interpreting each line as a key-value pair. The actual `map` function is called individually for each of these pairs and creates an arbitrarily large list of new key-value pairs from it:

```
map(key, value) -> List(key', value')
```

Shuffle Phase

The shuffle phase sorts the resulting pairs from the map phase locally by their keys, after which, MapReduce assigns them to a reducer according to their keys. The framework makes sure all pairs with the same key are assigned to the same reducer. Because the output from the map phase can be distributed arbitrarily across the cluster, the output from the map phase needs to be transferred across the network to the correct producers in the shuffle phase. Because of this, it is normal for large volumes of data to cross the network in this step.

Reduce Phase

The reducer finally collates all the pairs with the same key and creates a sorted list from the values. The key and the sorted list of values provides the input for the `reduce` function.

The `reduce` function typically compresses the list of values to create a shorter list – for example, by aggregating the values. Commonly, it returns a single value as its output. Generally speaking, the `reduce` function creates an arbitrarily large list of key-value pairs, just like the map function:

```
reduce(key, List(values)) -> ➥
  List(key', value')
```

The output from the reduce phase can, if needed, be used as the input for another map-reduce iteration.

Example: Search Engine

A web search engine is a good example for the use of MapReduce. For a system like this, it is particularly important to be able to compute the relevance of the page on the web as accurately as possible. One of many criteria is the number of other pages that link to one page on the web. Viewed in a simple way, this assumption is also the basic idea behind the page rank algorithm that Google uses to evaluate the relevance of a page on the web.

For this to happen, Google continuously searches the web for new information and stores the links between the pages in doing so. If you consider the number of pages and links on the web, it quickly becomes clear why computing the page rank algorithm was one of the first applications for MapReduce at Google.

Figure 2 is a schematic for computing the number of incoming links for a page with the map-reduce method. The input comprises sets of (X, Y) pairs, each of which corresponds to a link from page X to page Y. It is subdivided into two blocks of six entries each – of course, in real life, the input would comprise far more blocks.

The framework assigns a mapper to each block of input, and the

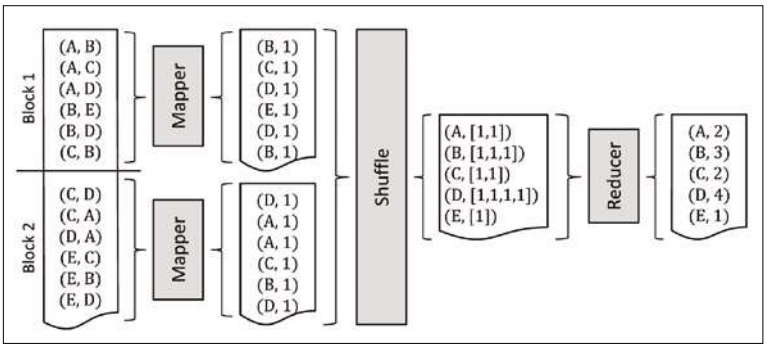


Figure 2: Schematic process of a map-reduce computation.

mapper runs the map function against every entry in the block. To count the number of pages that link to a specific page, it is useful to use the link target (the second value in the input pair) as a key for the output from the map function; all pairs with the same key will be collated downstream. The output value 1 from the map function indicates a link to the corresponding page:

```
method map(source,target)
  emit(target,1)
end
```

The shuffle phase collates all the output with identical keys from the map phase, sorts the output, and distributes it to the reducers. Consequently, for each linked page there is precisely one pair with the corresponding page as the key and a list of 1s as the value. The reducer then applies the `reduce` function to each of these pairs. In this case, it simply needs to add up the number of 1s in the list (pseudocode):

```
method Reduce(target,counts[c1,c2,...])
  sum <- 0
  for all c in counts[c1,c2,...] do
    sum <- sum + c
  end
  emit(target,sum)
end
```

If you take a look at the schematic, you quickly see that a mapper can create multiple key-value pairs for one page. The output from the mapper for block 1, for example, contains the (B, 1) pair twice because there are two links to page B in block 1. Another thing you notice is that the values are not aggregated until they reach the reducer. The framework uses a combiner, which prepares the output from a mapper before sending it across the network to the reducer, thus reducing the volume of data that needs to be transferred.
In the example here, the combiner can simply correlate all of the output from one mapper and compute the total (Figure 3). Combiners are useful, but they can't always be integrated into the process meaningfully; thus, they are thus optional. The user has to decide in each use case whether the combiner will improve the process or not.

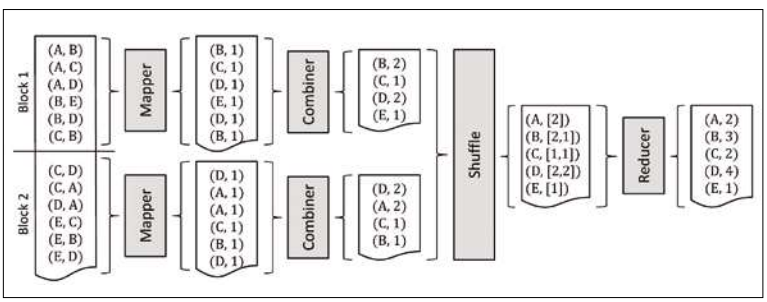


Figure 3: The use of a combiner makes sense for arithmetic operations in particular.

Hadoop

The Apache Hadoop Project is an open source implementation of Google's distributed filesystem (Google File System, GFS [2]) and the MapReduce Framework. One of the most important supporters promoting this project over the years is Yahoo!. Today, many other well-known enterprises, such as Facebook and IBM, as well as an active community, contribute to ongoing development. Through the years, many associative projects have arisen to add functionality by extending the classic framework.

Hadoop Ecosystem

The Hadoop Ecosystem contains numerous extensions that cover a variety of application scenarios.

- Pig is a system developed by Yahoo! to facilitate data analysis in the MapReduce framework. Queries are written in the data transfer motion language Pig Latin, which prefers an incremental and procedural style compared with the declarative approach of SQL. Pig Latin programs can be translated automatically into a series of map-reduce iterations, removing the need for the developer to implement map and reduce functions manually.
- Hive is a data warehouse developed by Facebook. In contrast to Pig Latin, the Hive query

language follows the declarative style of SQL. Hive automatically maps to the MapReduce framework at execution time.
- HBase is a column-oriented NoSQL database based on HDFS. In typical NoSQL database style, HBase is useful for random read/write access in contrast to HDFS.

Distributed Filesystem

The Hadoop Distributed Filesystem (HDFS) follows the pattern laid down by GFS. The architecture follows the classic master-slave principle, wherein one computer in the cluster (NameNode) takes care of management and the other computers (DataNodes) take care restoring the data blocks. The data block is stored on multiple computers, improving both resilience to failure and data locality, taking into account that network bandwidth is a scarce resource in a large cluster. To take some of the load off the network, Hadoop distributes computations in the map phase to the computers so that the greatest amount of data possible can be read locally. HDFS was designed and developed in particular for efficient support of write once/read many access patterns with large files. This also explains why the developers value fast data throughput, although this has a negative effect on latency. Changes to store files are not supported, except for the option of appending to files.

Hadoop MapReduce

The MapReduce implementation in Hadoop also conforms to the master-slave architecture. The JobTracker coordinates the sequence and assigns subtasks to the individual TaskTrackers. A TaskTracker can handle the role of a mapper as well as that of a reducer.

The most important characteristic of MapReduce is the linear scalability of the framework. Simply put, the computational duration of a MapReduce application can be reduced by half (approximately) by doubling the size of the cluster. In practical terms, the genuine scaling benefits will depend on many factors, such as the nature of the problem you are solving. Besides scalability, Hadoop possesses other characteristics that facilitate the development of distributed applications. For example, it automatically captures hardware failures and reruns subtasks that have not completed. It also performs subtasks multiple times toward end of computations to prevent a single outlier from unnecessarily slowing down the entire process (speculative execution). Automatic parallelization of execution by Hadoop doesn't mean that developers no longer need to worry about the process. On the contrary, you need to split the problem you want to solve into a fairly strict sequence of map and reduce phases, which is often very difficult or even impossible. This relatively inflexible schema is also one of the main criticisms leveled at MapReduce.

MapReduce vs. SQL

Managing and analyzing large volumes of data is the classic domain of relational databases, which use SQL as a declarative query language. Does the widespread use of MapReduce now mean that relational databases are superfluous? You can only answer this question on a case-by-case basis. The following aspects provide orientation:
- Data volume: Map reduce is suitable for very large volumes of data that go well beyond the processing capacity of a single computer. To make comparable strides with relational databases, you also need to parallelize query processing. Although possible, this approach will not typically scale in a linear way as you increase the number of computers used.
- Access patterns: One of the ideas behind MapReduce is that processing data sequentially in large blocks optimizes the bleed rate. In contrast, queries that only relate to a part of the data can be answered more efficiently with the help of indices in relational databases. If you wanted to answer a query of this kind with MapReduce, you would need to read the entire data record.
- Data representation: One assumption of relational databases make is that the data possesses an inherent structure (a schema). Users leveraged this structure to achieve what, in effect, is redundancy-free storage data in various tables. What this means for queries, however, is that they often need to combine information from various tables. MapReduce does not support the schema concept but leaves it to the user to convert the data in the map phase to the form required for the reduce phase. The benefit of this is that MapReduce supports more universal usage than a relational database.
- Ad hoc queries: One of the major strengths of relational databases is the declarative query language SQL. In MapReduce, the programmer has to solve each task individually.

This list could go on. One important point for a decision in an enterprise environment will certainly be the question of sustainability and longevity of Hadoop. Relational databases are firmly established, and many companies possess sufficient know-how to cover

their own requirements sufficiently. What you can expect, however, is coexistence of the two approaches, which supplement one another.

Hadoop in Production Use

In addition to the software downloads and tutorials available from the Apache Hadoop website, vendors offering packaged Hadoop distributions provide lots of resources and information on their websites. The list of Hadoop vendors includes Cloudera, Hortonworks, IBM, MapR, and Pivotal.

- Tutorials: For a comprehensive introduction to Hadoop, check out Tom White's book, *Hadoop: The Definitive Guide* [3]. The Cloudera website also includes step-by-step examples with small records.
- Programming language: Hadoop is implemented in Java, which is why the `map` and `reduce` functions are typically also implemented in Java; however, developers also have the option of using other languages such as Python or C++.
- Java API: A new MapReduce API for Java was introduced in Hadoop version 0.20. The classes belonging to the legacy API are available in the `org.apache.hadoop.mapred.*` Java package; the new ones are in `org.apache.hadoop.mapreduce.*`. Many of the examples available still use the old API. Nevertheless newcomers are advised to adopt the new interface at the outset; the MapReduce program cannot use both versions.

Conclusions

From Google's original idea of reducing the complexity of distributed applications, Apache Hadoop has established a rich ecosystem of versatile tools and data processing. In particular, because of its excellent scaling properties, built-in error torrents, and many useful automation features, Apache Hadoop is something that many enterprises and research groups would not want to do without in their daily work.

Of course, this doesn't mean that classic data processing systems like relational databases are no longer needed, but when handling the increasingly large volumes of digital knowledge available in the world, scaling systems like Hadoop will continue to gain importance. ■

Info

[1] Dean, J., S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters." In: OSDI '04: 6th Symposium on Operating Systems Design and Implementation (USENIX and ACM SIGOPS, 2004), pp. 137-150

[2] Ghemawat, S., H. Gobioff, and S. T. Leung: "The Google File System". In: M. L. Scott and L. L. Peterson, eds. Symposium on Operating Systems Principles (SOSP) (ACM, 2003), pp. 29-43

[3] White, T. Hadoop: The Definitive Guide, 3rd ed. O'Reilly, 2012.

The Authors

Hornung, Przyjaciel-Zablocki, and Schätzle are members of the scientific staff at the University of Freiburg computer science department, where they research parallel processing of semantic data.

Big data excavation with Apache Hadoop

Big Dig

Experience the power of supercomputing and the big data revolution with Apache Hadoop. By Kenneth Geisshirt

Apache Hadoop – or simply Hadoop – is a complex beast. It consists of a large number of components, and its architecture is large and complicated. The basic idea of Hadoop is to provide a platform for distributed storage and computation. The philosophy is to use commodity hardware. A Hadoop installation therefore consists of a number of computers linked together by Ethernet. Computers – or nodes – in the cluster have different roles. One node acts as master (in large clusters, you can have more than one master) and other nodes act as slaves.

As you learned in the previous article, Hadoop provides access to a distributed filesystem. The filesystem structure is similar to contemporary operating systems, where the kernel supports a filesystem, and userspace applications access the data through a virtual filesystem. The virtual filesystem hides the inner details of the actual filesystem. In Hadoop, the component that provides access to the filesystem is called the `NameNode`. This component is run by the master node.

Hadoop implements one filesystem by itself called Hadoop Distributed File System (HDFS), which is probably the most popular filesystem choice, but you will find almost a dozen commercial Hadoop offerings, and many of them have their own filesystems.

HDFS chops files into blocks and distributes them among the computers or nodes in the Hadoop cluster. The `NameNode` keeps track of the blocks; that is, when you access a file, the `NameNode` sends requests to the slaves. A component named `DataNode` is run by the slaves, and it will respond to the requests.

The default block size for chopped files is 64MB, but you can set it to any value. The blocks are stored on the filesystem of the nodes. It is important to understand that a distributed filesystem is not a Unix filesystem; for example, you cannot modify a file directly, other than appending to it.

A filesystem for Hadoop must be aware of location. In other words, Hadoop must know if two nodes are placed in different rack cabinets. Blocks are stored multiple times (the default is three times), and if one node vanishes, Hadoop can use one of the copies. Think of this as a form of RAID system across a network. However, Hadoop is more than a filesystem. It also schedules your computations, and you need to write your own parallel program. (In a later section, you find an example of how to do this.) A job is broken down to tasks by the `JobTracker` component running at the master.

The tasks are scheduled to be executed at the slaves. The crucial point is that Hadoop will schedule the tasks in such a way that they will be running at the node with the data the task is operating on. The slaves control and monitor the tasks with a component named `TaskTracker`.

When you deal with commodity hardware – and very large clusters – the probability for a crash is almost 1. When a slave crashes, the current task is rescheduled by `JobTracker`. The task will start running at a slave

that has a copy of the relevant block of data.

Comparing with HPC

Linux clusters dominate HPC and supercomputing. The philosophy of Linux clusters is to use commodity hardware; that is, to use standard (Intel-like) processors, hard drives, and Ethernet – with some modifications, such as specialized network equipment (Infiniband).
In high-performance computing, you find a number of specialized software packages. In particular, you need software for job control and scheduling. SGE and Torque are commonly used for what is often called "the queue system."
In a Linux cluster, data is shared using a network filesystem – often NFS, but CIFS is also possible.
When you can program an HPC solution for a Linux cluster, you are required to find a framework to coordinate a parallel program. MPI is common today. PVM was very popular a decade ago, and some legacy software packages might still require PVM.
Hadoop tries to play the same role as the queueing system, the filesystem, and the parallel programming framework, but instead of individual packages, Hadoop provides a complete stack. Moreover, Hadoop encourages the use of commodity hardware.

Getting Ready

The installation of Hadoop is not trivial. You can find commercial turn-key solutions, and even some cloud vendors have ready-made installation images for Hadoop.
In this article, I describe how to configure a Hadoop cluster with three nodes [3]. One node is used as the master, and two nodes are slaves.

Ubuntu Linux is a great distribution for Hadoop experiments. Of course, you can use any Linux distribution – or even a member of the BSD family. The advantage of using Ubuntu Linux is simply the Personal Package Archives (PPAs). As you will see shortly, you must add a number of PPAs to get through the installation. I used Ubuntu 12.04 "Precise Pangolin" for the example in this article.
In principle, you can use any commodity server for your Hadoop cluster. PCs with one processor, 2GB memory, and a 20GB hard drive are fine for testing and experimenting. If you are short of hardware, you might use a cloud service. It is possible to run a cluster on one server only, but distributed computations and filesystems are a bit more fun when you add the network layer of complexity.
As I mentioned, Hadoop is written in Java. The Hadoop developers recommend you use Oracle Java together with Hadoop. The quality of the open source alternatives (OpenJDK in particular) is rapidly increasing, and you might not run into trouble, but to be on the safe side, start by installing Oracle Java. The only little obstacle is that Ubuntu Linux does not provide packages for Oracle Java because of licensing issues, so Java is the first PPA to add.
Although you can find more than one Oracle Java PPA, a good one is *webupd8team*, which provides a set of scripts to download and install Oracle JRE and JDK 7. Four commands will install Java on you servers:

```
$ sudo add-apt-repository ➥
  ppa:webupd8team/java
$ sudo apt-get update
$ sudo mkdir -p /usr/lib/mozilla/plugins
$ apt-get install oracle-jdk7-installer
```

The `mkdir` command is required because the installer script will install a plug-in for Firefox – even though Firefox is not installed. The folder is required for a smooth installation.
The four commands must be executed on all machines in your Hadoop cluster. If you plan a very large cluster, I highly recommend you find a method for automatic deployment. Notice also that you are installing the Java development kit (JDK). As you learn later in this article, you are going to use a Java compiler; the Java runtime environment (JRE) is not enough.

Installation

Java is an essential component for Hadoop. Ubuntu Linux has packages for Hadoop, but they are outdated. Fortunately, a group of Ubuntu/Hadoop users maintain a set of Ubuntu packages you should definitely use with PPA.
The Hadoop PPA is divided into a number of repositories. It is possible to install bleeding-edge packages, but for most situations, the stable packages are recommended. These packages are very close to the upstream versions.
You must enable the Hadoop PPA on all computers in your cluster and install the packages. You can do so in three commands:

```
$ sudo apt-add-repository ➥
  ppa:hadoop-ubuntu/stable
$ sudo apt-get update
$ sudo apt-get install hadoop pig hive hbase
```

The last command installs Hadoop and three applications (Pig, Hive, and HBase). The three applications were developed using Hadoop as the framework.
Once you have installed the packages, it is a good idea to create a special user to control access to

Hadoop. For this article, I will use the user name `hduser`. Throughout your cluster, you must execute the following command:

```
$ sudo adduser --group hadoop hduser
```

The user `hduser` is not configured to locate Java and the Hadoop executable, which means that any command related to Hadoop will fail. On an Ubuntu Linux computer, with its default Bash shell, add the following lines to the `.bashrc` configuration file:

```
export HADOOP_HOME=/usr/lib/hadoop
export JAVA_HOME=/opt/java/64/jre1.7.0_05/
export PATH=$PATH:$HADOOP_HOME/bin
```

Of course you must check to see whether your Java installation is in `/opt/java/64/jre1.7.0_05`. It might also be worth considering creating a symbolic link and updating the link when the Java packages are updated. The `.bashrc` file must be replicated to all computers.

Hadoop uses the network within the cluster to distribute files and schedule tasks. A common mistake is that every computer in the cluster has the same name (often `localhost.localdomain`). You can use IP addresses in the configuration files, but proper names are easier to remember. For a small cluster, you can change the computer name in the file `/etc/hostname` and reboot the computer. Moreover, the names of all computers in the cluster can be listed in `/etc/hosts`. If your cluster is large, it is probably a good idea to use a local name and a DHCP server to manage the hostnames and IP addresses.

Most of the scripts to start up and shut down the components of a Hadoop cluster are executed at the master node. Secure Shell (SSH) is a simple and secure solution for executing a script on a remote computer. To execute a script without typing a password, you must use public/private keys without a pass phrase.

A security expert will probably not recommend that you generate an SSH key pair at the master computer and copy it to the rest of the cluster, but remember, your Hadoop installation is only going to run on a private network, and the security risk is minimal.

Configuration

The configuration presented in this article is a bare minimum. Hadoop has a broad set of settings and parameters. Depending on your work load and a zillion other things, you can tune Hadoop to perform better than the default configuration. The configuration you use must be replicated to all computers in the cluster at all times.

The distributed filesystem will spread the files in smaller chunks or blocks throughout the cluster. Each machine in the cluster will receive some blocks. These blocks must be stored somewhere on the physical hard drive in the machine. Keep in mind that the distributed filesystem is not an ordinary Linux filesystem, and you cannot access it using common command-line tools like `ls` and `cat`.

Even though you cannot access the files in the distributed filesystem directly, you must create a directory for it on the local hard drive. The `hduser` account comes in handy because you can use it to restrict access to blocks stored locally. The following two commands create a directory and set the restrictions.

```
$ sudo mkdir -p /app/hadoop/tmp
$ sudo chown hduser.hadoop /app/hadoop/tmp
```

The file name `/app/hadoop/tmp` is chosen randomly. If you want another location, feel free to change it. In a Hadoop cluster for very large data sets, you can install a special hard drive for the distributed filesystem. Moreover, it is worth considering which Linux filesystem to use and its parameters. You can gain some performance by tuning the filesystem parameters. The Hadoop block size (64MB by default) can affect the parameters of the Linux filesystem.

The configuration files for Hadoop are stored in the `/etc/hadoop/conf` directory. The file `masters` tells Hadoop which computer is the master, and the `slaves` file list all the computers that can execute tasks. For a small cluster, the master computer can be a slave. For a large cluster, the master computer will use so much processing power for scheduling tasks that it can hardly contribute to task processing.

Because this example installs a three-node cluster, you should add the master to the `slaves` file. It is possible to add extra computers and remove retired computers with a command-line utility. You should see `slaves` as the initial set of computers in your cluster.

Three configuration files are required to start Hadoop. Because Hadoop is written in Java, it will probably be no surprise that these three configuration files are in XML format.

The first configuration file is `core-site.xml`. The most important parameter (or property, as it is call in the Hadoop language) specifies the location where the distributed filesystem can store the blocks. The directory `/app/hadoop/tmp` was create previously, and you must use this directory in your configuration file.

```
<property>
   <name>hadoop.tmp.dir</name>
   <value>/app/hadoop/tmp</value>
</property>
```

The distributed filesystem replicates the blocks; that is, it stores the blocks on more than one computer. The number of copies is controlled by the `hdfs-site.xml` configuration file. Because the cluster consists of three computers, I can have three copies of the blocks:

```
<property>
   <name>dfs.replication</name>
   <value>3</value>
</property>
```

It is possible to have either a single computer or two computers. If you choose to install a smaller cluster, you must adjust the value in the `hdfs-site.xml` configuration file.
The last configuration file is `mapred-site.xml`. Use this file to list the machine that acts as a scheduler for MapReduce jobs. The job tracker listens to a TCP port to communicate with the clients, and the port must be included. Port 54311 is the default:

```
<property>
   <name>mapred.job.tracker</name>
   <value>master:54311</value>
</property>
```

The configuration file to customize for your cluster is `hadoop-env.sh`. The most important parameter is `JAVA_HOME` – the same environment variable used with the `hduser` user to define the Java directory – and you must repeat it in this configuration file.
Before starting Hadoop for the first time, you must prepare or format the distributed filesystem. To format the filesystem, log in as `hduser` and execute the following command on the master:

```
$ hadoop namenode -format
```

Finally, your Hadoop cluster is ready to launch. On the master computer, execute the following two commands:

```
$ start-dfs.sh
$ start-mapred.sh
```

The first command starts the distributed filesystem, and the second command starts the scheduler and related processes. The two scripts will start relevant processes remotely on the slaves.
If you want to shut down your cluster, use the commands `stop-dfs.sh` and `stop-mapred.sh`.

Test Data

In the previous sections, the installation and configuration of your Hadoop cluster was the focus. Now it is time to put the system to work. Hadoop lets you analyze large data sets, and provides a web interface for monitoring the cluster (Figures 1 and 2).
You might already have a large data set you want to analyze. To make the following section a bit more interesting, in this section, I show you how to generate a test data set.
The Bureau of Transportation Statistics in the United States collects data on every flight in the country. This data set is publicly available [4], and you can download this flight data set month by month. About 110 different values are recording for every flight, and you will find about 500,000 flights every month. For every flight, you can find the date of the flight, the origin, the destination, and whether the flight was delayed.
When you download a month of flight data, you get a ZIP file. In this ZIP file, you find a general description (`readme.html`) and the data. Each flight is represented by a line with comma-separated values. A file can be prepared for Hadoop by using the following commands:

```
$ unzip -o ➥
  On_Time_On_Time_Performance_2012_?.zip
$ grep ^[0-9] ➥
  On_Time_On_Time_Performance_2012_?.csv ➥
  | tr -d"\"" > ➥
  On_Time_Performance_2012_1H.csv
```

The `grep` command is required because each file begins with a header. The rest of the line begins with the year of the flight. The values are quoted, and the `tr` command removes the quotes. The file `On_Time_Performance_2012_1H.csv` is about 1.3GB in size and consists of a little more than 3 million lines. This might sound like a large data file, but it is actually relatively small in the world of business.
The `On_Time_Performance_2012_1H.csv` file is split into 21 blocks (1.3GB/64MB). The blocks are spread over three nodes (master and two slaves) so each computer in the cluster will have seven primary blocks, but each computer will also receive the other computers' primary blocks as backup. This implies that copying a 1.3GB file to the distributed file system will give rise to a lot of network activity.
When you copy the data file to Hadoop's filesystem, the splitting and replication begin. As you can imagine, a file of 1.3GB takes some time to distribute. You copy the file to the Hadoop distributed filesystem with the following command:

```
$ hadoop dfs -copyFromLocal ➥
  On_Time_Performance_2012_H1.csv /
```

The first time you copy a file to your cluster, it can be fun to watch the logfiles at one of the

slaves. Java applications are known for their ability to produce many log messages, and Hadoop is not an exception to that rule.

MapReduce

Distributed and parallel programming are old disciplines of computer science. It is inherently difficult to develop a program that spans more than one processor, and it does not matter whether the processors are located in the same computer or separated by a network.

Automatic parallelization has been regarded as the holy grail in computer science and compiler development for decades. No great breakthrough has occurred in this area despite the vast amounts of resources spent.

Hadoop is an attempt to address the parallelization problem. It does not pretend to solve the general problem, but Hadoop is a parallelization platform or framework. For a software developer, Hadoop presents a certain technique or paradigm. This paradigm is not new, and it goes under many names, but in Hadoop, it is called MapReduce. The term was originally coined by Google for their implementation of a programming model for big data, and it is a common design pattern in functional programming. The idea of MapReduce is to transform an algorithm into two phases.

The *map* phase takes a subset of the data set and maps it. The mapping consists of finding the relevant data and performing some computation on it. The mapping is done by a number of independent mappers. In Hadoop, the mappers are executed as tasks on the slaves of the cluster.

The *reduce* phase combines the mapped data from each mapper. It is important to notice that the mappers work independently. This means that not all algorithms will fit the MapReduce paradigm. In the HPC world, such algorithms are often referred to as embarrassingly parallel.

Consider the following simple example: calculate the sum of a number of numbers. It is possible to break down the sum into a number of sums of a subset of the numbers. Say you want to add 100 numbers. You can give 10 numbers to 10 persons and ask them to add them. Each person is a mapper, and they work independently. When they finish, they give you the subtotals, and you can now add 10 numbers to find the total. You act as the reducer.

As you can see, the MapReduce implementation of the summation problem will give you a speed-up of almost 10 times if you have 10 persons and almost 15 times if you have 15 persons. Although distributing the workload has an over-

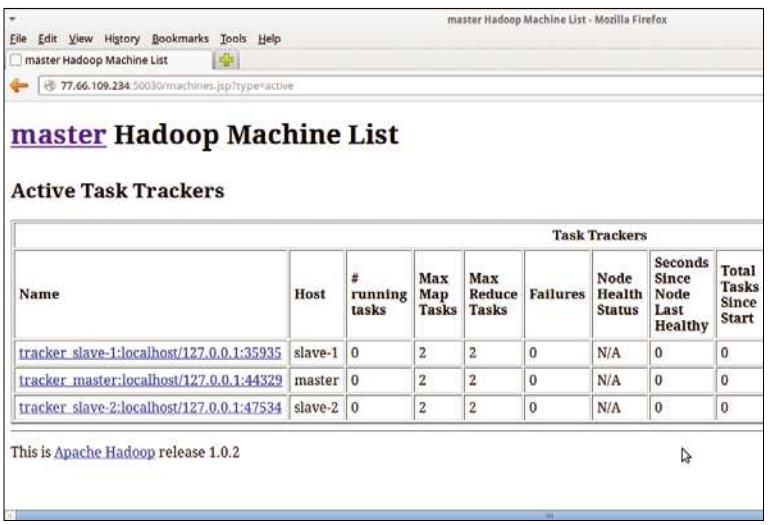


Figure 1: Hadoop provides a web interface for monitoring your cluster.

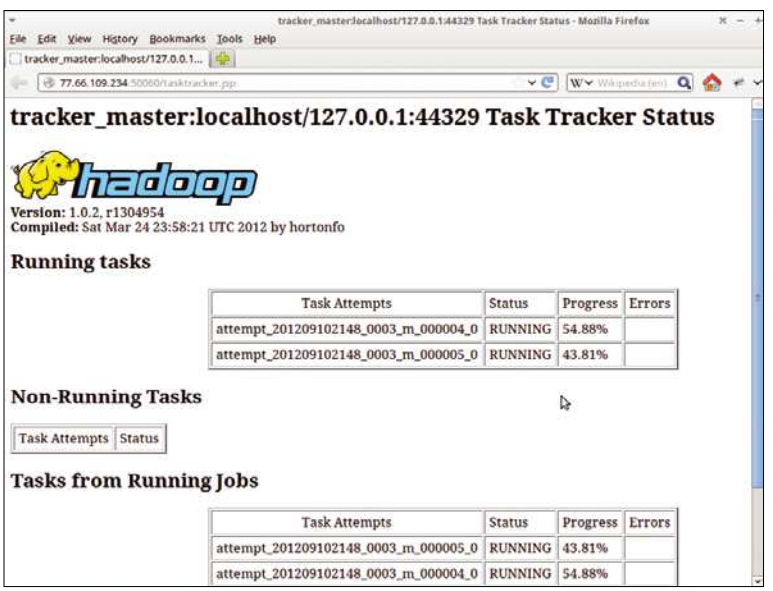


Figure 2: Use your web browser to monitor running tasks. This cluster currently has two running tasks.

head, if the map phase takes much longer than the reduce phase, the MapReduce paradigm promises you an almost linear speed-up. The interesting bit is that Hadoop automatically breaks your job down into map and reduce tasks. As files are split into blocks and distributed to the computers in the cluster, Hadoop will break down the job into tasks that operate on blocks.

The test data presented in the previous section can be analyzed by a MapReduce algorithm. Consider the following query: You want to compute how many times every airport has had an arrival in the first six months of 2012. If you know SQL, you might load the data set into a relational database and perform the following query (assuming each value in the data set is presented as an attribute in a table):

```
SELECT Dest, COUNT(*) ➥
  FROM OnTime GROUP BY Dest
```

The attribute `Dest` is the destination airport. In the data set, the airport is stored using the three-letter airport code. For example, LAX is Los Angeles International Airport and DEN is the Denver Airport. Implementing the query using MapReduce is almost trivial. Listing 1 shows a Hadoop implementation. The map phase will emit the airport code and the constant `1` for every occurrence. The data file consists of comma-separated values, and the mapper must split the line and find the airport code. In Listing 1, the map phase is implemented by the subclass `Map`. The reduce phase sums the value for each airport. This means the reducer adds a numbers of 1s and emits the result. In Listing 1, you find the reduce phase as subclass `Reduce`.

To optimize the query, a combiner is introduced. A combiner is a reducer running at the slave. Instead of transferring many 1s, the combiner adds the 1s and emits the subtotal to the master. A combiner might not always be possible, but in this case it is. In the method `main` you find how the mapper, combiner, and reducer are added. The Hadoop implementation of the query is done in Java. Before you can submit it to Hadoop, you must compile and link it. The following two commands do this:

```
$ javac -classpath ➥
    /usr/lib/hadoop/hadoop-core-1.0.2.jar ➥
    -d ontime_classes OnTime.java
$ jar -cvf ontime.jar ontime_classes
```

The distributed filesystem does not allow overwriting files. You must delete old output files before executing your MapReduce job. After your query has been executed, you can find the output in the distributed filesystem. During the execution of the job, Hadoop will print where to find the output. Combining these steps leads to the following commands:

```
$ hadoop dfs -rmr /output
$ hadoop jar ontime.jar ➥
   com.linuxmagazine.OnTime ➥
   /On_Time_Performance_2012_H1.csv /output
$ hadoop dfs -cat /output/part-00000
```

Applications

During the installation of your Hadoop cluster, you installed three applications: Pig, Hive, and HBase. Pig is a programming language that takes advantages of the MapReduce capabilities of Hadoop. It is a data flow language that does not look like C++ or PHP. It is such closer to functional programming. Functional programming languages have one very nice feature: functions have no side effects. This implies that a program written in a functional language is much easier to parallelize. Besides that, functional and data flow languages might also be easier for statisticians and mathematicians to learn. A Pig program is rewritten as a set of MapReduce tasks. It is possible to load CSV files in Pig and perform operations. The query outlined in the previous sections can be written as:

```
A = LOAD '/On_Time.csv' AS ➥
    (Year:chararray,Month:chararray, ➥
    DayofMonth:chararray, Dest:chararray);
B = GROUP A BY Dest;
C = FOREACH B GENERATE COUNT(A);
```

Hive is a data warehouse that gives you an SQL-like syntax for querying the data. The queries are automatically rewritten into MapReduce tasks. You can create tables and load CSV files into them. Your famous test data can be analyze by the following three statements. You will notice that the third statement – the actual query – is very like the SQL statement you saw in the description of the test data.

```
CREATE TABLE OnTime ➥
    (Year STRING, Month STRING,DayofMonth ➥
    STRING, Dest STRING) ➥
        ROW FORMAT DELIMITED FIELDS   ➥
    TERMINATED BY '44' LINES ➥
    TERMINATED BY '\n';

LOAD DATA LOCAL INPATH 'On_Time.csv' ➥
    OVERWRITE INTO TABLE OnTime;

SELECT Origin,COUNT(*) FROM OnTime ➥
    GROUP BY Origin;
```

Summary

Hadoop is a platform for distributed storage and computation.

You can say that the parallelization is aligned with the distribution of data. It is not a completely automated parallelization, but Hadoop adds many nifty tools and techniques for parallel programmers.

If your data set can fit within the physical memory of a computer, you should write a single program that reads from memory. But if you have very large data sets, Hadoop might be worth the effort. Hadoop is not the holy grail of parallel programming, but it will offer you help in getting closer. Applications like Pig and Hive take you a step further by hiding the complexity of distributed and parallel computation.

The MapReduce paradigm is very strong if your solutions can be described in terms of it. As Hadoop progresses, new applications will help decision makers make better decisions. ■

Info

[1] "How Target Figured Out A Teen Girl Was Pregnant Before Her Father Did": http://www.forbes.com/sites/kashmirhill/2012/02/16/how-target-figured-out-a-teen-girl-was-pregnant-before-her-father-did/

[2] SDSS: http://www.sdss3.org/dr9/

[3] Michael Noll Hadoop Tutorials: http://www.michael-noll.com/tutorials

[4] TransStats: http://www.transtats.bts.gov/Fields.asp?Table_ID=236

[5] White, T. *Hadoop: The Definitive Guide*, 2nd ed. O'Reilly Media, 2010.

[6] George, L. *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[7] Gates, A. *Programming Pig*. O'Reilly Media, 2011.

[8] Lin, J., and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool, 2010.

The Author

Kenneth Geisshirt is a software developer and tech writer living in a suburb of Copenhagen, Denmark, with his wife and two children. He is a chemist by education and a geek by nature. He has been a free software user, developer, and advocate since the early 1990s. When he is not talking astronomy with his son and mathematics with his daughter, he is working for a small start-up company focusing on NoSQL databases and big data.

Listing 1: OnTime.java

```
01 package com.linuxmagazine;
02
03 import java.io.*;
04 import java.util.*;
05
06 import org.apache.hadoop.fs.Path;
07 import org.apache.hadoop.filecache.DistributedCache;
08 import org.apache.hadoop.conf.*;
09 import org.apache.hadoop.io.*;
10 import org.apache.hadoop.mapred.*;
11 import org.apache.hadoop.util.*;
12
13 public class OnTime {
14    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {
15      private final static IntWritable one = new IntWritable(1);
16
17      public void map(LongWritable key, Text value,
          OutputCollector<Text, IntWritable> output, Reporter reporter)
          throws IOException {
18        String line = value.toString();
19        String[] strArr = line.split(",");   // CSV file
20        Text airport = new Text(strArr[14]); // Dest column
21        output.collect(airport, one);
22      }
23    }
24
25    public static class Reduce extends MapReduceBase implements
        Reducer<Text, IntWritable, Text, IntWritable> {
26      public void reduce(Text key, Iterator<IntWritable> values,
          OutputCollector<Text, IntWritable> output, Reporter reporter)
          throws IOException {
27        int sum = 0;
28        while (values.hasNext()) {
29          sum += values.next().get();
30        }
31        output.collect(key, new IntWritable(sum));
32      }
33    }
34
35    public static void main(String[] args) throws Exception {
36      JobConf conf = new JobConf(OnTime.class);
37      conf.setJobName("ontime");
38
39      conf.setOutputKeyClass(Text.class);
40      conf.setOutputValueClass(IntWritable.class);
41
42      conf.setMapperClass(Map.class);
43      conf.setCombinerClass(Reduce.class);
44      conf.setReducerClass(Reduce.class);
45
46      conf.setInputFormat(TextInputFormat.class);
47      conf.setOutputFormat(TextOutputFormat.class);
48
49      FileInputFormat.setInputPaths(conf, new Path(args[0]));
50      FileOutputFormat.setOutputPath(conf, new Path(args[1]));
51
52      JobClient.runJob(conf);
53    }
54 }
```

Splunk's Stephen Sorkin describes Splunk's exciting new Hunk analytics software for Hadoop

Q&A with Stephen Sorkin

Splunk's new Hunk analytics software makes it easy to extract and visualize data in Hadoop. We sat down with Splunk Chief Strategy Officer Stephen Sorkin to find out more about Hunk and what it will mean for big data.

ADMIN Magazine: Splunk just launched a new product known as Hunk. What is Hunk and what is it for? How will it be used?

Stephen Sorkin: Hunk is an integrated analytics platform that will enable everyone in your organization to rapidly explore, analyze, and visualize data in Hadoop. Hunk combines the Splunk Search Processing Language (SPL™), interactive user interface, REST API, and SDKs for data at rest in Hadoop. We built it so that customers with large amounts of historical data stored in Hadoop can quickly get value through exploring, analyzing, reporting, and sharing.

AM: Exploring, analyzing, reporting, and sharing? Bring that image to life for us. What kind of data will Hunk be used for? Could you describe a specific scenario where someone would want to use Hunk to obtain information?

Sorkin: Splunk's emphasis is on machine-generated big data, like logfiles, clickstream data, geolocation readings, and sensor data. Splunk Enterprise provides real-time and historical analysis of this machine data from disparate sources. It takes care of everything – managed forwarders, indexing, storage, analytics, visualizations, and dashboards, all from a single product.
With Hunk, we're applying that concept to data in Hadoop. Hunk is especially powerful for large-scale analytics and ad hoc exploration of data, for example, in analyzing customer behavior on web or mobile apps over months or years. Another use would be to mine data from firewall, proxy, and other logs to find and understand modern security threats. With Hunk, you can conduct deep analysis, detect patterns, and find anomalies across terabytes or petabytes of raw data without specialized training or fixed schemas. Hunk works with Apache Hadoop or your choice of Hadoop distribution.

AM: What is Splunk Virtual Index and how does it fit in with Hunk?

Sorkin: The Splunk Virtual Index is the bridge from our Search Processing Language to data stored in an external system. This patent pending technology translates a Splunk search into a MapReduce job for Hadoop and lets you access the result of the job as it's being generated. You don't need to know anything about MapReduce. In the future, Hunk Virtual Indexes might extend to other data systems as well, like Apache Cassandra, MongoDB, HBase, and even relational databases.

AM: As I understand it, Pig is the data analysis framework built into Hadoop, and I guess Map and Reduce functions also provide a means for obtaining data. How does Hunk fit in with these tools?

Sorkin: Pig (like Hive) is a higher level abstraction in Hadoop to avoid MapReduce programming directly. Hunk translates SPL directly into MapReduce. In both Splunk Enterprise and Hunk, every search comes with its own unique schema. This means you don't

need to know anything about the data in advance. With Hunk, even event breaking and timestamp extraction are done at search time. When you ask Hunk a question, it starts bringing back the answer immediately, while the MapReduce job is running. This approach allows you to search interactively by pausing and refining queries.

AM: What's the learning curve for Hunk? Will users need to learn a new scripting language? How long will it take for a company that already has Hadoop to set up Hunk, learn to use it, and start getting productive results from it?

Sorkin: Setting up Hunk is as easy as setting up Splunk Enterprise, and we take pride in the short amount of time it takes our customers to get value from their data. You can download either Splunk Enterprise or Hunk and install it in minutes, and our interactive UI makes it very easy to start interacting with data immediately. Of course, it takes time to become a master of SPL, but you can accomplish most simple tasks by pointing and clicking, or by downloading an app from the Splunk apps website [1].

The developer framework enables you to integrate data and functionality from Hunk into enterprise big data applications using a standards-based web framework, a documented REST API, and software development kits (SDKs) for Java, JavaScript, C#, Python, PHP, and Ruby. Developers can build big data apps on top of data in Hadoop with custom dashboards, flexible UI components, and custom visualizations for common development languages such as JavaScript, Django, and Python.

AM: I understand that Hunk for Hadoop is just one of several Hunk variants you have planned (e.g., Hunk for Cassandra, Hunk for MongoDB, etc.). Is your ultimate goal to abstract the details of underlying data store to create a universal platform for big data analytics?

Sorkin: We think that SPL and our UI are ideal for interacting with raw, unstructured, and polystructured big data, and for enriching results with data from relational databases. By extending our Virtual Index support to NoSQL data stores, we can make it easy for our customers to get value from the data at rest and to achieve the benefits of combining the data from many different stores for more sophisticated analysis.

AM: This is the first release of Hunk, but developers are always looking down the road. Do you have new features in mind for later releases? Where would you like Hunk to be in two to three years?

Sorkin: Beyond support for systems other than the Hadoop Distributed File System (HDFS), and proprietary variants like MapR, we think that we can do a lot to make the art of data exploration easier, both for HDFS specifically and for data in general.

Figure 1: Running a query in Hunk.

Hunk Product Tour

Hunk is built upon a schema-on-the-fly technology, which means you don't need to know anything about the structure of data in advance. Search results return with an automatic structure based on fields, keywords, patterns over time, top values, and more.

"The combination of Splunk software with the processing power of MapReduce and Hadoop generates transparent and sustainable competitive advantage for your company within minutes." – Marcus Buda, Senior Data Architect, Otto Group

otto group

Results Preview

When you run a query in Hunk (Figure 1), it streams back interim results immediately while the MapReduce job continues to run in the background. The ability to pause and refine queries without having to wait for full MapReduce jobs to finish delivers a faster, more interactive experience.

Drag-and-Drop Analytics

Hunk empowers business and IT teams to analyze raw data in Hadoop. Data models describe relationships in the underlying raw data, making it more meaningful

and usable. You can quickly generate charts, visualizations, and dashboards using the Pivot interface, without the need to master SPL. Drill down from anywhere in a chart to the underlying raw events or to another dashboard, form, view, or external website. Save reports, integrate them into dashboards, and then view them all from your desktop or mobile device (Figure 2).

"I'm super excited about Hunk. Hunk is solving one of the top issues that our customers have – access to the skills and know-how to leverage the data inside of Hadoop. Splunk has a very beautiful UI that is very easy to learn. So it bridges that gap and makes it very easy to access the data inside of Hadoop." – *Dr. Amr Awadallah, Co-founder and CTO, Cloudera*

Custom Dashboards and Views

Business and technical colleagues can edit dashboards with a simple interface and change chart types on the fly with integrated charting controls. Hunk dashboards combine multiple charts and views of your data in Hadoop to satisfy the needs of multiple business and IT stakeholders (Figure 3). Use the ODBC driver (beta) to integrate with third-party data visualization and business intelligence software. Role-based access controls protect sensitive data.

Rich Developer Framework

The web framework enables developers to integrate data and functionality from Hunk into enterprise big data applications using a standards-based web framework (Figure 4), documented REST API, and software development kits (SDKs) for C#, Java, JavaScript, PHP, Python, and Ruby. Developers can build Hunk apps with custom dashboards, flexible UI components, and custom data visualizations using common development languages such as JavaScript, Django, and Python. With Hunk, building apps on top of HDFS is as easy as building any modern web application. Learn more at the Splunk developer portal [2]. ■

Figure 2: Integrate and visualize your data with custom reports.

Figure 3: Hunk dashboards let you view your data at a glance.

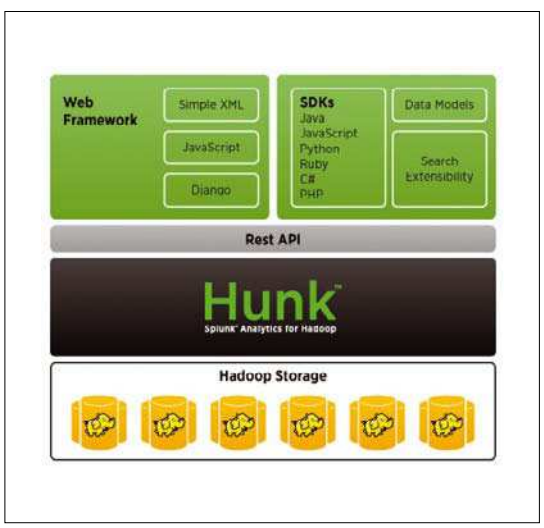


Figure 4: Hunk's standards-based web framework is easy to integrate and extend.

Info

[1] Splunk apps: [apps.splunk.com]

[2] Splunk dev: [dev.splunk.com/hunk]

Raw Data to Analytics in <60 Minutes

In Good Time

Building on Splunk's years of experience with big data analytics technology, Hunk brings dramatic improvements in speed and simplicity for users who are looking for answers in Hadoop. By Ledion Bitincka

Hunk™: Splunk Analytics for Hadoop [1] is a full-featured, integrated analytics platform that enables you to explore, analyze, and visualize data interactively in Hadoop. Hunk empowers everyone in your organization to derive actionable insights quickly and easily from raw, unstructured, and polystructured data in Hadoop. Hunk works with Apache Hadoop [2] or with your Hadoop distribution provider (e.g., Cloudera [3], Hortonworks [4], IBM [5], MapR [6], or Pivotal [7]).

In this article, which is adapted from one of my posts to the Splunk blog [8], I'll show you how, in just one hour, you can go from downloading Hunk to analyzing raw data in Hadoop.

> *"The fact that Splunk is bringing ease of use to sophisticated Hadoop problems is welcome from every angle. The power of Hunk comes across in how easy it is to just plug it in, throw it in there, and suddenly have all of your answers. I wish every product worked this nicely."* M.C. Srivas, Co-founder and CTO, MapR

Minutes 0-20: Set Up the Environment

To get up and running with Hunk, you'll need the following software packages available or installed in the server running Hunk:

- Hunk bits – Download Hunk [9] and try it for free for 60 days.
- A virtual or physical 64-bit Linux OS on which to install Hunk.
- Java – At least version 1.6, or whichever version is required by the Hadoop client libraries.
- Hadoop client libraries – You can get these from your Hadoop vendor, or if you're using the Apache downloads, you can fetch the client libraries from the Apache archive [10].

When you download the Hunk 60-day free trial, you have the option to pick from `.rpm`, `.deb`, or `.tgz` file formats. The instructions that follow refer to the `.tgz` download.

To install the Hunk bits, untar the package:

```
tar -xvf ?
  splunk-6.0-<BUILD#>-Linux-x86_64.tgz
```

To start Splunk, type:

```
./splunk/bin/splunk start
```

Next, download and follow the instructions for installing or updating Java and the Hadoop client libraries. Make sure you keep note of `JAVA_HOME` and `HADOOP_HOME`, because you will need them in the next section.

Minutes 20-40: Configure Hunk

To configure Hunk, you can either go to *Settings* | *Virtual indexes* and use the manager user interface or edit the configuration files at `indexes.conf`. Here, I'll go through the user interface method. For examples on editing the conf files, refer to my blog [8].

1. Log in to Hunk (Figure 1; default user: *admin*, password: *changeme*).

Figure 1: **Logging in to Hunk.**

2. Navigate to *Settings* | *Virtual indexes*.
3. Create an external results provider.
4. Specify environment information, including Java home, Hadoop home, and some cluster information, starting with the Hadoop version, JobTracker host port, and default filesystem (Figure 2).
5. After saving the provider, you can then move on to creating virtual indexes for this provider by switching to the *Virtual indexes* tab after saving the Hadoop provider (Figure 3).
6. The main configuration requirement for a virtual index is a path that points to the data you want the virtual index to represent. You can optionally specify a whitelist of regular expressions (regex) that matches only the files you want to be part of the virtual index. If the data is partitioned by time, you can also tell Hunk about how the time partitioning is implemented (see the next section for more information about time partitioning).
7. After saving the virtual index, you can immediately start exploring its content by clicking *Search* (Figure 4).

Time Partitioning and Schematization

Time is a an important concept in Splunk software. If your data is organized in a directory structure using date partitioning – and this is a very common practice – you can configure Hunk to recognize the date partitioning scheme and use it to optimize the search experience. The logic goes like this: Match a regular expression against the path, concatenate all the capturing groups, and interpret that string using the given format string; finally, add or subtract a number of seconds (offset) from the resulting time. The offset comes in handy when you want to extend the extracted time range to build in some safety. For example, a few minutes of a given day might end up in the next/previous day's directory, or the directory structure could be in a different time zone from the Hunk server).

Hunk does the whole time extraction routine twice to come up with a time range – that is, extract an earliest time and a latest time. When the time range extraction is configured, Hunk is able to ignore directories or files that fall outside of the search time range. In Hunk-speak, this is known as time-based partition pruning.

If the data is unstructured, you can tell Hunk how to schematize it at search time. At this point, you're entering classic Splunk software setup and configuration. One way for Hunk to bind a schema to the data is to assign an already defined `sourcetype` to the

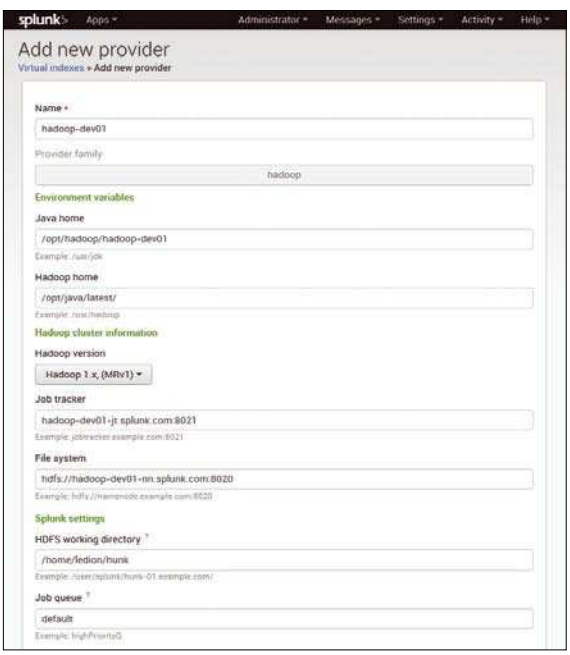


Figure 2: **Entering environment information.**

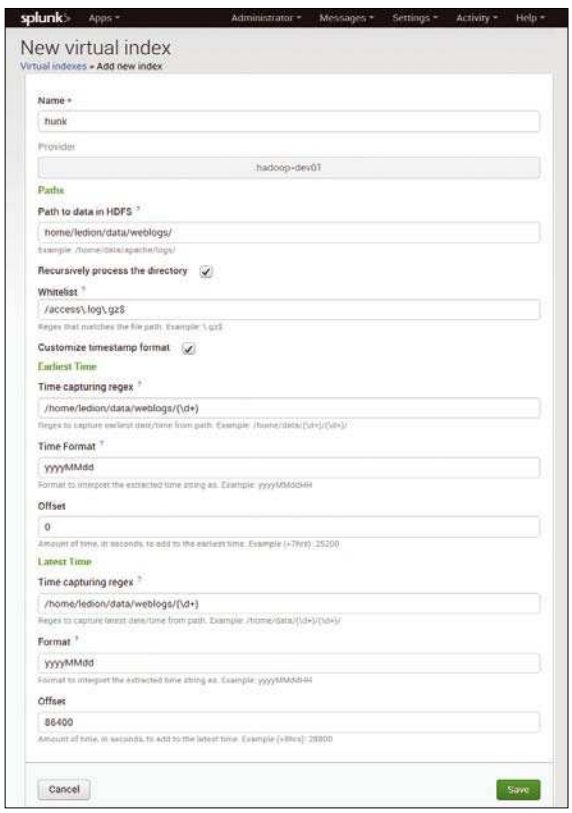


Figure 3: **Creating provider virtual indexes.**

Figure 4: Click Search to explore a virtual index.

data. Hunk ships with a few predefined sourcetypes; one of them is `access_combined`, a sourcetype used to interpret Apache web logs. The `access_combined` sourcetype is defined in `$SPLUNK_HOME/etc/system/default/props.conf`, and it defines how access log data should be processed (e.g., defining that each event is a single line, finding the timestamp, and extracting fields from the raw event).
In the file `$SPLUNK_HOME/etc/system/local/props.conf`, tell Hunk to assign sourcetype `access_combined` to all the data in your virtual index,

```
[source::/home/ledion/data/weblogs/...]
sourcetype = access_combined
```

(i.e., all the data under `/home/ledion/data/weblogs/`).

Minutes 40-59: Analyze Your Hadoop Data

Now you are ready to start exploring and analyzing data in Hadoop. Hunk runs searches against the virtual index data as if it were a native Splunk Enterprise index. I'm going to show two examples: highlighting data exploration and analytics.
To begin, explore the raw data by typing:

```
index=hunk
```

To get a chart showing the status codes over a 30-day window using daily buckets, refine your query by typing

```
index=hunk | timechart ➥
   span=1d count by status
```

Minutes 59-∞: Keep on Splunking!

With Hunk, you have an unlimited number of ways to slice, dice, and analyze your data in Hadoop. For information on how to use the Splunk Processing Language (SPL™), refer to the Splunk software documentation [11] and the tutorial on data models and Pivot [12]. To learn more about how you can use Hunk to search images stored in a Hadoop cluster, refer to the Splunk blog on images, Splunk, and Hunk [13]. Happy Splunking! ■

Info

[1] Hunk: [http://www.splunk.com/view/hunk/SP-CAAAH2E]
[2] Apache Hadoop: [http://hadoop.apache.org/]
[3] Cloudera: [http://www.cloudera.com/]
[4] Hortonworks: [http://hortonworks.com/]
[5] IBM InfoSphere BigInsights: [http://www-01.ibm.com/software/data/infosphere/biginsights/]
[6] MapR: [http://www.mapr.com/products/mapr-editions]
[7] Pivotal: [http://www.gopivotal.com/products/pivotal-hd]
[8] Adapted from the Splunk blog of the same title: [http://bit.ly/190bbRN]
[9] Hunk download: [www.splunk.com/download/hunk]
[10] Hadoop client libraries: [http://archive.apache.org/dist/hadoop/core/]
[11] SPL: [http://bit.ly/1bRPrp0]
[12] Data models and Pivot: [http://bit.ly/13Vpbv]
[13] Images with Splunk, Hunk: [http://bit.ly/1fjg1KV]

The Author

Ledion Bitincka is a Principal Architect at Splunk, where he has worked since 2007. Hunk was a side project that soon grew into a fully supported commercial product. Bitincka conceived, designed, and led the software development of Hunk™: Splunk Analytics for Hadoop. Hunk is a new product from Splunk that combines the Splunk Search Processing Language (SPL™), data models, and the Pivot interface with the scale and power of Hadoop.

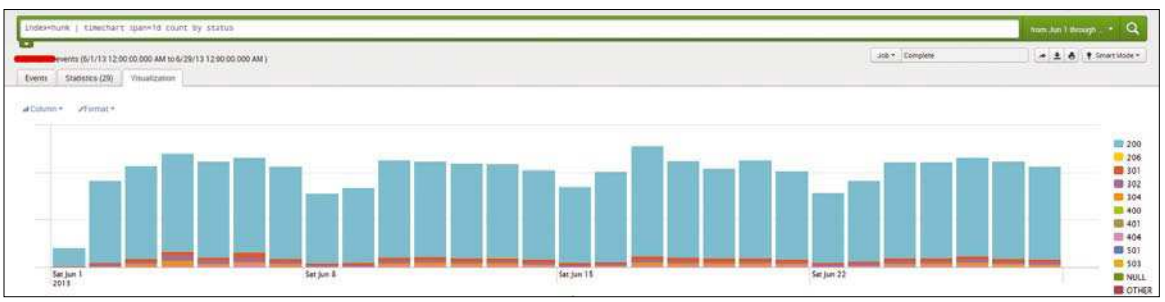


Figure 5: Analyzing Hadoop data.

Exploring the business value of Hadoop

Data Talks

Hortonworks has helped many companies adopt and implement an Apache Hadoop strategy over a number of years. We have helped many companies to learn about Hadoop technology, and we have also learned a lot about data and how to use Hadoop to create big data value across the enterprise. Hadoop is a critical piece of an emerging modern data architecture. You can collect massive amounts of social media data, clickstream data, web logs, financial transactions, videos, and machine/sensor data from equipment in the field. These "new" data sources all share the common big data characteristics of volume (size), velocity (speed), and variety (type) – characteristics once considered low to medium value or even *exhaust data*: too expensive to store and analyze. And it is this type of data that is turning the conversation from "data analytics" to "big data analytics" because it offers so much insight for business advantage.

To be clear, these data types are not strictly "new" – most have existed for some time. Text data, for example, has been with us since before King Tut. With Hadoop, businesses are learning to see these types of data as inexpensive, accessible, daily sources of insight and competitive advantage, all from what they were previously deleting, shredding, or saving to tape.

Each of the important types of big data can provide very different value. Consider two of the most popular use cases: clickstream data for a 360-degree view of the customer and server log data for fraud prevention, security, and compliance [1].

Clickstream Data

Clickstream data provides invaluable information to the Internet

marketer. Analysts review the clickstream data to discover which web pages visitors view, and in what order. This data is the result of a succession of mouse clicks (the clickstream) that a website visitor executes. Clickstream analysis can reveal how users research products and also how they complete their online purchases.

Clickstream Analysis

Clickstream data is often used to understand how website visitors research and consider purchasing products. With clickstream analysis, online marketers can optimize product web pages and promotional content to improve the likelihood that a visitor will learn about the products and then click the buy button. With a huge record of actual behavior patterns, web marketers can judge the effectiveness of different types of collateral and calls to action – with the confidence that their results are statistically significant and reproducible. For a particular product, a video might cause visitors to purchase more often than would a white paper. For another product, a white paper might outperform a datasheet. Clickstream analysis sheds light on customer behavior during the actual purchase process. With patterns across millions of shopping carts, marketers can understand why clusters of customers abandon a cart at the same point in the purchase process. They can also see which products customers buy together, and then create pricing and promotional strategies to sell the product bundles that their customers define through their online behavior.

But clickstream data is not just for consumer web retailers. Any company can analyze the clickstream to see how well their website meets the needs of customers, employees, or constituents.

Hadoop Makes Clickstream More Valuable

Tools like Adobe Omniture and Google Analytics already help web teams analyze clickstreams, but Hadoop adds three key benefits. First, Hadoop can join clickstream data with other data sources like CRM (customer relationship management) data, customer demographics, sales data from brick-and-mortar stores, or information on advertising campaigns. This additional data often provides much more complete information than an isolated analysis of the clickstream alone.

Second, Hadoop scales easily so that organizations can store years of data without much incremental cost, allowing web and business analysts to perform temporal or year-over-year analysis on clickstreams. Analysts can save years of data on commodity machines and find deeper patterns that competitors might miss. Storing all of the data in the Hadoop data lake makes it easy to join diverse datasets in different ways for different purposes. Analysts can then run the same data analysis again in slightly different ways over time.

Finally, Hadoop makes website analysis easier. Without Hadoop, clickstream data is typically very difficult to process and structure. With Hadoop, even a beginning web business analyst can use Apache Hive or Apache Pig scripts to organize clickstream data by user session and then refine it to feed it to analytics or visualization tools. It is also easy to schedule recurring, periodic assessments and comparisons of behavior. Hadoop makes storing and refining the data easy, so the analyst can focus on discovery. With the combination of the Hortonworks data platform and Hunk: Splunk Analytics for Hadoop, website and ecommerce analysts can rapidly understand clickstream, CRM, demographics, and additional data sources and then share findings with easy-to-build dashboards and visualizations.

"Hunk will help Hortonworks customers explore, analyze, and visualize data in Apache Hadoop, driving more intelligent decisions across the entire organization. Hortonworks is working very closely at an engineering level with Splunk and is pleased to certify Hunk on the Hortonworks Data Platform (HDP) 2.0, including its YARN-based architecture and integration with Apache Ambari for management and provisioning." – Shaun Connolly, Vice President of Corporate Strategy, Hortonworks

Historical Log Data

Large enterprises build, manage, and protect their own proprietary, distributed information networks.

Server logs are the computer-generated records that report data on the operations of those networks. They are like the EKG readings for the network: When a problem exists, a server log is one of the first places the IT team looks for a diagnosis.

The two most common use cases for server log data are network security breaches and network compliance audits. In both of these cases, server log information is vital for both rapid problem resolution and longer term forensics and resource planning.

Hadoop Protects Network Security

Barely a week goes by without news of a high-profile network breach by malicious individuals and groups. Enterprises and government agencies invest vast sums on antivirus and firewall software to protect their networks from malware and outside attacks, and those solutions usually work. But when security fails, Hadoop helps large organizations understand and repair the vulnerability quickly. Hadoop also facilitates root cause analysis to create lasting protection.

Companies often don't know about their system vulnerabilities until they've already been exploited. For example, in 2011, Sony's PlayStation Network was attacked over a three-day period. On the fourth day, Sony suspended the network, which remained offline for nearly a month. Names, birthdays, and credit card numbers from nearly 25 million account holders were stolen. Sony announced the data breach six days after suspending the network. According to Sony, they did not notify the public earlier because they needed the time "to understand the scope of the breach, following several days of forensic analysis by outside experts."

Hadoop can make that type of forensic analysis faster. If an IT administrator knows that server logs are always archived into the Hadoop data lake to join other types of data for historical analysis, she can establish standard, recurring processes to flag any abnormalities. She can also prepare and test data exploration queries and transformations for easy use when she suspects an intrusion.

Preparing for IT Compliance Audits

Of course, regulators write laws to prevent crises like the one suffered by Sony. The following compliance standards require organizations to monitor networks, ensure high levels of security for their confidential assets, and provide network compliance audit reports to auditors:

- Payment Card Industry Data Security Standards (PCI DSS)
- Sarbanes Oxley (SOX)
- Health Insurance Portability and Accountability Act (HIPAA)
- Federal Information Security Management Act (FISMA)
- The Gramm-Leach-Bliley Act (GLBA)

Regulatory bodies require organizations to retain log data for long periods, allowing auditors to authenticate security incidents by checking the audit trails from the log data. Storing years of log data in relational databases can be expensive. Hadoop can be a persistent, low-cost platform to show compliance.

Customers of Splunk Enterprise can archive older machine data in Hadoop as an alternative to a frozen bucket, then use Hunk: Splunk Analytics for Hadoop for ad hoc exploratory analytics of trends and patterns that might point to fraud or advanced persistent threats. Risk management, fraud, and security departments can even search across both native indexes of data stored in Splunk Enterprise and Hunk virtual indexes of historical data stored in Hadoop.

Hortonworks, Hadoop, and You

We encourage those interested in Hadoop to follow us, get engaged with Hortonworks learning tools, or download the Hortonworks Sandbox, a single-node installation of Hortonworks Data Platform (HDP) that can run right on a laptop. Hadoop has the potential to have a profound impact on the data landscape, and by understanding the basics, enterprises and public-sector organizations can greatly reduce the complexity. Download the Hortonworks Sandbox to get started with Hadoop today [2] and reference the Splunk tutorial and video [3] for how to connect Hortonworks Sandbox with Hunk: Splunk Analytics for Hadoop to explore, analyze, and visualize data in Hortonworks Sandbox and HDP. ■

Info:

[1] Adapted from Hortonworks, "Business Value of Hadoop As Seen Through Data." July 2013, ©2013 Hortonworks Inc.

[2] Download Hortonworks Sandbox: [http://hortonworks.com/products/hortonworks-sandbox/]

[3] Tutorial for how to connect Hunk™: Splunk Analytics for Hadoop to Hortonworks Sandbox and Hortonworks Data Platform (HDP): [http://bit.ly/17GN8FT]

Start with big data and Splunk software. End with an unfair advantage.

Splunk software transforms your machine-generated big data into valuable insights faster than you thought possible. Make your business more responsive, productive and profitable with the world's leading platform for machine data. Over two-thirds of the Fortune 100 use Splunk software and have the business results to prove it.

Learn more at splunk.com/listen.

splunk> listen to your data™