JOCHEN L. LEIDNER AND
GARY BEROSIK

building and installing a Hadoop/MapReduce cluster from commodity components: a case study

Jochen L. Leidner, Ph.D., is a research scientist in the corporate Research and Development group at Thomson Reuters and a director at Linguit Ltd. He holds a doctorate degree in Informatics from the University of Edinburgh, where he has been a Royal Society Enterprise Fellow in Electronic Markets and a postdoctoral researcher, and he has master's degrees in Computational Linguistics, English Language and Literature, and Computer Speech, Text and Internet Technology. His research interests include natural language processing, search engines, statistical data mining, and software engineering. Jochen is a member of ACM, ACL, and SIGIR and has co-authored over 20 peer-reviewed papers and several patent applications.

leidner@acm.org

Gary Berosik is a lead software engineer at Thomson Reuters Research and Development and an adjunct faculty member in the Graduate Programs in Software at the University of St. Thomas in St. Paul, MN. His interests include software engineering, parallel/grid/cloud processing, statistical machine learning algorithms, learning-support technologies, agent-based architectures, and technologies supporting business intelligence and information analytics.

gary.berosik@thomsonreuters.com

WE DESCRIBE A STRAIGHTFORWARD way to build, install, and operate a compute cluster from commodity hardware. A compute cluster is a *utility* that allows you to perform larger-scale computations than are possible with individual PCs. We use commodity components to keep the price down and to ensure easy availability of initial setup and replacement parts, and we use Apache Hadoop as middleware for distributed data storage and parallel computing.

Background

At the time of writing, single desktop computers and even mobile devices have become faster than the supercomputers of the past. At the same time, storage capacities of disk drives have been increasing by multiple orders of magnitude. As a result of mass production, prices have decreased and the number of users of such commodity machines has increased. Meanwhile, pervasive networking has become available and has led to the distribution and sharing of data and, consequently, distributed communication, creation, consumption, and collaboration. Perhaps paradoxically, the ever-increasing amount of digital content that is the result of more powerful machine storage and networking is intensifying the demand for information and for making sense of activities, preferences, and trends. The analysis of large networks such as the World Wide Web is such a daunting task that it can only be carried out on a network of machines.

In the 1990s, Larry Page, Sergey Brin, and others at Stanford University used a large number of commodity machines in a research project that attempted to crawl a copy of the entire Web and analyze its content and hyperlink graph structure. The Web quickly grew, becoming too large for human-edited directories (e.g., Yahoo) to efficiently and effectively point people at the information they were looking for. In response, Digital Equipment Corporation (DEC) proposed the creation of a keyword index of all Web pages, motivated by their desire to show the power of their 64-bit Alpha processor. This effort became known as the AltaVista search engine. Later, the aforementioned Stanford group developed a more sophisticated search engine named BackRub, later renamed Google.

Today, Google is a search and advertising company, but is able to deliver its innovative services only due to massive investments in the large-scale

distributed storage and processing capability developed in-house. This capability is provided by a large number of PCs, the Google File System (GFS), a redundant cluster file system, and MapReduce, parallel data processing middleware. More recently, the Apache Hadoop project has developed a reimplementation of parts of GFS and MapReduce, and many groups have subsequently embraced this technology, permitting them to do things that they could not do on single machines.

MapReduce/Hadoop Concepts

The key innovation of Hadoop [3], modeled after Google's MapReduce [11], is to eliminate synchronization problems by imposing a programming model that makes it possible to automatically address synchronization issues under the hood. This distributed programming model was inspired by functional languages like LISP and SML. In functional programming, new data is created from old data without modifying state, and this model is more suitable for parallelization, as synchronization issues are less often an issue. Each MapReduce task comprises two phases: (a) a Map step iterates over a set of data elements (called splits or slices), creating a key-value pair representation, and, optionally, carrying out other computations on each element; and (b) a Reduce step that transforms the set of data elements produced by the Map step into a single data element.

A naive example would be to compute the sum of squares of a time series. A vector of input numbers, [3, 1, 2, 19, 3] could each be squared independently from one another by a Map task, resulting in the intermediate tuple vector [(1, 9), (1, 1), (1, 4), (1, 361), (1, 9)]. The keys are artificially the same for all tuples to ensure that each tuple gets processed by the same Reducer, as there is an implicit sort step, not shown in Figure 1, that happens after all Mappers have completed but before the Reducer starts. Then the Reduce step could carry out the summation, aggregating the data vector to the single scalar 384. Jobs submitted by the user to the Hadoop/MapReduce system get broken down to a set of tasks (i.e., there may be many parallel Mappers working on slices of the data: see Figure 1). The key innovation of HDFS, a redundant distributed file system modeled after Google's GFS [10], is to optimize storage and streaming access to large files on commodity hardware. This is achieved by means of *n*-time replication (e.g., *n*=3 means every file is held on three machines at any one time, and if one of them dies, another copy will be created to make up for the loss). HDFS blocks on disk are large (typically 64 MB). The command-line program hadoop can take UNIX-style commands as arguments to list, copy, remove, etc., files as well as to upload from the local file system to HDFS and back.

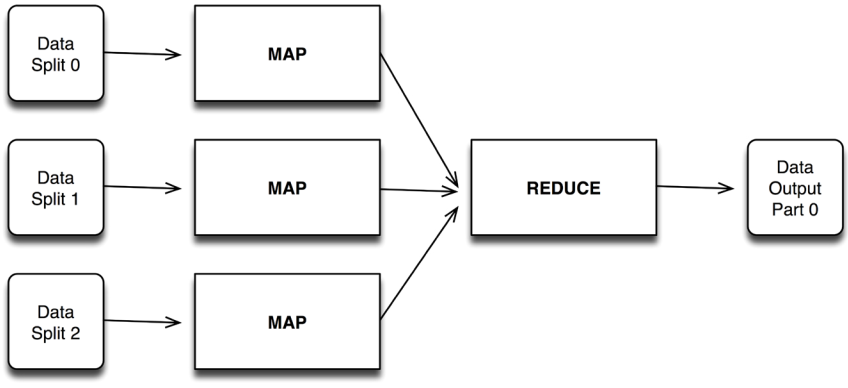

FIGURE 1: MAPREDUCE PROCESSING MODEL

Procurement

We describe how to build such a cluster next. We choose the GNU/Linux operating system because it is very efficient, scalable, stable, secure, is available in source code without licensing impediments, and has a large user base, which ensures rapid responses to support questions. We select the Ubuntu distribution of the operating system because it has good support and a convenient package management system and is offered in a server edition that contains only server essentials (Ubuntu Server). At the time of writing, release 9.07 was current. Little, if anything, of the described approach depends on this particular version. You will need $5,000–$8,000 for a 10-node installation and two person-days of your time. You will also need the following components:

- **Master**: We need a desktop PC running Linux as a master machine. In most cases, no master purchase will be necessary, because an existing high-end PC will be available. In one installation, we picked a Dell Optiplex (2x1 TB HDD configured as RAID, 24" TFT display), which was already available. Cost: $0 (if you don't have a main machine yet, I'd suggest ordering a Mac Pro or a Dell for $3,000).
- **Switch**: Using a professional-grade Gigabit switch (e.g., the auto-configuring Netgear ProSafe 24) means we can save valuable time for setup; it has 24 ports, giving us room for expansion. Gigabit Ethernet is important (and now affordable), as there is going to be a lot of traffic between nodes.
- **Network cabling**: Get one CAT6 patch cable per node to connect it with the switch. Cost is $18 + $3*n*, where *n* = number of nodes; when ordering in bulk, plan on $40 for a 10-node cluster.
- **Nodes**: We should plan for at least three slave-node PCs in order to have an advantage over a single, powerful desktop PC or server and to deploy the various core Hadoop processes. Since we are drawing from commodity components, we should pick an attractive package deal rather than waste time on customizing a node. Criteria are price, CPU speed, number of cores, RAM, number of hard disk slots, energy consumption, cooling, and noise. The hard disk drive size is not a criterion, because they are going to be replaced (cheap commodity PC deals include only very small disk drives). The amount of RAM is important, but since it can be cheaply replaced what matters more is the potential rather than the existing memory size: 2 GB should be considered the absolute minimum, 4 GB/node RAM is recommended (16 GB would be ideal, but at the time of writing is unlikely to be found in consumer machines). For the cluster shown in Figure 2, Acer X2 nodes with a dual-core Athlon X2 64-bit CPU (1.2 GHz), 3 GB RAM, 320 GB HDD were selected because they offer 4,800 bogo-MIPS/core for around $400. Cost is $400*n*; plan on $4,000 for 10 nodes.
- **Hard disk drives** (1 TB or higher): Determine the type of hard disk drive with the lowest $/TB cost. High-capacity drives (at the time of writing, 2 TB) are overly expensive, and the average size of drives used in commodity PCs as you buy them is too small, so the drives they come with (e.g., 320 GB) have to be replaced. Cost is $100n: plan on $1,000 for 10 nodes.
- **Enclosure** (shelf or rack): The nodes, switch, etc., need to live somewhere. A 42U 19" rack is the standard for data centers, however it may prove an unreasonable choice for several reasons: first, the cost of a new rack could easily exceed the total of the hardware expenses for the cluster itself, and, second, since the nodes are commodity machines as opposed to "professional" 19" servers, they may be hard to fix, so the main advantage of the 19" rack may be lost on them.
  Alternatives are cheap IKEA shelves made from wood or metal or anything similar. Finding a solution with wheels to keep the cluster mobile avoids

having to disassemble it, should the need for relocating it arise. Cost: approximately $400 (but if you are a system administrator, it is highly likely that you already have a spare rack or shelf somewhere).

- **Socket multiplier**: Use a fused socket multiplier to cope with the plugs of all nodes and the master. Cost: $20.

FIGURE 2. HYDRA, A MINIATURE CLUSTER WITH ONE MASTER PC AND THREE NODES, IS THE FIRST AUTHOR'S PRIVATE CLUSTER FOR HOME OFFICE USE. THIS IS ONE OF A SERIES OF INSTALLATIONS OF VARYING SIZES WE HAVE BEEN WORKING ON.

Physical Setup and Assembly

It is important to pick a suitable location for the cluster up front. Power consumption for a cluster of reasonable size will be considerable. For instance, 10 nodes of Acer X1700 at 220 watts each amount to 2200/120 = 18.4 amperes, which is just under the limit of what the circuit breaker of a typical household or small office will be able to carry. In addition, consider the significant heat generation that a large cluster can create.

Upgrade RAM and insert HDD drives. If you use a closed rack that comes with a door (recommended to keep the noise level down, but even more expensive than its open siblings), you may consider removing all cases of the node PCs altogether to improve ventilation (this "naked" configuration was pioneered by Google).

Operating System Installation

Download Ubuntu Server 9.07 or higher and burn a medium; connect all nodes with the switch using the CAT6 cables; connect all nodes with the power source. Connect the master PC's screen to the first node to be installed, insert the Ubuntu CD to boot, and install Ubuntu Server. Set up the HDD partitions as follows:

- **boot** (1 GB) not mounted, ext3, bootable, primary (important: this one should be at the beginning of the disk, or you may run into BIOS boot problems, e.g., "grub error 2")
- **root** (50 GB) mounted as /, ext3, logical

- **main** (0.94 TB) mounted as /var (var is a standard convention indicating variable data, i.e., a high amount of input/output is to be expected), ext3, logical
- **swap** (2 GB) not mounted, type swap, logical using manual partitioning (last option)
- Don't use logical volume management (LVM) or encryption.

Set the time zone to UTC/GMT and install software packages:

- Ubuntu server (always included, no action required)
- LAMP
- OpenSSH

You will also have to give the computer a name (e.g., the cluster in Figure 2 is called Hydra, so its nodes are called hydra1, hydra2, . . .). After the installation you should be able to re-boot and to issue:

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install sun-java6-jdk xorg gdm xfce4 xemacs21
```

Since our example cluster is experimental, we wanted to be able to run X11 on the nodes; every package from xorg onwards is not necessary for a production cluster. Depending on the tasks that we anticipate using the cluster for, we may want to consider installing additional packages. One of us does a lot of Web research, data crawling, analytics, and statistical machine learning, so it makes sense to get some crawlers, the R statistics system, and libraries for numeric computing:

```
sudo apt-get install wget curl lynx r-base r-base-dev python-numpy python-scipy
```

At this point, our first node is operational as far as the operating system is concerned, but in order to make combined used of its nodes as a single quasi-utility we still need to install Apache Hadoop and then replicate the setup to the other nodes.

Example Hadoop Installation on a Small Cluster

After the installation of the operating system, we can now turn to the setup of Apache Hadoop as our middleware for redundant storage and parallel processing. When we are done, we should have three processes running on our three nodes, as follows:

```
hydra1  Acer X1700  Master      NameNode;DataNode;JobTracker
hydra2  Acer X1700  Slave       SecondaryNameNode;DataNode;TaskTracker
hydra3  Acer X1700  Slave       DataNode;TaskTracker
```

Note that the use of the word Master here pertains to Hadoop and is distinct from the cluster master PC (which has the keyboard and screen attached).

1. Ensure that Java is set up. The latest Hadoop releases depend on Java 6.x or later. Download/install/test the latest 6.x Java release if this is not already set up.

It is recommended to follow the procedures for single node cluster setup as described in the online article by Michael G. Noll on running Hadoop in Ubuntu Linux environments [1].

We suggest you do this on each machine of a multi-node cluster to help verify the operational status of Hadoop on each node before continuing to set up a multi-node configuration. The steps below show examples of following these instructions.

2. Change to user hadoop. Add a hadoop group and hadoop user for that group.

```
<your-user-name>@hydra1:~$ sudo addgroup hadoop
<your-user-name>@hydra1:~$ sudo adduser --ingroup hadoop hadoop
```

3. Add the following exports for proxy and JAVA_HOME to the .bashrc file for both your user and the new hadoop user:

```
export http_proxy=<yourProxy>:<proxyPort>
export JAVA_HOME=<yourJavaHomePath>
```

4. Configure and test SSH operation on all nodes (required by Hadoop).

As the hadoop user, on the master node, create the RSA key:

```
hadoop@hydra1:~$ ssh-keygen -t rsa -P ""
```

Copy or append the new key to the authorized_keys file in the .ssh directory:

```
hadoop@hydra1:~$ cat /home/hadoop/.ssh/id_rsa.pub >> ~hadoop/.ssh/
    authorized_keys
```

Try to connect to the local machine with the hadoop user. Respond with a yes when prompted to "continue connecting."

```
hadoop@hydra1:~$ ssh localhost
```

5. Download Hadoop 0.19.2

As of this writing, there are known instability issues with the 0.20 release, so release 0.19.2 is used for this installation. For example, download from:

```
http://newverhost.com/pub/hadoop/core/hadoop-0.19.2/hadoop-0.19.2.tar.gz
```

(Note: there are also API differences between 0.19 and 0.20+.) Now, with administrator permissions, install this release in the desired directory location. The example installation steps below assume the original download was to the directory location: /home/<your-user-name>/Desktop. Note that *these steps must be performed as root*.

Uncompress the Hadoop release to the desired location:

```
root@hydra1:/usr/local#  tar xzf /home/<your-user-name>/Desktop/
hadoop-0.19.2.tar.gz
```

Rename the release as desired, and change ownership of all the release contents to permit use by the hadoop user:

```
root@hydra1:/usr/local# mkdir /var/hadoop
root@hydra1:/usr/local# ln –s /var/hadoop hadoop
root@hydra1:/usr/local# mv hadoop-0.19.2/* hadoop/*
root@hydra1:/usr/local# chown -R hadoop:hadoop hadoop
```

6. Add an export for HADOOP_HOME to the .bashrc file for both your user and the hadoop user:

```
export HADOOP_HOME=<yourHadoopHomePath>
```

7. Edit the file /usr/local/hadoop/conf/hadoop-env.sh by uncommenting the export for JAVA_HOME and setting it to the correct value:

```
export JAVA_HOME=<yourJavaHomePath>
```

8. Edit the file /usr/local/hadoop/conf/hadoop-site.xml to contain single node test configuration settings. Adjust values to suit your own configuration needs. There are *many* possible default configuration parameter settings

that can be adjusted. See the /usr/local/hadoop/conf/hadoop-defaults.xml file for more information about the complete set of adjustable parameters.

```
<configuration>

<property>
  <name>hadoop.tmp.dir</name>
  <value>/usr/local/hadoop/tmp/datastore/hadoop-${user.name}</value>
  <description>
  A base location for other temp datastore directories.
  </description>
</property>

<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:54310</value>
  <description>
  The name of the default file system.
  A URI whose scheme and authority determine the filesystem implementation.
  The URI's scheme determines the config property (fs.SCHEME.impl) naming
  the filesystem implementation class. The URI's authority is used to
  determine the host, port, etc. for a file system.
  </description>
</property>

<property>
  <name>mapred.job.tracker</name>
  <value>localhost:54311</value>
  <description>
  The host and port that the MapReduce job tracker runs at.
  If "local," then jobs are run in-process as a single map and reduce task.
  </description>
</property>

<property>
  <name>dfs.replication</name>
  <value>1</value>
  <description>
  Default block replication.
  The actual number of replications can be specified when the file is created.
  The default is used if replication is not specified at create time.
  </description>
</property>

</configuration>
```

Note that dfs.replication specifies the number of copies of each file that is kept on the cluster by HDFS's redundancy mechanism. For a single "pseudo-cluster" setup, we set this to 1 until that node is operational, then we must change it (e.g., back to its default of 3).

9. With administrator privileges, create the hadoop tmp/datastore directory for your user and the hadoop user and change ownership to allow use by the hadoop and your user:

```
root@hydra1:/usr/local/hadoop# mkdir –p tmp/datastore/hadoop-hadoop
root@hydra1:/usr/local/hadoop# chown -R hadoop:hadoop tmp/datastore/
    hadoop-hadoop
root@hydra1:/usr/local/hadoop#mkdirtmp/datastore/hadoop-<your-user-name>
root@hydra1:/usr/local/hadoop# chown -R <your-user-name>:<your-user-
    name>   tmp/datastore/hadoop-<your-user-name>
```

10. Test Hadoop execution in single-node mode, using HDFS.

As the hadoop user, format the NameNode:

```
hadoop@hydra1:~$ $HADOOP_HOME/bin/hadoop namenode -format
```

Start the (single-node) cluster:

```
hadoop@hydra1:~$ $HADOOP_HOME/bin/start-all.sh
```

Verify that the expected Hadoop processes are running using Java's jps:

```
hadoop@hydra1:~$ jps
27069 JobTracker                26641 NameNode
26729 DataNode                  29425 Jps
26923 SecondaryNameNode         27259 TaskTracker
```

With administrator permission, use netstat to verify that Hadoop is listening on the expected/configured ports. For example:

```
root@hydra1:~# netstat -plten | grep java | grep 127.0.0.1
tcp6  0  0 127.0.0.1:54310  :::*          LISTEN  1001   590635 26641/java
tcp6  0  0 127.0.0.1:54311  :::*          LISTEN  1001   594563 27069/java
tcp6  0  0 127.0.0.1:51633  :::*          LISTEN  1001   601104 27259/java
```

11. Set up and run a Hadoop example test application to verify operability.

Adjust the <HadoopHome>/conf/hadoop-env.sh file to set JAVA_HOME, HADOOP_HOME, and a reasonable CLASSPATH (if desired) for Hadoop executions:

```
# == file "hadoop-env.sh" ==
export JAVA_HOME=/usr/lib/jvm/java-6-sun/jre
export HADOOP_HOME=/usr/local/hadoop
export CLASSPATH=$HADOOP_HOME/hadoop-0.19.2-core.jar:$HADOOP_
    HOME/hadoop-0.19.2-examples.jar:$HADOOP_HOME/hadoop-0.19.2-test.
    jar:$HADOOP_HOME/hadoop-0.19.2-tools.jar:$CLASSPATH:$classpath
```

Run the test example program. The following example executes the pi program included in the distributed Hadoop examples. Use the source command to ensure that the settings are kept by the executing shell process.

```
hadoop@hydra1:~$ source hadoop-env.sh
hadoop@hydra1:~$ $HADOOP_HOME/bin/hadoop jar          $HADOOP
    _HOME/hadoop-0.19.2-examples.jar pi 2 10
```

The output should look similar to the following:

```
Number of Maps = 2 Samples per Map = 10
Wrote input for Map #0
Wrote input for Map #1
Starting Job
09/10/22 13:17:50 INFO mapred.FileInputFormat: Total input paths to process : 2
09/10/22 13:17:50 INFO mapred.JobClient:  Running job:
    job_200910221225_0001
09/10/22 13:17:51 INFO mapred.JobClient:    map 0% reduce 0%
09/10/22 13:18:00 INFO mapred.JobClient:    map 50% reduce 0%
09/10/22 13:18:03 INFO mapred.JobClient:    map 100% reduce 0%
09/10/22 13:18:10 INFO mapred.JobClient:    map 100% reduce 100%
09/10/22 13:18:11 INFO mapred.JobClient:  Job complete:
    job_200910221225_0001
09/10/22 13:18:11 INFO mapred.JobClient:  Counters: 16
09/10/22 13:18:11 INFO mapred.JobClient:    File Systems
09/10/22 13:18:11 INFO mapred.JobClient:      HDFS bytes read=236
```

```
09/10/22 13:18:11 INFO mapred.JobClient:     HDFS bytes written=212
09/10/22 13:18:11 INFO mapred.JobClient:     Local bytes read=78
09/10/22 13:18:11 INFO mapred.JobClient:     Local bytes written=218
09/10/22 13:18:11 INFO mapred.JobClient:   Job Counters
09/10/22 13:18:11 INFO mapred.JobClient:     Launched reduce tasks=1
09/10/22 13:18:11 INFO mapred.JobClient:     Launched map tasks=2
09/10/22 13:18:11 INFO mapred.JobClient:     Data-local map tasks=2
09/10/22 13:18:11 INFO mapred.JobClient   Map-Reduce Framework
09/10/22 13:18:11 INFO mapred.JobClient:     Reduce input groups=2
09/10/22 13:18:11 INFO mapred.JobClient:     Combine output records=0
09/10/22 13:18:11 INFO mapred.JobClient:     Map input records=2
09/10/22 13:18:11 INFO mapred.JobClient:     Reduce output records=0
09/10/22 13:18:11 INFO mapred.JobClient:     Map output bytes=64
09/10/22 13:18:11 INFO mapred.JobClient:     Map input bytes=48
09/10/22 13:18:11 INFO mapred.JobClient:     Combine input records=0
09/10/22 13:18:11 INFO mapred.JobClient:     Map output records=4
09/10/22 13:18:11 INFO mapred.JobClient:     Reduce input records=4
Job Finished in 21.342 seconds
Estimated value of PI is 3.2
```

Congratulations! At this point you have a simple, single-node Hadoop environment up and running!

12. Shut down the Hadoop processes in the single-node cluster:

```
hadoop@hydra1:~$ $HADOOP_HOME/bin/stop-all.sh
```

The output should look similar to the following:

```
stopping jobtracker
localhost: stopping tasktracker
stopping namenode
localhost: stopping datanode
localhost: stopping secondarynamenode
```

To configure the Hadoop middleware to handle a *multi*-node cluster, we recommend you follow the procedures for setting up multi-node clusters described in an online article by Michael G. Noll [2].

You now face the issue of having to install the whole node's environment (Linux, packages, Hadoop) from one node to the remaining nodes in a near-identical way. For smaller clusters this can be done manually. For larger clusters with nodes that possibly have different hardware specifications, stronger tools need to be used to define machine classes, separate configurations for each class, and assist in the distribution of these configurations to the appropriate node machines. In these settings, various sources suggest the use of configuration management tools like Puppet [4], Cfengine [5], or Bcfg2 [6]. More concretely, there are several solutions to this, depending on your experience and number of nodes:

1. Burn an ISO image with your setup and use this with the remaining nodes.
2. Insert the empty hard disk drives as secondary drives in the master PC temporarily in order to copy over the entire disk using the dd(1) command.
3. Clone the disk over a network connection using dd(1) and netcat (nc(1)) as outlined by [7].
4. Install the other nodes manually (estimated time: about 30 min/node).

Method 3 is superior for large clusters, but method 4 is fast enough for smaller clusters. Remember that the hostname must be unique, so you may

have to set it manually after cloning the node setups by manually invoking the hostname(1) command for each node.

In order to automate the installation completely, [9] recommends using static IP addresses for the nodes, setting the hostname by keeping a file hostnames.new that contains the node names and their static IP addresses, and then generating a set of node-specific kick-start files from a master template (here called "anaconda-ks.cfg," with NODE_HOSTNAME and NODE_STATIC_IP being placeholders) as follows:

```
for i in $(cat ~/hostnames.new) ; do \
  cat anaconda-ks.cfg | sed s/NODE_HOSTNAME/$i/g | sed s/NODE_STATIC_
    IP/$(grep $i /etc/hosts | awk '{print $1}')/g > ks-$i.cfg ; \
done
```

In this approach, the operating system is booted over the network using the node-specific kick-start file.

There is yet another mode of operation to install Hadoop on more than one node very conveniently: Cloudera Inc., a cluster/cloud computing startup, which recently hired Hadoop architect Doug Cutting, permits you to enter your desired cluster configuration on a Web interface (my.cloudera.com), which automatically creates customized installers (e.g., *.rpm packages) that contain all the cluster configuration information.

Operating the Cluster

Now that your Hadoop cluster is fully operational, we recommend you try out the word count example from the Apache Hadoop tutorial [8], which shows you how the UNIX wc(1) command can be distributed across a cluster.

In general, to use Hadoop, you can choose several modes of operation:

1. Use its native Java API. To do this, you will have to write mapper and reducer classes that extend org.apache.hadoop.mapred.MapReduceBase and implement Mapper<S> and Reducer<T>, respectively (S and T are type signatures).

2. Use Hadoop Streaming. If you already have a set of command-line tools such as taggers, parsers, or classifiers that you would like to utilize, you can invoke them as mappers and reducers, or you can write mappers and reducers in Python, Perl, etc. This is an excellent way to prototype a new system pipeline, but anecdotal evidence suggests the startup cost of scripting language interpreters/compilers may be prohibitive for production use (recoding a Python/Perl program in C++ for use in Streaming or using the native Java API may be faster by a large factor).

3. Use Hadoop's C++ interface. There is a C++ wrapper library, Hadoop Pipes, which uses socket communication to talk to the Hadoop daemon processes directly (avoiding JNI).

Besides the MapReduce parallel processing functionality and the HDFS distributed redundant file system, there are some other useful sub-projects in Apache Hadoop: **Avro** is a system for serialization of data. It is schema-based and uses a fast, compact binary format. **Chukwa** is a sub-system for managing and monitoring the collection and updating of distributed log files. **HBase** is a free open-source database for Hadoop modeled after Google's Bigtable. Facebook's contribution is **Hive**, which comprises a toolkit and library for processing text and logfiles and which contains an interactive environment, a query compiler and evaluation engine for HQL, an SQL-like query language, compiler, driver, and execution engine, as well as a

metastore called SerDe (short for Serialization and Deserialization). **Pig** is a versatile scripting language contributed by Yahoo that permits easy iteration over data records/tuples, sorting, joins, and counting, besides user-defined functions. Users with exposure to SQL will quickly pick up Pig idioms, and a nice property of the interpreter, which is implemented in Java, is that it can execute scripts both inside and outside the Hadoop/HDFS infrastructure. **ZooKeeper** is a centralized service for a couple of things that make distributed computing challenging, namely, maintaining configuration information, naming services, providing distributed synchronization, and providing a notion of groups. For example, the distributed synchronization primitives offered permit the implementation of a distributed queue. The Pig distribution contains some tutorial examples on mining Web search engine queries that we highly recommend; running these will give you an idea of the power of the MapReduce paradigm and its free Hadoop implementation.

Next, you can think of applications in your own areas of interest and express them in terms of mappers and reducers to execute them on your cluster. We are particularly interested in machine learning, information retrieval (indexing, retrieval, clustering), graph analysis of social networks, and data mining from log files. Check out trendingtopics.org for an example data mining application in the area of automatic trend analysis, which was built with Hadoop. Whether you want to sort petabytes of customer records in "record" time (a Hadoop cluster currently holds the record for a sorting benchmark) or crunch logfiles to find hidden statistical relationships in your data, Hadoop is your friend (and you are not alone, as Yahoo, Facebook, Twitter, etc., are heavily relying on it as well). If you seek further inspiration, we recommend the book *Beautiful Data* [13].

Summary and Conclusion

We have described a successfully completed project to build a cluster computing utility from commodity parts. The cluster is affordable (<$5,000), can be built incrementally, and is more powerful than servers that were priced over a quarter million dollars just a few years ago. Hadoop provides powerful OS middleware for large-scale batch processing such as the automatic analysis of large document collections. We expect that in the future, enterprise versions of commodity operating systems will incorporate some of these capabilities, but we hope the above introduction can serve to give the interested reader a head start (for more detailed, recipe-style instructions targeting a non-system administrator audience, also consult [12]).

Happy Hadooping!

REFERENCES

[1] Michael G. Noll, "Running Hadoop on Ubuntu Linux (Single-Node Cluster)": http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Single-Node_Cluster).

[2] Michael G. Noll, "Running Hadoop on Ubuntu Linux (Multi-Node Cluster)": http://www.michael-noll.com/wiki/Running_Hadoop_On_Ubuntu_Linux_(Multi-Node_Cluster).

[3] Tom White, *Hadoop: The Definitive Guide* (O'Reilly/Yahoo! Press, 2009).

[4] Puppet online information: http://reductivelabs.com/trac/puppet/wiki/DocumentationStart.

[5] Cfengine online information: http://www.cfengine.org/manuals/cf3-reference.html.

[6] Bcfg2 online information: http://trac.mcs.anl.gov/projects/bcfg2/wiki/UsingBcfg2.

[7] Gite Vivek, "Copy hard disk or partition image to another system using a network and netcat (nc)": http://www.cyberciti.biz/tips/howto-copy-compressed-drive-image-over-network.html.

[8] Apache Hadoop Map/Reduce Tutorial online information: http://hadoop.apache.org/common/docs/current/mapred_tutorial.html.

[9] Installing CentOS on a cluster via NFS online information: http://biowiki.org/InstallingCentOSOnClusterViaNFS.

[10] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung, "The Google File System," *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003),* Bolton Landing, NY: 29-43.

[11] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI '04)*: 137–150.

[12] Jochen L. Leidner and Gary Berosik (2009), "Building and Installing a Hadoop/MapReduce Cluster from Commodity Components Technical Report": http://arxiv.org/ftp/arxiv/papers/0911/0911.5438.pdf.

[13] Toby Segaran and Jeff Hammermacher, *Beautiful Data: The Stories Behind Elegant Data Solutions* (O'Reilly, 2009).

Disclaimer. All opinions expressed in this article are the authors' and do not reflect any official opinion or endorsement by the Thomson Reuters Corporation.