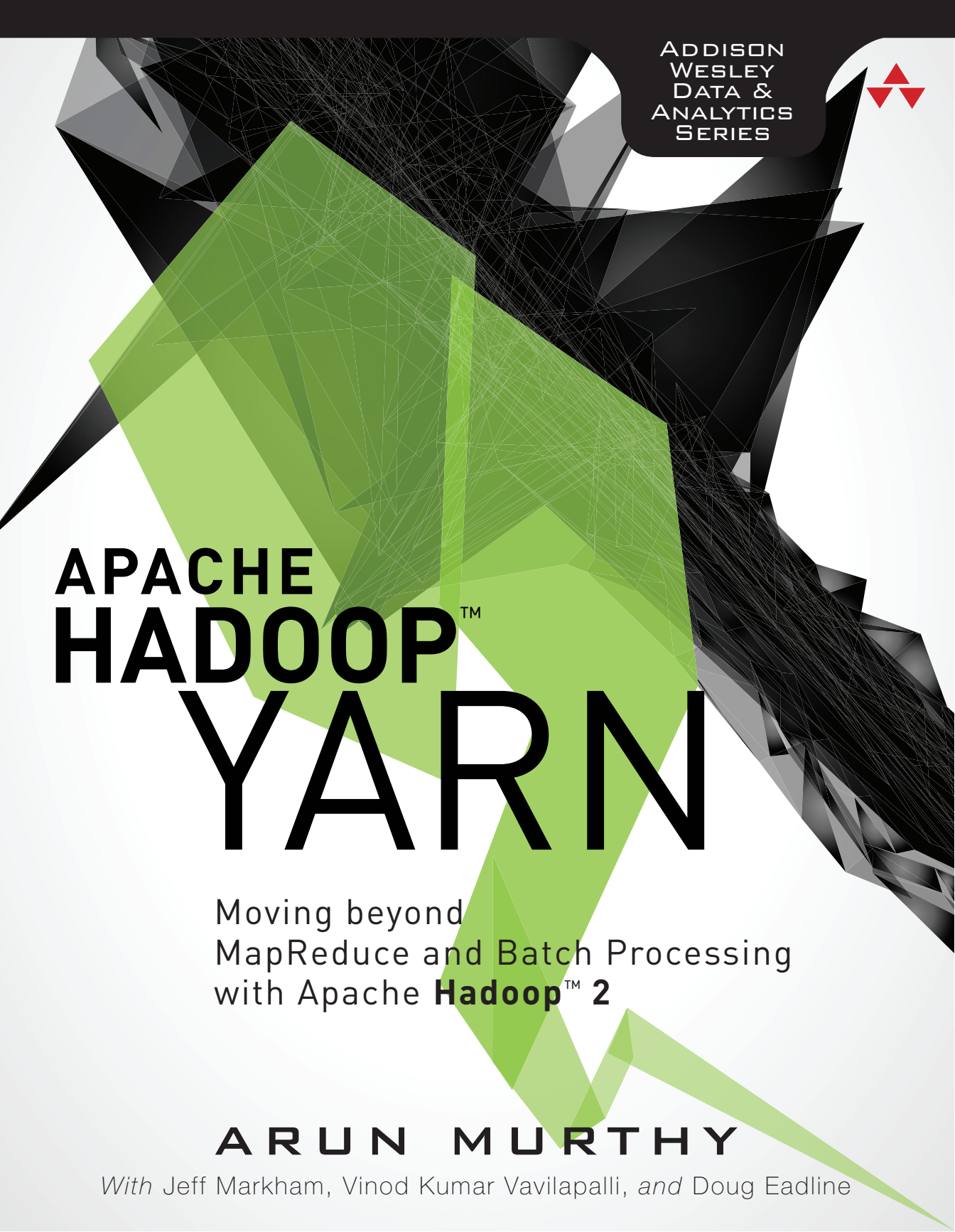
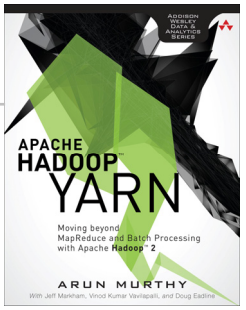

Addison
Wesley
Data &
Analytics
Series
APACHE
HADOOP™
YARN
Moving beyond
MapReduce and Batch Processing
with Apache Hadoop™ 2
ARUN MURTHY
With Jeff Markham, Vinod Kumar Vavilapalli, and Doug Eadline



Apache Hadoop YARN will be published in the winter of 2014, with continually updated drafts available on Safari Books Online (www.safaribooksonline.com).

Draft Manuscript

This manuscript has been provided by Pearson Education and Hortonworks at this early stage to create awareness for the upcoming publication. **It has not been fully copyedited or proofread**; we trust that you will judge this book on technical merit, not on grammatical and punctuation errors that will be corrected prior to publication.

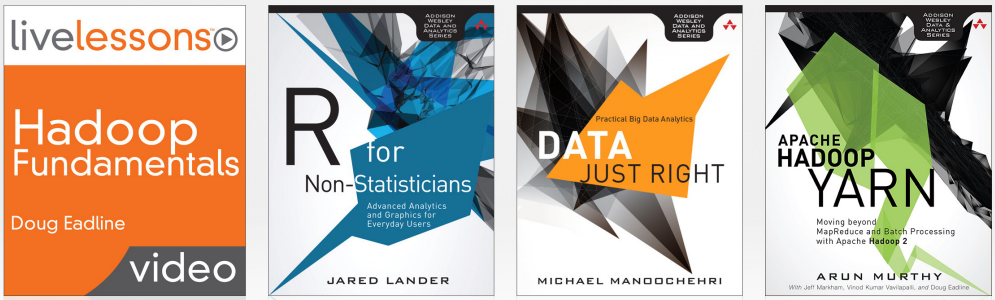
Learn how to implement and use YARN, the new generation of Apache Hadoop that empowers applications of all types to move beyond batch and implement new distributed applications IN Hadoop!

This authoritative guide is the best source of information for getting started with, and then mastering, the latest advancements in Apache Hadoop. As you learn how to structure your applications in Apache Hadoop 2, it provides you with an understanding of the architecture of YARN (code name for Hadoop 2) and its major components. In addition to multiple examples and valuable case studies, a key topic in the book is running existing Hadoop 1 applications on YARN and the MapReduce 2 infrastructure.

Data processing in Apache Hadoop has undergone a complete overhaul, emerging as Apache Hadoop YARN. This generic compute fabric provides resource management at datacenter scale and a simple method by which to implement distributed applications (MapReduce and a multitude of others) to process petabytes of data on Apache Hadoop HDFS. YARN significantly changes the game, recasting Apache Hadoop as a much more powerful system by moving it beyond MapReduce into additional frameworks. Two of the primary authors of the YARN project, Arun C. Murthy, the Founder of the YARN project, and Vinod K. Vavilapalli, the YARN Project Lead, take you through the key design concepts of YARN itself. They also provide you a tour of how new applications can be written in an elegant and simple manner to get more out of Hadoop clusters as Hadoop is no longer a one-trick pony. Learn how existing MapReduce applications can be seamlessly migrated to YARN in a hassle-free manner and how other existing components in Apache Hadoop ecosystem such as Apache Hive, Apache Pig & Apache HBase improve thanks to YARN.

Apache Hadoop™ YARN

The Addison-Wesley Data & Analytics Series



Visit **informit.com/awdataseries** for a complete list of available publications.

The **Addison-Wesley Data & Analytics Series** provides readers with practical knowledge for solving problems and answering questions with data. Titles in the series will tackle three primary areas of focus:

1) **Infrastructure:** how to store, move, and manage data
2) **Algorithms:** how to mine intelligence or make predictions based on data
3) **Visualizations:** how to represent data and insights in a meaningful and compelling way

The series aims to tie all three of these areas together to help the reader build end-to-end systems for fighting spam, making recommendations, building personalization, detecting trends, patterns, or problems and gaining insight from the data exhaust of systems and user interactions.

Make sure to connect with us!
informit.com/socialconnect

informIT.com the trusted technology learning source | Addison-Wesley | Safari Books Online

Apache Hadoop™ YARN

Moving Beyond MapReduce and Batch Processing with Apache Hadoop 2

Arun Murthy

with
Jeffrey Markham
Vinod Vavilapalli
Doug Eadline

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of the early draft of this manuscript, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

Upon publication the publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
1-800-382-3419
corpsales@pearsontechgroup.com

For sales outside of the U.S., please contact™

International Sales
international@pearsoned.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Executive Editor
Debra Williams Cauley

Senior Development Editor
Chris Zahn

Managing Editor
John Fuller

Publishing Coordinator
Kim Boedigheimer

Book Designer
Chuti Prasertsith

Contents at a Glance

About the Authors

Arun Murthy has contributed to Apache Hadoop full-time since the inception of the project in early 2006. He is a long-term Hadoop Committer and a member of the Apache Hadoop Project Management Committee. Previously, he was the architect and lead of the Yahoo Hadoop MapReduce development team and was ultimately responsible, technically, for providing Hadoop MapReduce as a service for all of Yahoo—currently running on nearly 50,000 machines! Arun is the Founder and Architect of the Hortonworks Inc., a software company that is helping to accelerate the development and adoption of Apache Hadoop. Hortonworks was formed by the key architects and core Hadoop committers from the Yahoo! Hadoop software engineering team in June 2011. Funded by Yahoo! and Benchmark Capital, one of the preeminent technology investors, their goal is to ensure that Apache Hadoop becomes the standard platform for storing, processing, managing and analyzing big data. He lives in Silicon Valley.

Jeff Markham is a Solution Engineer at Hortonworks Inc., the company promoting open source Hadoop. Previously, he was with VMware, Red Hat, and IBM helping companies build distributed applications with distributed data. He's written articles on Java application development and has spoken at several conferences and to Hadoop User Groups. Jeff is a contributor to Apache Pig and Apache HDFS.

Vinod Kumar Vavilapalli has been contributing to Apache Hadoop project full-time since mid-2007. At Apache Software Foundation, he is a long term Hadoop contributor, Hadoop committer, member of the Apache Hadoop Project Management Committee and a Foundation Member. Vinod is a MapReduce and YARN go-to guy at Hortonworks Inc. For more than five years he has been working on Hadoop and still has fun doing it. He was involved in HadoopOnDemand, Hadoop-0.20, CapacityScheduler, Hadoop security, MapReduce and now is a lead developer and the project lead for Apache Hadoop YARN. Before Hortonworks, he was at Yahoo! working in the Grid team that made Hadoop what it is today, running at large scale—up to tens of thousands of nodes. Vinod loves reading books, of all kinds, and is passionate about using computers to change the world for better, bit by bit. He has a Bachelors degree from the Indian Institute of Technology Roorkee in Computer Science and Engineering. He lives in Silicon Valley and is reachable at twitter handle @tshooter.

Douglas Eadline, PhD, began his career as a practitioner and a chronicler of the Linux Cluster HPC revolution and now documents big data analytics. Starting with the first Beowulf How To document, Dr. Eadline has written hundreds of articles, white papers, and instructional documents covering virtually all aspects of HPC computing. Prior to starting and editing the popular ClusterMonkey.net web site in 2005, he served as Editor-in-chief for *ClusterWorld Magazine*, and was Senior HPC Editor for *Linux Magazine*. Currently, he is a consultant to the HPC industry and writes a monthly column in *HPC Admin Magazine*. Both clients and readers have recognized Dr. Eadline's ability to present a "technological value proposition" in a clear and accurate style. He has practical hands on experience in many aspects of HPC including, hardware and software design, benchmarking, storage, GPU, cloud, and parallel computing. He is the author of *Hadoop Fundamentals LiveLessons* video from Pearson.

1

YARN Quick Start

A production Apache Hadoop system can take time to set up properly and is not necessary to start experimenting with many of the YARN concepts and attributes. This chapter provides a quick start guide to installing Hadoop version 2.0.4 on a single machine (workstation, server, or hefty laptop).

A more complete description of other installation options such as those required by a production cluster setup is given in Chapter 4, "Installing YARN." Before we begin with the quick start though, there are a few background skills that will help with installation. These skills include rudimentary knowledge of Linux, package installation, and basic system administration commands.

A basic Apache Hadoop YARN system has two core components:

- The Hadoop Distributed File System for storing data, which will be referred to as HDFS.
- Hadoop YARN for implementing applications to process data.

There are other Apache Hadoop components, such as Pig or Hive, that can be added after the two core components are installed and operating properly.

Steps to Configure a Single Node YARN Server

The following type of installation is often referred to as "pseudo distributed" because it mimics some of the functionality of a distributed Hadoop cluster. A single machine is, of course, not practical for any production use, nor is it parallel. A small scale Hadoop installation can provide a simple method for learning Hadoop basics, however.

The recommended *minimal* installation hardware is a dual-core processor with 2 GBs of RAM and 2 GBs of available hard drive space. The system will need a recent Linux distribution with Java installed (Red Hat Enterprise Linux or rebuilds, Fedora, Suse Linux Enterprise, or OpenSuse). Red Hat Enterprise Linux 6.3 is used for this installation example. A bash shell environment is also assumed. The first step is to download Apache Hadoop.

Step 1: Download Apache Hadoop

Download the latest distribution from the Hadoop web site (http://hadoop.apache.org/). For example, as root do the following:

```
# cd /root
# wget http://mirrors.ibiblio.org/apache/hadoop/common/hadoop-2.0.4-alpha/hadoop-2.0.4-alpha.tar.gz
```

Next create and extract the package in /opt/yarn:

```
# mkdir /opt/yarn
# cd /opt/yarn
# tar xvzf /root/hadoop-2.0.4-alpha.tar.gz
```

If the archive was extracted correctly, the following directory structure should be under /opt/yarn/hadoop-2.0.4-alpha. (Note that, depending on the source distribution, your version may be different.)

```
etc/
+ hadoop
include/
lib/
+ native
libexec/
sbin/
share/
+ doc
   + . . .
+ hadoop
   + . . .
```

The rest of these steps will create a basic single machine YARN installation.

Step 2: Set JAVA_HOME

For Hadoop 2, the recommended version of Java can be found at http://wiki.apache.org/hadoop/HadoopJavaVersions. As mentioned, Red Hat Enterprise Linux 6.3 is the base installation which includes Open Java 1.6.0_24. Make sure the java-1.6.0-openjdk RPM is installed. In order to include JAVA_HOME for all bash users (others shells must be set in a similar fashion) make an entry in /etc/profile.d:

```
# echo "export JAVA_HOME=/usr/lib/jvm/java-1.6.0-openjdk-1.6.0.0.x86_64/" > /etc/profile.d/java.sh
```

To make sure JAVA_HOME is defined for this session, source the new script:

```
# source /etc/profile.d/java.sh
```

Other Linux distributions may differ, and the steps that follow will need to be adjusted.

Step 3: Create Users and Groups

It is best to run the various daemons with separate accounts. Three accounts (yarn, hdfs, mapred) in group hadoop can be created as follows:

```
# groupadd hadoop
# useradd -g hadoop yarn
# useradd -g hadoop hdfs
# useradd -g hadoop mapred
```

Step 4: Make Data and Log Directories

Hadoop needs various data and log directories with various permissions. Enter the following to create these directories:

```
# mkdir -p /var/data/hadoop/hdfs/nn
# mkdir -p /var/data/hadoop/hdfs/snn
# mkdir -p /var/data/hadoop/hdfs/dn
# chown hdfs:hadoop /var/data/hadoop/hdfs -R
# chown yarn:hadoop /var/log/hadoop/yarn -R
```

Next, move to the YARN installation root and create the log directory and set the owner and group as follows:

```
# cd /opt/yarn/hadoop-2.0.4-alpha
# mkdir logs# chmod g+w logs# chown yarn:hadoop . -R
```

Step 5: Configure core-site.xml

From the base of the Hadoop installation path (e.g., /opt/yarn/hadoop-2.0.4-alpha/), edit the etc/hadoop/core-site.xml file. The original installed file will have no entrees other than the `<configuration> </configuration>` tags. There are two properties that need to be set. The first is the fs.default.name property that sets the host and request port name for the NameNode (Metadata server for HDFS). The second is hadoop.http.staticuser.user, which will set the default user name to hdfs. Copy the following lines to the Hadoop etc/hadoop/core-site.xml file and remove the original empty `<configuration> </configuration>` tags.

```
<configuration>
	<property>
		<name>fs.default.name</name>
		<value>hdfs://localhost:9000</value>
	</property>
	<property>
		<name>hadoop.http.staticuser.user</name>
		<value>hdfs</value>
	</property>
</configuration>
```

Step 6: Configure hdfs-site.xml

From the base of the Hadoop installation path, edit the etc/hadoop/hdfs-site.xml file. In the single node pseudo distributed mode, we don't need or want the HDFS to replicate file blocks. By default, HDFS keeps three copies of each file in the filesystem. There is no need for replication on a single machine, thus the dfs.replication value will be set to one.

In hdfs-site.xml, we specify the NameNode, Secondary NameNode, and DataNode data directories that we created in Step 4. These are the directories used by the various components of HDFS to store data. Copy the following into Hadoop etc/hadoop/hdfs-site.xml and remove the original empty `<configuration> </configuration>` tags.

```
<configuration>
 <property>
   <name>dfs.replication</name>
   <value>1</value>
 </property>
 <property>
   <name>dfs.namenode.name.dir</name>
   <value>file:/var/data/hadoop/hdfs/nn</value>
 </property>
 <property>
   <name>fs.checkpoint.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>fs.checkpoint.edits.dir</name>
   <value>file:/var/data/hadoop/hdfs/snn</value>
 </property>
 <property>
   <name>dfs.datanode.data.dir</name>
   <value>file:/var/data/hadoop/hdfs/dn</value>
 </property>
</configuration>
```

Step 7: Configure mapred-site.xml

From the base of the Hadoop installation, edit the etc/hadoop/mapred-site.xml file. A new configuration option for Hadoop 2 is the capability to specify a framework name for MapReduce, setting the mapreduce.framework.name property. In this install we will use the value of "yarn" to tell MapReduce that it will run as a YARN application. First, copy the template file to the mapred-site.xml.

```
# cp mapred-site.xml.template mapred-site.xml
```

Next, copy the following into Hadoop etc/hadoop/mapred-site.xml file and remove the original empty `<configuration> </configuration>` tags.

```
<configuration>
```

```
<property>
   <name>mapreduce.framework.name</name>
   <value>yarn</value>
 </property>
</configuration>
```

Step 8: Configure yarn-site.xml

From the base of the Hadoop installation, edit the etc/hadoop/yarn-site.xml file. The yarn.nodemanager.aux-services property tells NodeManagers that there will be an auxiliary service called mapreduce.shuffle that it needs to implement. After we tell the NodeManagers to implement that service, we give it a class name as the means to implement that service. In this case, it's the `yarn.nodemanager.aux-services.mapreduce.shuffle.class`. Specifically, what this particular configuration does is tell MapReduce how to do its shuffle. Because NodeManagers won't shuffle data for a non-MapReduce job by default, we need to configure such a service for MapReduce. Copy the following to the Hadoop etc/hadoop/yarn-site.xml file and remove the original empty `<configuration> </configuration>` tags.

```
<configuration>
<property>
   <name>yarn.nodemanager.aux-services</name>
   <value>mapreduce.shuffle</value>
 </property>
 <property>
   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
 </property>
</configuration>
```

Step 9: Modify Java Heap Sizes

The Hadoop installation has various environment variables that determine the heap sizes for each Hadoop process. These are defined in the etc/hadoop/*-env.sh files used by Hadoop. The default for most of the processes is a 1GB heap size, but since we're running on a workstation that will probably have limited resources compared to a standard server, we need to adjust the heap size settings. The values that follow are what are adequate for a small workstation or server. They can be adjusted to fit your machine.

Edit etc/hadoop/hadoop-env.sh file to reflect the following (Don't forget to remove the "#" at the beginning of the line.):

```
HADOOP_HEAPSIZE=500
HADOOP_NAMENODE_INIT_HEAPSIZE="500"
```

Next, edit the mapred-env.sh to reflect the following:

```
HADOOP_JOB_HISTORYSERVER_HEAPSIZE=250
```

Finally, edit yarn-env.sh to reflect the following:

```
JAVA_HEAP_MAX=-Xmx500m
```

The following will need to added to yarn-env.sh

```
YARN_HEAPSIZE=500
```

Step 10: Format HDFS

In order for the HDFS NameNode to start, it needs to initialize the directory where it will hold its data. The NameNode service tracks all the meta-data for the filesystem. The format process will use the value assigned to dfs.namenode.name.dir in etc/hadoop/hdfs-site.xml earlier (i.e., /var/data/hadoop/hdfs/nn). Formatting destroys everything in the directory and sets up a new file system. Format the NameNode directory as the HDFS superuser, which is typically the 'hdfs' user account.

From the base of the Hadoop distribution, change directories to the 'bin' directory and execute the following commands.

```
# su - hdfs
$ cd /opt/yarn/hadoop-2.0.4-alpha/bin
$./hdfs namenode -format
```

If the command worked, you should see the following near the end of a long list of messages:

```
INFO common.Storage: Storage directory /var/data/hadoop/hdfs/nn has been successfully
formatted.
```

Step 11: Start the HDFS Services

Once formatting is successful, the HDFS services must be started. There is one for the NameNode (metadata server), a single DataNode (where the actual data is stored), and the SecondaryNameNode (checkpoint data for the NameNode). The Hadoop distribution includes scripts that set up these commands as well naming various other values like PID directories, log directories, and other standard process configurations. From the sbin directory from Step 10 execute the following as user hdfs:

```
$ cd ../sbin
$ ./hadoop-daemon.sh start namenode starting namenode, logging to /opt/yarn/hadoop-
2.0.4-alpha/logs/hadoop-hdfs-namenode-limulus.out
$ ./hadoop-daemon.sh start secondarynamenodestarting secondarynamenode, logging to
/opt/yarn/hadoop-2.0.4-alpha/logs/hadoop-hdfs-secondarynamenode-limulus.out
$ ./hadoop-daemon.sh start datanodestarting datanode, logging to /opt/yarn/hadoop-
2.0.4-alpha/logs/hadoop-hdfs-datanode-limulus.out
```

If the daemon started, you should see responses above that will point to the log file. (Note that the actual log file is appended with ".log" not ".out.") . As a sanity check, issue a `jps` command to see that all the services are running. The actual PID values will be different than shown in this listing:

```
$ jps
15140 SecondaryNameNode
15015 NameNode
```

```
15335 Jps
15214 DataNode
```

If the process did not start, it may be helpful to inspect the log files. For instance, examine the log file for the NameNode. (Note that the path is taken from the command above.)

```
vi /opt/yarn/hadoop-2.0.4-alpha/logs/hadoop-hdfs-namenode-limulus.log
```

All Hadoop services can be stopped using the hadoop-daemon.sh script. For example, to stop the datanode service enter the following:

```
$ ./hadoop-daemon.sh stop datanode
```

The same can be done for the Namenode and SecondaryNameNode

Step 12: Start YARN Services

As with HDFS services, the YARN services need to be started. One ResourceManager and one NodeManager must be started as user yarn:

```
# su - yarn
$ cd /opt/yarn/hadoop-2.0.4-alpha/sbin
$ ./yarn-daemon.sh start resourcemanager
starting resourcemanager, logging to /opt/yarn/hadoop-2.0.4-alpha/logs/yarn-yarn-resourcemanager-limulus.out
$ ./yarn-daemon.sh start nodemanager
starting nodemanager, logging to /opt/yarn/hadoop-2.0.4-alpha/logs/yarn-yarn-nodemanager-limulus.out
```

As with starting the HDFS daemons in the previous step, the status of running daemons is sent to the respective log files. To check whether the services are running issue a `jps` command. The following shows all the necessary services to run YARN on a single server:

```
$ jps
15933 Jps
15567 ResourceManager
15785 NodeManager
```

In if there are missing services, check the log file for the specific service.. Similar to HDFS, the services can be stopped by issuing a stop argument to the daemon script:

```
./yarn-daemon.sh stop nodemanager
```

Step 13: Verify the Running Services Using the Web Interface

Both HDFS and the YARN Resource Manager have a web interface. These interfaces are a convenient way to browse many of the aspects of your Hadoop installation. To monitor HDFS enter the following:

```
$ firefox  http://localhost:50070
```

Connecting to port 50070 will bring up the web interface similar to what is shown in Figure 1.1.

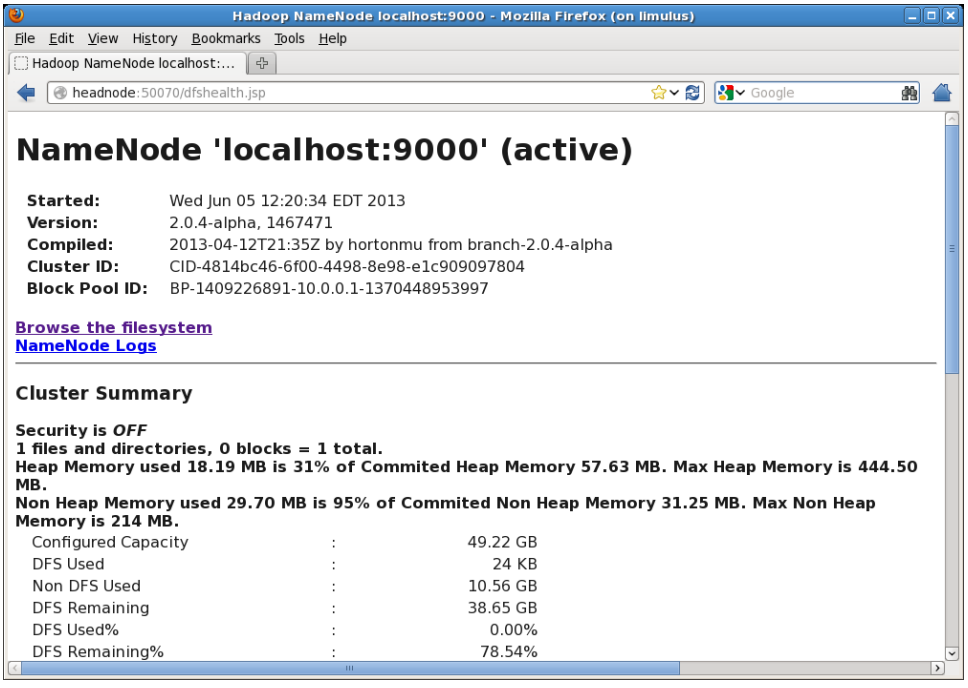


Figure 1.1 Webpage for HDFS filesystem

A web interface for the Resource Manager can be viewed by entering the following:

```
$ firefox http://localhost:8088
```

A web page similar to what is shown in Figure 1.2 will be displayed.

Run Sample MapReduce Examples

To test your installation, run the sample "pi" program that calculates the value of Pi using a quasi-Monte Carlo method and MapReduce. Change to user hdfs and run the following:

```
# su - hdfs
$ cd /opt/yarn/hadoop-2.0.4-alpha/bin
$ ./hadoop jar ../share/hadoop/mapreduce/hadoop-mapreduce-examples-2.0.4-alpha.jar pi -Dmapreduce.clientfactory.class.name=org.apache.hadoop.mapred.YarnClientFactory -libjars ../share/hadoop/mapreduce/hadoop-mapreduce-client-jobclient-2.0.4-alpha.jar 16 1000
```

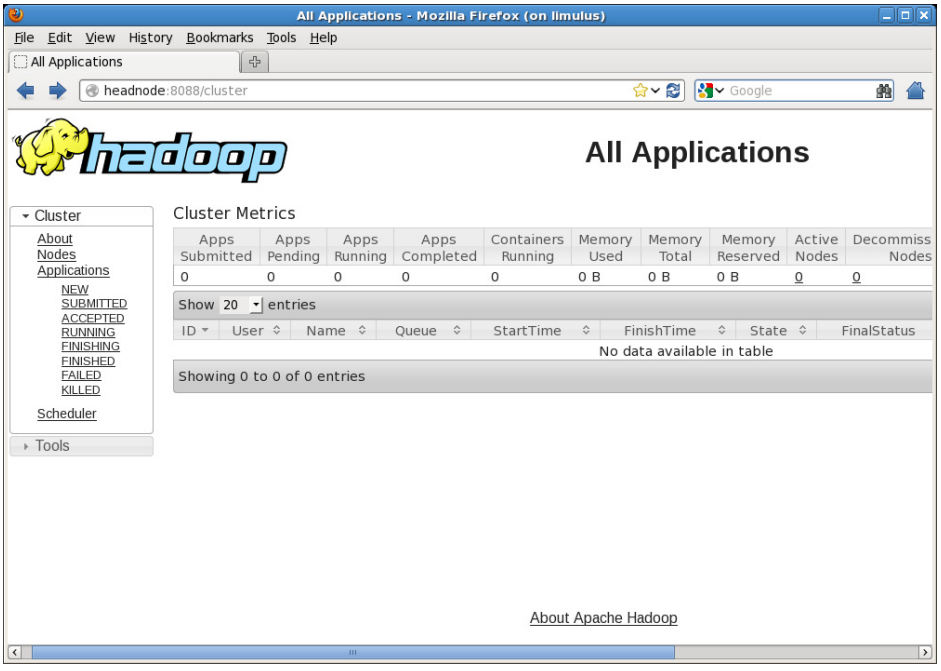


Figure 1.2 Webpage for YARN Resource Manager

If the program worked correctly, the following should be displayed at the end of the program output stream:

```
Estimated value of Pi is 3.14250000000000000000
```

This previous example submits a MapReduce job to YARN from the included samples in the share/hadoop/mapreduce directory. The master JAR file has several sample applications to test your YARN installation. After submitting the job, progress can viewed by updating the Resource Manager web page mentioned previously.

You can get a full list of examples by entering:

```
./hadoop jar ../share/hadoop/mapreduce/hadoop-mapreduce-examples-2.0.4-alpha.jar
```

A list of options for each example can be produced by adding the example name to the above command. The following is a list of the included jobs in the examples JAR file.

- **aggregatewordcount**: An Aggregate based map/reduce program that counts the words in the input files.
- **aggregatewordhist:** An Aggregate based map/reduce program that computes the histogram of the words in the input files.

- **bbp:** A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
- **dbcount**: An example job that counts the pageview counts from a database.
- **distbbp**: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
- **grep:** A map/reduce program that counts the matches to a regex in the input.
- **join:** A job that effects a join over sorted, equally partitioned datasets
- **multifilewc**: A job that counts words from several files.
- **pentomino:** A map/reduce tile laying program to find solutions to pentomino problems.
- **pi:** A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
- **randomtextwriter:** A map/reduce program that writes 10GB of random textual data per node.
- **randomwriter**: A map/reduce program that writes 10GB of random data per node.
- **secondarysort**: An example defining a secondary sort to the reduce.
- **sort:** A map/reduce program that sorts the data written by the random writer.
- **sudoku:** A sudoku solver.
- **teragen:** Generate data for the terasort
- **terasort:** Run the terasort
- **teravalidate:** Checking results of terasort
- **wordcount:** A map/reduce program that counts the words in the input files.

Some of the examples require files to be copied to/from HDFS. For those unfamiliar with basic HDFS operation, an HDFS quick start is provided in Appendix B, “HDFS Quick Start.”

Wrap Up

With a working installation of YARN, the concepts, examples, and applications found in this book can be explored further without the need for a large production cluster. Keep in mind that there are many configuration aspects that were simplified for this single machine installation. In particular, a single workstation/server install does not have a true parallel HDFS or parallel MapReduce component. Additional production installation scenarios will be provided in Chapter 4, “Installing YARN.”

2

YARN and the Hadoop Ecosystem

The new Apache Hadoop YARN resource manager is introduced in this chapter. In addition to enabling non-MapReduce tasks to work within a Hadoop installation, YARN provides several other advantages over the previous version of Hadoop including better scalability, cluster utilization, and user agility.

YARN also brings with it several new services that separate it from the standard Hadoop MapReduce model. A new ResourceManger acting as a pure resource scheduler is the sole arbitrator of cluster resources. User applications, including MapReduce jobs, ask for specific resource requests via the new ApplicationMaster component, which in in-turn negotiates with the ResourceManager to create an application container within the cluster.

By incorporating MapReduce as a YARN framework, YARN also provides full backward compatibility with existing MapReduce tasks and applications.

Beyond MapReduce

The Apache Hadoop ecosystem continues to grow beyond the simple MapReduce job. Although MapReduce is still at the core of many Hadoop 1.0 tasks, the introduction of YARN has expanded the capability of a Hadoop environment to move beyond the basic MapReduce process.

The basic structure of Hadoop with Apache Hadoop MapReduce v1 (MRv1) can be seen in Figure 2.1. The two core services, HDFS and MapReduce, form the basis for almost all Hadoop functionality. All other components are built around these services and must use MapReduce to run Hadoop jobs.

Apache Hadoop provides a basis for large scale MapReduce processing and has spawned a big data ecosystem of tools, applications, and vendors. While MapReduce methods enable the users to focus on the problem at hand rather than the underlying processing mechanism, they do limit some of the problem domains that can run in the Hadoop framework.

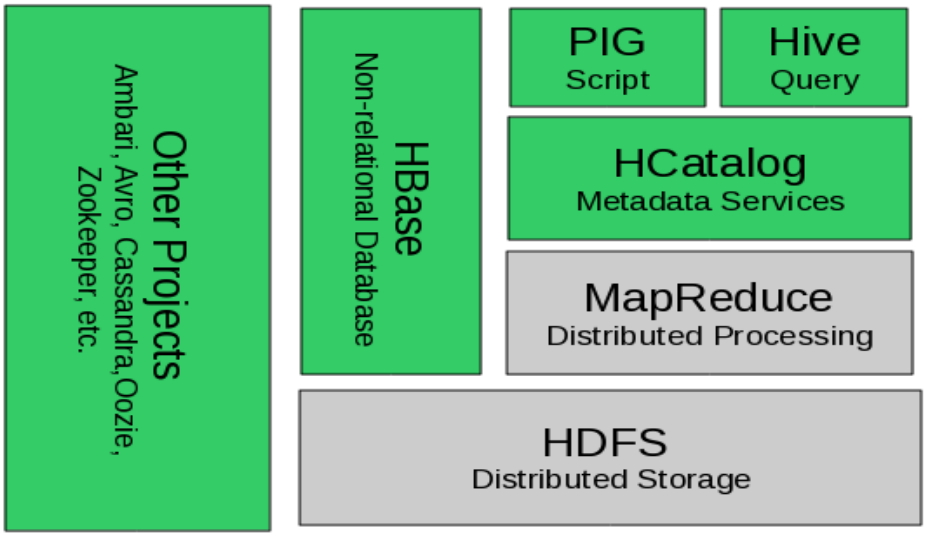


Figure 2.1 The Hadoop 1.0 ecosystem, MapReduce and HDFS are the core components, while other are built around the core.

To address these needs, the YARN (Yet Another Resource Negotiator) project was started by the core development team to give Hadoop the ability to run non-MapReduce jobs within the Hadoop framework. YARN provides a generic resource management framework for implementing distributed applications. Starting with Apache Hadoop version 2.0, MapReduce has undergone a complete overhaul and is now re-architected as an application on YARN to be called MapReduce version 2 (MRv2). YARN provides both full compatibility with existing MapReduce applications and new support for virtually any distributed application. Figure 2.2 illustrates how YARN fits into the new Hadoop ecosystem.

The introduction of YARN does not change the capability of Hadoop to run MapReduce jobs. It does, however, position MapReduce as merely one of the application frameworks within Hadoop, which works the same as it did in MRv1. The new capability offered by YARN is the use of new non-MapReduce frameworks that add many new capabilities to the Hadoop ecosystem.

The MapReduce Paradigm

The MapReduce processing model consists of two separate steps. The first step is an embarrassingly parallel map phase where input data is split into discreet chunks that can be processed independently. The second and final step is a reduce phase where the output of the map phase is aggregated to produce the desired result. The simple, and fairly restricted, nature of the programming model lends itself to very efficient and extremely large-scale implementations across thousands of low cost commodity servers (or nodes).

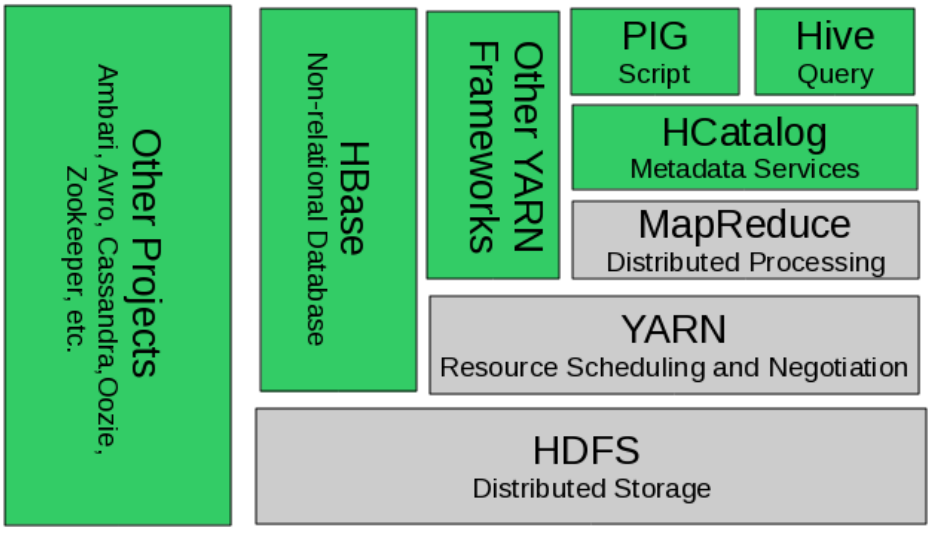


Figure 2.2 YARN adds a more general interface to run non-MapReduce jobs within the Hadoop framework

When MapReduce is paired with a distributed file-system such as Apache Hadoop HDFS (Hadoop File System), which can provide very high aggregate I/O bandwidth across a large cluster of commodity servers, the economics of the system are extremely compelling—a key factor in the popularity of Hadoop.

One of the keys to Hadoop performance is the *lack of data motion* where compute tasks are moved to the servers on which the data reside and not the other way around (i.e., large data movement to compute servers is minimized or eliminated). Specifically, the MapReduce tasks can be scheduled on the same physical nodes on which data are resident in HDFS, which exposes the underlying storage layout across the cluster. This design significantly reduces the network I/O patterns and keeps most of the I/O on the local disk or on a neighboring server within the same server rack.

Apache Hadoop MapReduce

To understand the new YARN process flow, it will be helpful to review the original Apache Hadoop MapReduce design. As part of the Apache Software Foundation, Apache Hadoop MapReduce has evolved and improved as an open-source project. The project is an implementation of the MapReduce programming paradigm described above. The Apache Hadoop MapReduce project itself can be broken down into the following major facets:

- The end-user MapReduce API for programming the desired MapReduce application.
- The MapReduce framework, the run-time implementation of various phases such as the map phase, the sort/shuffle/merge aggregation and the reduce phase.

- The MapReduce system, the back-end infrastructure required to run MapReduce applications, manage cluster resources, schedule thousands of concurrent jobs, etc.

This separation of concerns has significant benefits, particularly for end-users where they can completely focus on their application via the API and let the combination of the MapReduce Framework and the MapReduce System deal with the complex details such as resource management, fault-tolerance, and scheduling.

The current Apache Hadoop MapReduce system is composed of the several high level elements as shown in Figure 2.3. The master process is the JobTracker, which is the clearing house for all MapReduce jobs on in the cluster. Each Node has a TaskTracker process that manages tasks on the individual node. The TaskTrakers communicate with and are controlled by the JobTracker.

The JobTracker is responsible for *resource management* (managing the worker server nodes i.e. TaskTrackers), *tracking resource consumption/availability* and also *job life-cycle management* (scheduling individual tasks of the job, tracking progress, providing fault-tolerance for tasks. etc).

The TaskTracker has simple responsibilities—launch/teardown tasks on orders from the JobTracker and provide task-status information to the JobTracker periodically.

The Apache Hadoop MapReduce framework has exhibited some growing pains. In particular, with regards to the JobTracker, several aspects including scalability, cluster utilization, capability for users to control upgrades to the stack, i.e., *user agility* and, support for workloads other than MapReduce itself have been identified as desirable features.

The Need for Non-MapReduce Workloads

MapReduce is great for many applications, but not everything; other programming models better serve requirements such as graph processing (e.g., Google Pregel/Apache Giraph) and iterative modeling using Message Passing Interface (MPI). As is often the case, much of the enterprise data is already available in Hadoop HDFS and having multiple paths for processing is critical and a clear necessity. Furthermore, since MapReduce is essentially batch-oriented, support for real-time and near real-time processing has become an important issue for the user base. A more robust computing environment within Hadoop enables organizations to see an increased return on the Hadoop investments by lowering operational costs for administrators, reducing the need to move data between Hadoop HDFS and other storage systems, providing other such efficiencies.

Addressing Scalability

The processing power available in data-centers continues to increase rapidly. As an example, consider the additional hardware capability offered by a commodity server over a three-year period:

- 2009 – 8 cores, 16GB of RAM, 4x1TB disk
- 2012 – 16+ cores, 72GB of RAM, 12x3TB of disk.

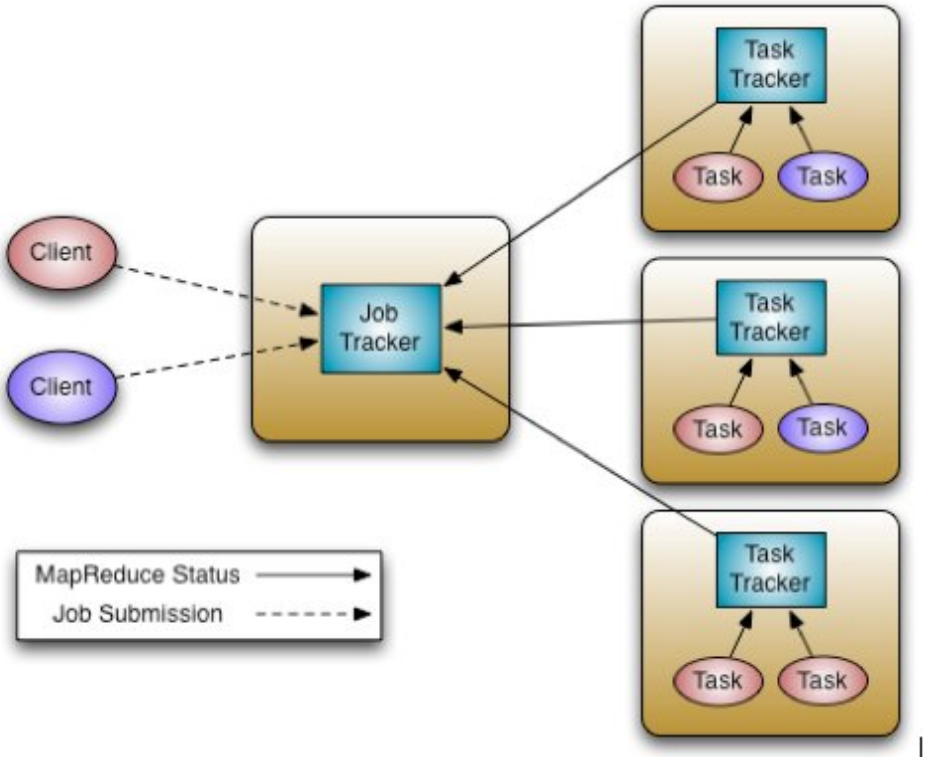


Figure 2.3 Current Hadoop MapReduce Control Elements

These new servers are often available at the same price-point as those of previous generations. In general, servers are twice as capable today as they were 2-3 years ago—on every single dimension. Apache Hadoop MapReduce is known to scale to production deployments of approximately 5000 server nodes of 2009 vintage. Thus, for the same price the number of CPU cores, amount of RAM, and local storage available to the user will put continued pressure on the scalability of new Apache Hadoop installations.

Improved Utilization

In the current system, the JobTracker views the cluster as composed of nodes (managed by individual TaskTrackers) with distinct map slots and reduce slots, which are not *fungible*. Utilization issues occur because maps slots might be 'full' while reduce slots are empty (and vice-versa). Improving this situation is necessary to ensure the entire system could be used to its maximum capacity for high utilization and applying resources when needed.

User Agility

In real-world deployments, Hadoop is very commonly offered as a shared, multi-tenant system. As a result, changes to the Hadoop software stack affect a large cross-section of, if not the entire, enterprise. Against that backdrop, users are very keen on controlling upgrades to the software stack as it has a direct impact on their applications. Thus, allowing multiple, if limited, number of versions of the MapReduce framework is critical for Hadoop.

Apache Hadoop YARN

The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker, in other words resource management and job scheduling/monitoring, into separate daemons: a global ResourceManager and per-application ApplicationMaster (AM). The ResourceManager and per-node slave, the NodeManager (NM), form the new, and *generic***,** operating system for managing applications in a distributed manner.

The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is, in effect, a *framework specific* entity and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the component tasks.

The ResourceManager has a pluggable scheduler component, which is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is a *pure scheduler* in the sense that it performs no monitoring or tracking of status for the application, offering no guarantees on restarting failed tasks either due to application failure or hardware failures. The scheduler performs its scheduling function based on the *resource requirements* of an application by using the abstract notion of a *resource container,* which incorporates resource dimensions such as memory, CPU, disk, network etc.

The NodeManager is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (CPU, memory, disk, network), and reporting the same to the ResourceManager.

The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress. From the system perspective, the ApplicationMaster itself runs as a normal *container*. An architectural view of YARN is provided in Figure 2.4.

One of the crucial implementation details for MapReduce within the new YARN system is the reuse of the existing MapReduce framework without any major surgery. This step was very important to ensure compatibility for existing MapReduce applications and users.

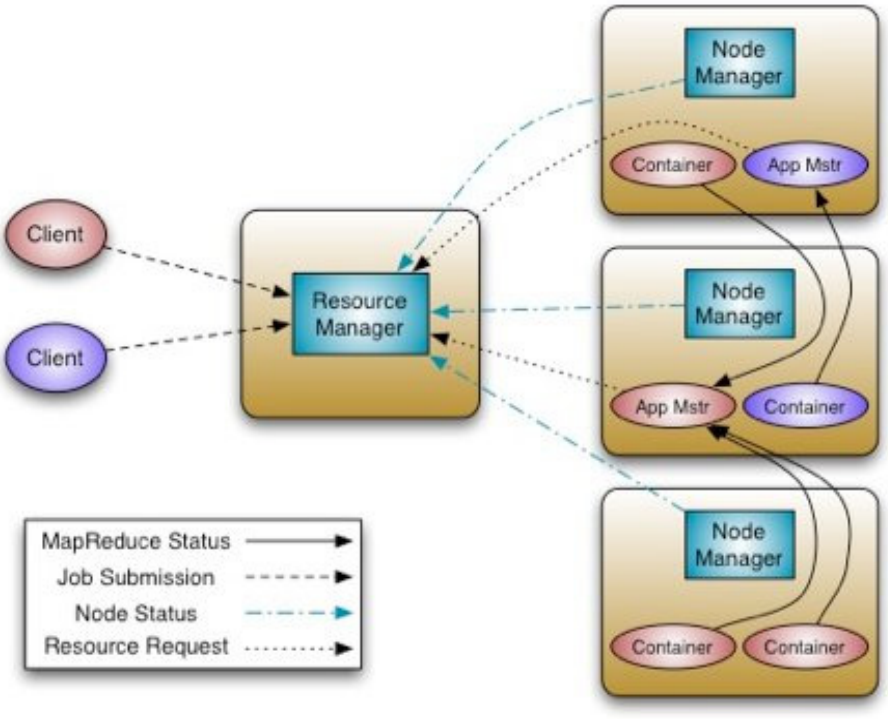


Figure 2.4 New Yarn Control Elements

YARN Components

By adding new functionality, YARN brings in new components into the Apache Hadoop workflow. These components provide a finer grain of control for the end user and at the same time offer more advanced capabilities to the Hadoop ecosystem.

Resource Manager

As mentioned, the YARN ResourceManager is primarily a *pure scheduler*. It is strictly limited to arbitrating available resources in the system among the competing applications. It optimizes for cluster utilization (keeps all resources in use all the time) against various constraints such as capacity guarantees, fairness, and SLAs. To allow for different policy constraints the ResourceManager has a *pluggable scheduler* that enables different algorithms such as capacity and fair scheduling to be used as necessary.

ApplicationMaster

An important new *concept* in YARN is the ApplicationMaster. The ApplicationMaster is, in effect, an *instance* of a *framework-specific library* and is responsible for negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the containers and their resource consumption. It has the responsibility of negotiating appropriate resource containers from the ResourceManager, tracking their status and monitoring progress.

The ApplicationMaster design enables YARN to offer the following important new features:

- Scale: The Application Master provides much of the functionality of the traditional ResourceManager so that the entire system can scale more dramatically. Simulations have shown jobs scaling to 10,000 node clusters composed of modern hardware without significant issue. As a pure scheduler the ResourceManager does not, for example, have to provide fault-tolerance for resources across the cluster. By shifting fault tolerance to the ApplicationMaster instance, control becomes local and not global. Furthermore, since there is an instance of an ApplicationMaster per application, the ApplicationMaster itself isn't a common bottleneck in the cluster.
- Open: Moving all application framework specific code into the ApplicationMaster generalizes the system so that we can now support multiple frameworks such as MapReduce, MPI and Graph Processing.

These features were the result of some key YARN design decisions:

- Move all complexity (to the extent possible) to the ApplicationMaster while providing sufficient functionality to allow application-framework authors sufficient flexibility and power.
- Since it is essentially user-code, do not trust the ApplicationMaster(s). In other words. no ApplicationMaster is a privileged service.
- The YARN system (ResourceManager and NodeManager) has to protect itself from faulty or malicious ApplicationMaster(s) and resources granted to them at all costs.

It's useful to remember that, in reality, every application has its own instance of an ApplicationMaster. However, it's completely feasible to implement an ApplicationMaster to manage a set of applications (e.g., ApplicationMaster for Pig or Hive to manage a set of MapReduce jobs). Furthermore, this concept has been stretched to manage long-running services, which manage their own applications (e.g., launch HBase in YARN via a hypothetical HBaseAppMaster).

Resource Model

YARN supports a very general resource model for applications. An application (via the ApplicationMaster) can request resources with highly specific requirements such as:

- Resource-name (including hostname, rackname and possibly complex network topologies)

- Amount of Memory
- CPUs (number/type of cores)
- Eventually resources like disk/network I/O, GPUs, etc.

ResourceRequest and Containers

YARN is designed to allow individual applications (via the ApplicationMaster) to utilize cluster resources in a shared, secure and multi-tenant manner. It also remains aware of cluster topology in order to efficiently schedule and optimize data access (i.e., reduce data motion for applications to the extent possible).

In order to meet those goals, the central Scheduler (in the ResourceManager) has extensive information about an application's resource needs, which allows it to make better scheduling decisions across all applications in the cluster. This leads us to the ResourceRequest and the resulting Container.

Essentially an application can ask for specific resource requests via the ApplicationMaster to satisfy its resource needs. The Scheduler responds to a resource request by granting a container, which satisfies the requirements laid out by the ApplicationMaster in the initial ResourceRequest.

A ResourceRequest has the following form:

```
<resource-name, priority, resource-requirement, number-of-containers>
```

These components are described as follows:

- Resource-name is either hostname, rackname or * to indicate no preference. Future plans may support even more complex topologies for virtual machines on a host, more complex networks, etc.
- Priority is intra-application priority for this request (not across multiple applications).
- Resource-requirement is required capabilities such as memory, CPU, etc. (currently YARN only supports memory and CPU).
- Number-of-containers is just a multiple of such containers.

Essentially, the Container is the resource allocation, which is the successful result of the ResourceManager granting a specific ResourceRequest. A Container grants rights to an application to use a specific amount of resources (memory, CPU etc.) on a specific host.

The ApplicationMaster has to take the Container and present it to the NodeManager managing the host, on which the container was allocated, to use the resources for launching its tasks. For security reasons, the Container allocation is verified, in the secure mode, to ensure that ApplicationMaster(s) *cannot fake allocations* in the cluster.

Container Specification

While a Container, as described above, is merely a *right* to use a specified amount of resources on a specific machine (NodeManager) in the cluster, the ApplicationMaster has to provide considerably more information to the NodeManager to actually *launch* the container. YARN allows applications to launch any process and, unlike existing Hadoop MapReduce, it isn't limited to Java applications.

The YARN Container launch specification API is platform agnostic and contains:

- Command line to launch the process within the container.
- Environment variables.
- Local resources necessary on the machine prior to launch, such as jars, shared-objects, auxiliary data files etc.
- Security-related tokens.

This design allows the ApplicationMaster to work with the NodeManager to launch containers ranging from simple shell scripts to C/Java/Python processes on Unix/Windows to full-fledged virtual machines.

Wrap Up

The release of Apache Hadoop YARN provides many new capabilities to the existing Hadoop big data ecosystem. While the scalable MapReduce paradigm has enabled previously intractable problems to be efficiently managed on large clustered systems, YARN provides a framework for managing both MapReduce and non-MapReduce tasks of greater size and complexity. YARN provides the framework to apply low cost commodity hardware to virtually any big data problem.