

Hadoop

IN PRACTICE

Alex Holmes



- 42 TECHNIQUES
- GET IT DONE
- GET SAVVY

MEAP

 MANNING



MEAP Edition
Manning Early Access Program
Hadoop in Practice version 6

Copyright 2012 Manning Publications

For more information on this and other Manning titles go to
www.manning.com

Table of Contents

Part I: Background and Fundamentals

Chapter 1: Getting started

Part II: Data Logistics

Chapter 2: Moving data in and out of Hadoop

Chapter 3: Data serialization: working with text and beyond

Part III: Big Data patterns

Chapter 4: Applying MapReduce patterns to Big Data

Chapter 5: Streamlining HDFS for Big Data

Chapter 6: Diagnosing and tuning performance problems

Part IV: Data science

Chapter 7: Utilizing data structures and algorithms

Chapter 8: Integrating R and Hadoop for statistics and more

Chapter 9: Predictive analytics with Mahout

Part V: Taming the elephant

Chapter 10. Hacking with Hive

Chapter 11. Programming Pipelines with Pig

Chapter 12. Crunch and other technologies

Chapter 13. Testing and debugging



Preface

0.1 General Information

0.1.1 Hadoop Distributions

There are many factors to weigh in your Hadoop distribution decision. Do you go with the Apache distribution of Hadoop, or pick a commercial distribution? The advantages of going with a non-Apache distribution of Hadoop include support, packaged installation, back-porting of important bugs, and the addition of enterprise features to help with administrative activities. Some vendors such as MapR go even further and offer a complete replacement for layers such as the distributed file system, and provide a HA replacement of components such as the NameNode and JobTracker.

When it comes to support you have two options, go without a support contract, or pay one of the vendors that offer a support contract. If you pay a vendor for support, they typically will only support their own distribution of Hadoop. Deciding whether support is important to you will depend on a variety of factors, such as whether you have any Service Level Agreements (SLA's) around any of the work you do with Hadoop, and whether you have the required in-house Hadoop expertise to problem-solve issues as they come up. Often support costs are tied to the size of your cluster, and can get expensive fast, even with modest-sized clusters (under 100 nodes).

If you decide not to pay for support, then in some ways your decision around which distribution to use requires even more thought, as you are now responsible for diagnosing and fixing issues. As a result an emphasis should be placed on the size of the user community support for the distributions, as this will surely provide an indication of the level of support you will receive.

Let's review the top Hadoop distribution offerings on the market today.

APACHE

Apache's distribution of Hadoop is the one with the least amount of bells and whistles. Assistance and packaged installation, management and support are not provided. However this is the distribution with the largest community support, where there's a good chance of receiving help with any queries. If you are running a non-Apache distribution of Hadoop and post a question to the Apache mailing lists, there's a chance your question won't be answered and instead you'll be directed to post to the distributor's mailing list.

CLOUDERA

Cloudera pioneered the notion of creating supported Hadoop distributions back in 2009. It made the smart decision to hire some of the top Hadoop committers such as Doug Cutting, Tom White, Jeff Hammerbacher and Todd Lipcon to mention just a few.

Their Cloudera Distribution for Hadoop (CDH) offers a packaged distribution of Hadoop which is freely available. They not only distribute and support the core MapReduce and HDFS components, but also a large number of other Hadoop-related projects ¹ such as Pig, Hive, HBase, Sqoop and more. They also offer a Cloudera Enterprise suite, available to customers with a support contract, which consists of tools and access to resources to help improve the management and monitoring of your Hadoop cluster.

Footnote 1 <http://www.cloudera.com/hadoop-details/>

Cloudera also provides popular training and certification courses.

HORTONWORKS

The Yahoo team that worked on Hadoop was spun-off by Yahoo into a separate company, Hortonworks, to allow them to be more focused on providing their own Hadoop distribution. It contains high-profile Hadoop committers such as Arun Murthy, Owen O'Malley and Alan Gates.

At the time of writing they are still working on creating a Hadoop distribution. They are also planning training and certification.

MAPR

MapR joined the Hadoop distribution ranks in 2011 with a revolutionary new file system and High Availability features for both MapReduce and HDFS.

MapR offers a free version of its distribution, titled "M3", and an enterprise

edition called "M5". M3 includes a light-weight version of their file system, MapReduce, HBase, Hive, Pig and more.²

Footnote 2 <http://www.mapr.com/doc/display/MapR/Installation+Guide>

With the M5 edition MapR go above and beyond the offerings of Cloudera and Hortonworks by providing their own distributed file system which does not contain some of the limitations contained in HDFS, such as being able to work with small files and a distributed High Availability (HA) NameNode. It also offers a HA MapReduce Job Tracker, and mirroring capabilities to allow you to mirror data between different clusters.

There are a number of other Hadoop distributions³ which may be worth evaluating from Greenplum and others.

Footnote 3 <http://wiki.apache.org/hadoop/Distributions%20and%20Commercial%20Support>

You may be curious about what version of Hadoop these distributions are bundling. Let's find out, and in the process review the current and future Hadoop versions to see where Hadoop is today, as well as what is coming in the future.

0.1.2 Hadoop Versions

Hadoop only recently hit a 1.0 version (the release was originally targeted to be 0.20.205.1, but it was deemed mature and stable enough to become the 1.0 version), which could be interpreted to denote a lack of maturity. However versions marked stable on the Hadoop website have usually gone through extensive testing at large Yahoo clusters, and are considered safe for use in production. Therefore Hadoop is more stable than the version numbers may suggest.

At the time of writing the 0.20.X and 0.20.20X series of releases are the only releases marked as stable, and therefore production-ready. This is reinforced by looking at the versions included in the Hadoop distributions, which are all 0.20.X-based. There is a 0.21 release of Hadoop, but it hasn't been marked as stable, in part due to a broken implementation of file append. At the time of writing, active development is occurring on the 0.20 and 0.23 code bases, with 0.23 being the next large release.

THE API CAMELEON (AKA MAPREDUCE)

The MapReduce API has had its share of drama in its lifetime. The `org.apache.hadoop.mapred` package has existed since 2007, but was deprecated in the 2009 0.20 release of Hadoop in favor of a new package in `org.apache.hadoop.mapreduce`. A year later the 0.21 code base un-deprecated the old API due to the new API being considered incomplete. Fast forward two years later at the end of 2011 and now the 0.20 releases are also being updated to undeprecate the API. And to top things off there's a third MapReduce API planned in the 0.23 release!

All of this may put you in a quandary about which API you should choose. We find it curious that all the example MapReduce code bundled with Hadoop still uses the old (now undeprecated code), yet mailing list activities suggest that at some point the old API may be removed altogether. Therefore, where possible, we will focus on the "new" `org.apache.hadoop.mapreduce` API for this book.

HADOOP 0.23 AND BEYOND

Upcoming Hadoop releases have many major features which will help solve some problems that we face with Hadoop today. Let's touch upon a few which look promising.

- HDFS Federation, tracked by Jira ticket [HDFS-1623](#), is a move to scale HDFS by supporting the notion of multiple independent HDFS namespaces, managed by multiple NameNodes. This helps scale the number of blocks, files and directories, which today are bounded by the memory on a single NameNode. Multiple NameNode also means NameNodes can be dedicated for HBase, or dedicated to single tenants in a multi-tenant environment to prevent overloading and denying NameNode services to other cluster users.
- Next-Gen MapReduce, also dubbed MapReduce 2.0, is tracked in Jira ticket [MAPREDUCE-279](#), and is a complete overhaul of the MapReduce architecture. The primary goal is to provide improve reliability and availability, and to be able to scale to hundreds of thousands of nodes in a cluster. The architecture generalizes the notion of a cluster with resources, and actually moves MapReduce into user-land, meaning rather than the entire cluster being a certain version of MapReduce, the cluster is simply a distributed resource pool, which can run multiple versions of MapReduce. This opens-up how a cluster can be utilized, so not just MapReduce jobs can be run, but also iterative processes, machine learning and other programming paradigms.
- High Availability NameNodes ([HDFS-1623](#)) and Job Trackers ([MAPREDUCE-225](#)) will remove the Single Points Of Failure (SPOF) in Hadoop's current architecture.
- HDFS sync support, meaning that appending to files will be permitted ([HDFS-265](#), included in 0.23)

0.1.3 Book Code and Hadoop Version

All of the text and examples in this book work with Hadoop 0.20.x (and 1.x), and most of the code is written using the newer `org.apache.hadoop.mapreduce` MapReduce API's. The few examples that leverage the older `org.apache.hadoop.mapred` package are usually a result of working with a third-party library or utility that only works with the old API.

0.1.4 Code for the Book

All of the code used in this book is available on github at <https://github.com/alexholmes/hadoop-book>. Building the code depends on Java 1.6 or newer, Git and Maven 3.0 or newer. Git is a source control management system, and GitHub provides hosted git repository services. Maven is used for our build system.

You can clone (download) our git repository with the following command.

```
$ git clone git://github.com/alexholmes/hadoop-book.git
```

After the sources are downloaded you can build our code.

```
$ cd hadoop-book
$ mvn package
```

This will create a Java JAR file `target/hadoop-book-1.0.0-SNAPSHOT-jar-with-dependencies.jar`. Running our code is equally simple with our included `bin/run.sh`.

If you are running on a CDH distribution then the scripts will run configuration-free. If you are running any other distribution you'll need to set the `HADOOP_HOME` environment variable to point to your Hadoop installation directory.

The `bin/run.sh` script takes as the first argument the fully-qualified Java class name of the example, followed by any arguments expected by the example class. As an example, to run our inverted index MapReduce code from Chapter 1 we would run the following.

```

$ hadoop fs -mkdir /tmp
$ hadoop fs -put test-data/ch1/* /tmp/

# replace the path below with the location of your Hadoop installation
# this isn't required if you are running CDH3
export HADOOP_HOME=/usr/local/hadoop

$ bin/run.sh com.manning.hip.ch1.InvertedIndexMapReduce \
  /tmp/file1.txt /tmp/file2.txt output

```

The above code won't work if you don't have Hadoop installed. Please refer to Chapter 1 for CDH installation instructions, or Appendix A for Apache installation instructions.

THIRD PARTY LIBRARIES

We use a number of third-party libraries for convenience purposes. They are included in the Maven-built JAR so there's no extra work required to work with these libraries. The following table contains a list of the libraries that are in prevalent use throughout the code examples.

Table 0.1 Common 3rd Party Libraries

| Library | Link | Details |
|---------------------|---|--|
| Apache Commons IO | http://commons.apache.org/io/ | Helper functions to help work with input and output streams in Java. We make frequent use of the <code>IOUtils</code> to close connections and for reading the contents of files into strings. |
| Apache Commons Lang | http://commons.apache.org/lang/ | Helper functions to work with Strings, dates and Collections. We make frequent use of the <code>StringUtils</code> class for tokenization. |

0.1.5 Data Sets

Throughout this book we work with three data sets to provide some variety throughout our examples. All the data sets are small to make them easy to work with. Copies of the exact data we used are available in our github repository in the directory <https://github.com/alexholmes/hadoop-book/tree/master/test-data> . We also sometimes have data that's specific to a chapter, which exists within chapter-specific subdirectories under the same github location.

NASDAQ FINANCIAL STOCKS

We downloaded the NASDAQ daily exchange data from InfoChimps ⁴ . We filtered this huge dataset down to just 5 stocks and their start-of-year values from 2000 through 2009. The data we used for this book is available on our github at <https://github.com/alexholmes/hadoop-book/blob/master/test-data/stocks.txt> .

Footnote 4

<http://www.infochimps.com/datasets/nasdaq-exchange-daily-1970-2010-open-close-high-low-and-volume>

The data is in CSV form, and the field are in the following order.

```
Symbol,Date,Open,High,Low,Close,Volume,Adj Close
```

APACHE LOG DATA

We created a sample log file in Apache Common Log Format ⁵ with some fake Class E IP addresses and some dummy resources and response codes. The file is available on our github at <https://github.com/alexholmes/hadoop-book/blob/master/test-data/apachelog.txt> .

Footnote 5 <http://httpd.apache.org/docs/1.3/logs.html#common>

NAMES

The government's census was used to retrieve names from <http://www.census.gov/genealogy/names/dist.all.last> and available at <https://github.com/alexholmes/hadoop-book/blob/master/test-data/names.txt> .

Getting Started

We live in the age of Big Data, where the data volumes that we need to work with on a day-to-day basis have outgrown the storage and processing capabilities of a single host. This Big Data era brings with it two fundamental challenges; how do we store and work with voluminous data sizes, and even more importantly, how do we understand and make new discoveries in our data in order to transform our data into competitive advantages for our businesses.

Hadoop fills a gap in the market for a distributed system with efficient storage and computational models to work with our voluminous data. Hadoop is a distributed system that consists of a distributed file system, and a mechanism to parallelize and execute programs on a cluster of machines. It has been adopted by technology giants such as Yahoo!, Facebook and Twitter to address their Big Data needs, and is making inroads across all industrial sectors. An instrumental part of Hadoop's success is due to its ability to grind elephantine data sizes with vast clusters of cheap hardware. Mention of cheap hardware may cause some grumbling about cheap equating to unreliable, but Hadoop's fault tolerance take care of that without the need for Operational intervention.

Working with Hadoop also supports our data mining and machine learning/predictive analytical needs. This is one of the primary roles of Hadoop clusters in Yahoo!, Facebook and Twitter - that of mining new facts out of large quantities of data, a field which has gaining the moniker of "Data Science". The ultimate upshot of all this activity is that knowledge we gain from Data Science can be turned into features that will differentiate us from our competitors.

When working with distributed systems such as Hadoop it is important to understand their functional and architectural properties such as consistency, real-time or batch access models, failover support, availability, and scalability to

name a few. We should also understand their limitations, how they compare to competitors in the market, and what use cases they are most effective at supporting. This chapter is intended to cover these areas with regards to Hadoop, and then look at how to install and run a job, which we'll also write.

NOTE**Book Audience**

This book is a hands-on, cookbook-style text targeted at users that have some hands-on experience with Hadoop, and understand the basic concepts of MapReduce and HDFS. The book *Hadoop in Action* by Chuck Lam contains the necessary prerequisites needed to understand and apply the techniques covered in this book.

Many techniques in this book are Java-based, and we also assume that users possess intermediate-level knowledge of Java. An excellent text for all levels of Java users is *Effective Java* by Joshua Bloch.

Let's get things started with a more detailed overview of Hadoop.

1.1 What is Hadoop?

Hadoop is a platform that provides both distributed storage and computational capabilities. It started life within Nutch, an open-source crawler and search engine implementation. In its early life Nutch suffered from scalability issues. After Google published its Google File System (GFS) and MapReduce papers presenting novel distributed storage and computational systems, it was decided to write implementations of these systems in Nutch. The success of this work resulted in its split from Nutch into a separate project which then became Hadoop as we know it today, a first-class Apache project.

This section will look at Hadoop from an architectural perspective, examine how it is being utilized in industry, consider some weaknesses and wrap-up with a brief look at some alternative technologies.

Hadoop proper, as seen in Figure 1.1, is a distributed master-slave architecture that consists of the Hadoop Distributed File System (HDFS) for storage, and MapReduce for computational capabilities. Traits intrinsic to Hadoop are data partitioning and parallel computation of very large data sets. Its storage and computational capabilities scale with the addition of hosts to a Hadoop cluster, and can reach volume sizes in the petabytes on clusters with thousands of hosts.

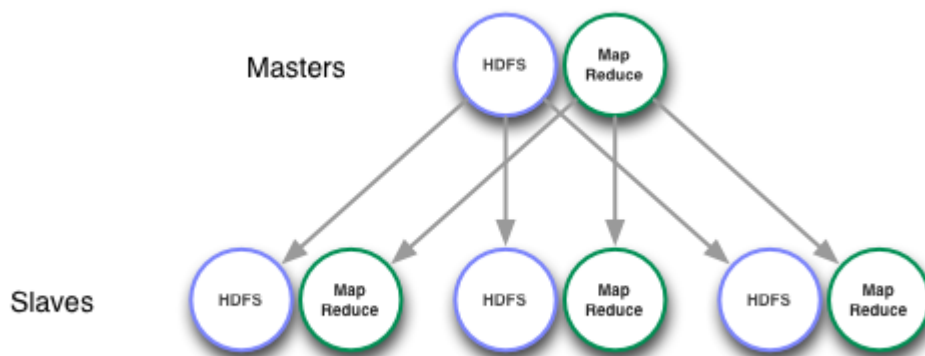


Figure 1.1 High Level Hadoop Architecture

Our first step in this section will be a look at the HDFS and MapReduce architecture.

1.1.1 Core Hadoop Components

HDFS offers a file system with sacrifices to certain file system archetypes to meet its performance and scalability needs. Our next section covers some HDFS basics.

HDFS

The Hadoop Distributed File System (HDFS) is the storage part of Hadoop. It is a distributed file system which is modelled after the Google File System (GFS) paper ¹. It is optimized for high throughput and works best when reading and writing large files (gigabytes and larger). To support this throughput it leverages unusually large (for a file system) block sizes and data locality optimizations to reduce network Input/Output (IO).

Footnote 1 The Google File System, <http://labs.google.com/papers/gfs.html>.

Scalability and availability are also key traits of HDFS achieved in part due to data replication and fault tolerance. HDFS will replicate files a configured number of times, is tolerant of both software and hardware failure, and will automatically re-replicate data blocks on nodes that have failed.

Figure 1.2 shows a logical representation of the components in HDFS, the NameNode and DataNodes. It also shows an application that's using the Hadoop File System library to access HDFS. Let's define the responsibilities of each of these components.

- **NameNode**. The NameNode can be viewed as the volume manager for HDFS. It maintains an in-memory mapping of files to blocks, and for each block keeps track of the nodes that actually store the data for that block. It is consulted by any client that needs to perform a file system operation.

- **DataNodes.** DataNodes are responsible for storing the actual file blocks. Their architecture allows them to pipeline writes to each other. They communicate with the NameNode to inform it of the blocks that the DataNode owns. In turn the NameNode communicates back to the DataNodes with block deletion and block re-replication messages.
- **HDFS Client.** The HDFS Client is an application that wants to perform a file system activity, which could be writing a file, deleting a file, and so on. It always consults the NameNode for any given operation, and depending on the operation it may interact directly with DataNodes (for example for file reads and writes).

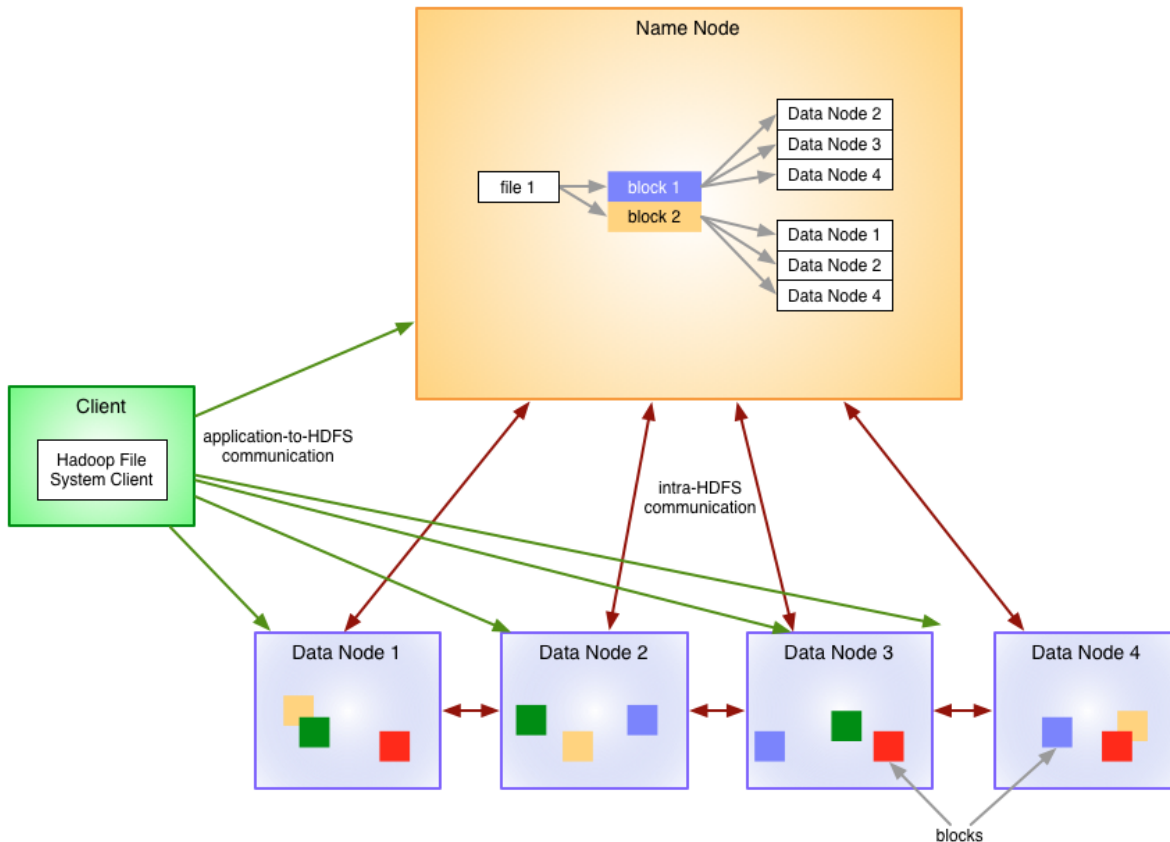


Figure 1.2 HDFS Architecture showing a HDFS client communicating with the master NameNode and slave DataNodes

We've covered the bare essentials of HDFS, but not to worry, we have many more details coming in Chapter 5. As it turns out HDFS is utilized by MapReduce, which is Hadoop's computation engine, so let's now take a look at what MapReduce has to offer us.

MAPREDUCE

MapReduce is a batch-based distributed computing framework modeled after Google's paper on MapReduce². MapReduce abstracts away typical processes involved in working with distributed systems, such as dealing with unreliable hardware and software, computational parallelization, and work distribution. It decomposes work submitted by a client into small parallelized map and reduce workers as seen in Figure 1.3.

Footnote 2 MapReduce: Simplified Data Processing on Large Clusters,
<http://labs.google.com/papers/mapreduce.html>.

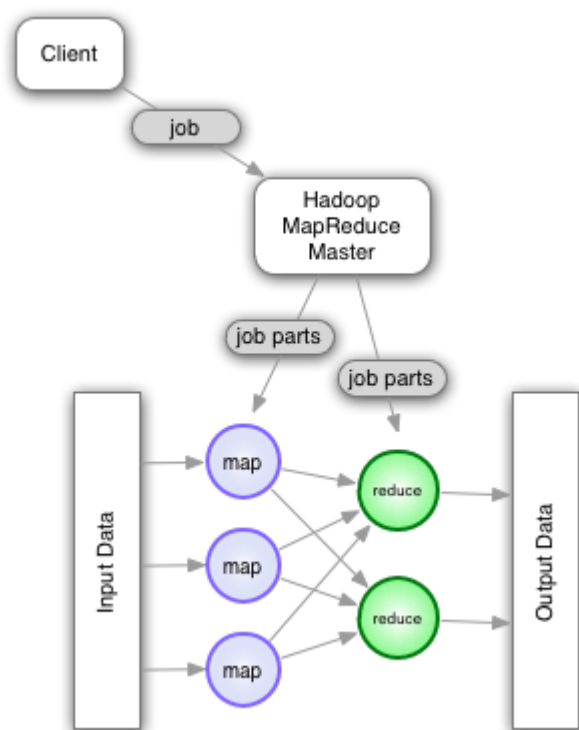


Figure 1.3 A client submitting a job to MapReduce, and the deconstruction of the job into multiple map and reduce parts

MapReduce is a programming model that is engineered to process and generate large data sets. It is inspired after the map and reduce constructs found in the Lisp functional programming language, and espouses a shared-nothing model³. Programmers need to define map and reduce functions, where the map functions output key/value tuples, which are processed by reduce functions to produce the final output.

Footnote 3 A shared-nothing architecture is a distributed computing concept which represents the notion that each node is independent and self-sufficient.

Figure 1.4 shows a pseudo-code definition of a map function with regards to its input and output. The map function takes as input a key/value pair, which represents a logical record from the input data source. In the case of a file, this could be a line, or if the input source is a table in a database, it could be a row. The map function then produces zero or more output key/value pairs for that one input pair. For example, if the map function was a filtering map function, it may only produce output if a certain condition is met. Or it could be performing a demultiplexing operation, where a single key/value yields multiple key/value output pairs.

```
map(key1, value1) → list(key2, value2)
```

Figure 1.4 A logical view of the map function

The power of MapReduce occurs in between the Map output and the Reduce input, in the Shuffle and Sort phases, as seen in Figure 1.5 . The Shuffle and Sort phases are responsible for two primary activities, determining the Reducer that should receive the Map output key/value pair (called Partitioning), and ensuring that for a given Reducer that all its input keys are sorted. Only the Map-side performs the partitioning, but both the Map and Reduce side participate in sorting. Notice in our figure that Map outputs for the same key go to the same Reducer, and are then combined together to form a single input record for the Reducer. Also note that each Reducer has all its input keys sorted, as in the example of "Reducer 2" in Figure 1.5 .

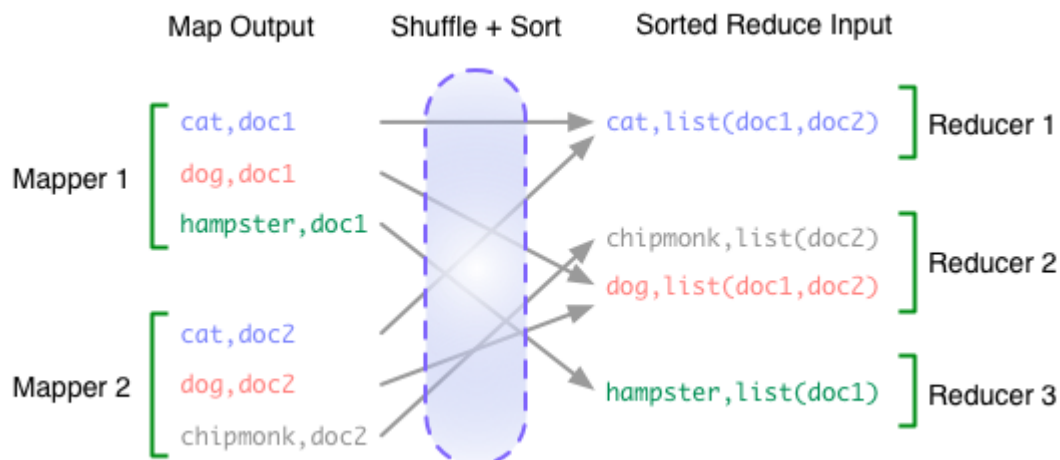


Figure 1.5 MapReduce's Shuffle and Sort

Figure 1.6 shows a pseudo-code definition of a reduce function. The Reduce function is called once per unique Map output key, and all the Map output values that were emitted across all the Mappers for that key. Like the Map function it can output zero-to-many key/value pairs. Reducer output can be written to flat files in HDFS, insert/update rows in a NoSQL database or write to any data sink depending on the requirements of the job.

```

reduce (key2, list (value2's)) → list(key3, value3)
      ↑
      |
all values for key 2

```

Figure 1.6 A logical view of the reduce function

Figure 1.7 shows Hadoop's MapReduce logical architecture. It follows a similar master-slave model as HDFS's. The main components of MapReduce are as follows.

- **JobTracker** . The JobTracker co-ordinates activities between the slave TaskTracker processes. It accepts MapReduce job requests from clients and schedules Map and Reduce tasks on TaskTrackers to perform the work.
- **TaskTracker**. The TaskTracker is a daemon process that spawns child processes to perform the actual Map or Reduce work. Map tasks typically read their input from HDFS, and write their output to the local disk. Reduce tasks read the Map outputs over the network and typically write their output back to HDFS.

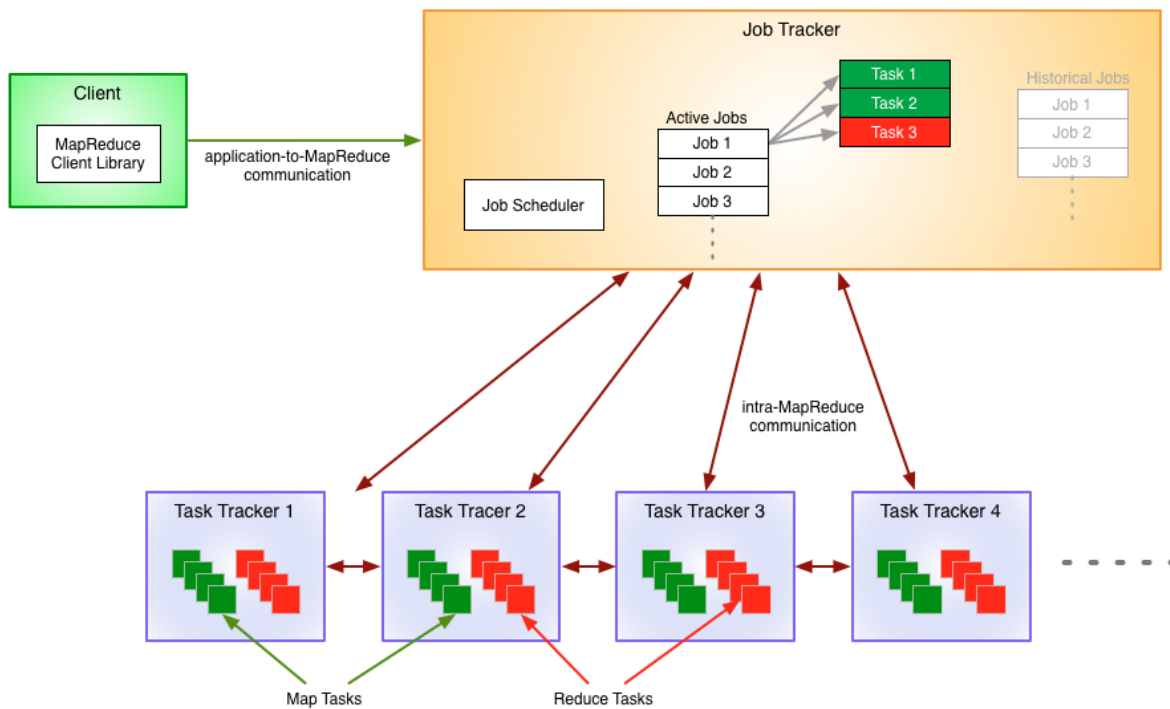


Figure 1.7 MapReduce Logical Architecture

We've covered the core Hadoop components, so let's now take a look at the Hadoop ecosystem, and specifically the products that we'll be covering in this book.

1.1.2 The Hadoop Ecosystem

The Hadoop ecosystem is diverse and grows by the day. It's impossible to keep track of all the various projects that interact with Hadoop in some form. For this book our focus is the tools that are currently receiving the highest adoption from users, as shown in Figure 1.8 .

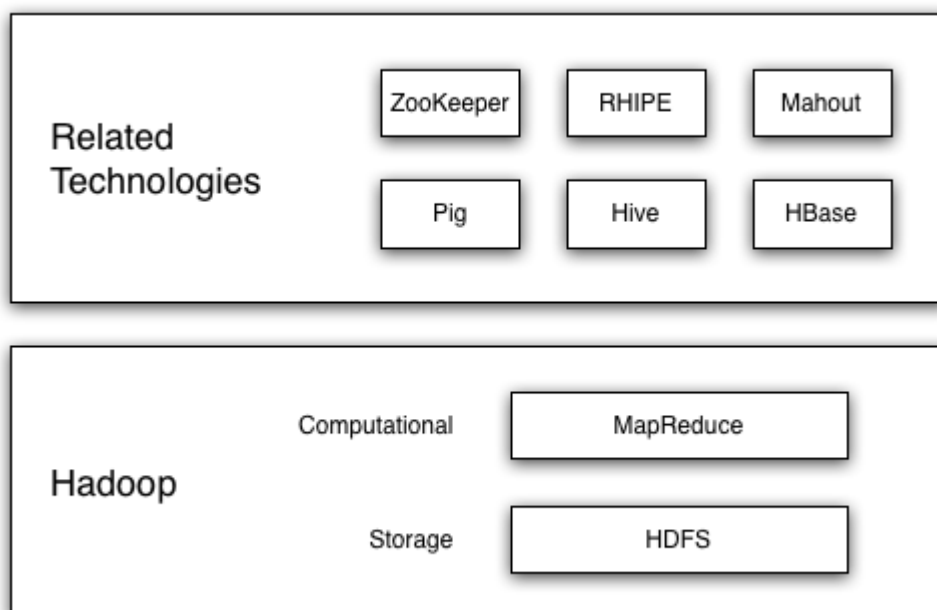


Figure 1.8 Hadoop and related technologies

We'll cover the physical distribution of these technologies in the next section, and specifically Figure 1.12 shows that distribution. How about a introductory paragraph on each one of these technologies?

Pig and Hive

MapReduce is not for the faint of heart, and the goals of many of these Hadoop-related projects is to increase the accessibility of Hadoop to programmers and non-programmers. Two tools that greatly simplify working with MapReduce are Pig and Hive, contributed by Yahoo and Facebook respectively. They both started as research projects that eventually became tenured Apache projects due to their popularity within their respective organizations, and in the community at large.

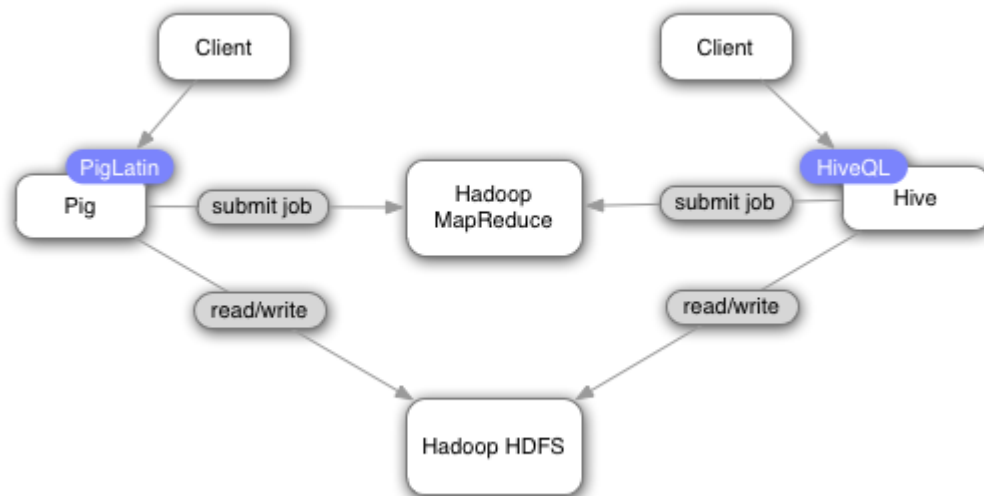


Figure 1.9 Pig and Hive High Level Architecture and Flow

Pig and Hive provide abstractions on top of Hadoop using their own language syntax. Pig uses a data flow language called PigLatin, and Hive caters to the database crowd with SQL-92 constructs along with Hive-specific extensions. Pig is more suited to ETL-like activities due to its pipeline multi-flow architecture. Hive is more like a data warehouse tool due to its syntax, and its metastore which stores schema information about data in Hive. This book will cover Pig and Hive in Chapter 10, and provide a number of techniques to work effectively with them.

HBase and Zookeeper

HBase addresses a different gap in the Hadoop ecosystem; MapReduce, Pig and Hive are all batch-based programming models, whereas HBase is a distributed real-time key/value data store. It is modelled after the Google BigTable white paper. In some ways HBase is conceptually similar to databases, supporting the notion of tables and columns along with a strong consistency model. Most similarities stop there however, since it employs column-oriented storage layout, and has no notion of relations or schema's. HBase leverages ZooKeeper, a High Availability co-ordination service, to keep track of the HBase hosts and for other co-ordination purposes. While ZooKeeper doesn't use any Hadoop technologies itself, it's rapidly gaining traction not only in HBase but in HDFS and MapReduce as a way to support High Availability coordination-related functions.

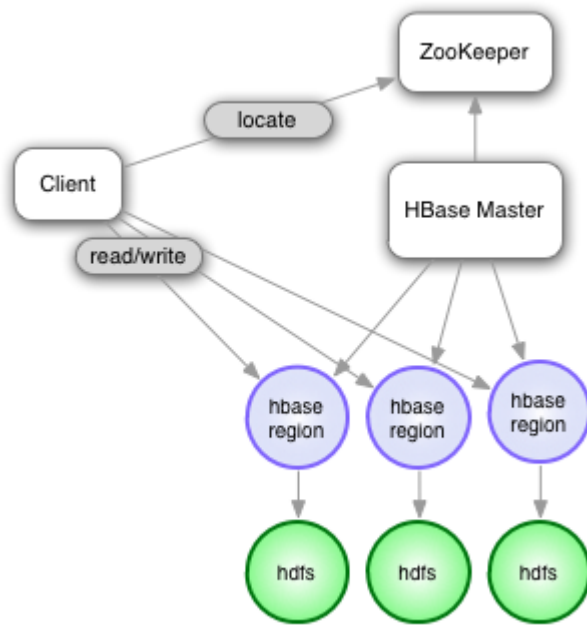


Figure 1.10 HBase High Level Architecture

RHIPE and Mahout

RHIPE and Mahout can help us with data mining, statistical analysis and machine learning activities when working with Hadoop. RHIPE stands for R and Hadoop Integrated Processing Environment. R is a popular statistical programming language, and RHIPE is the fusion of R and Hadoop, which combined allow distributed data analysis. Apache Mahout is a project that enables distributed machine learning on Hadoop, allowing the application of supervised and unsupervised learning algorithms over our data.

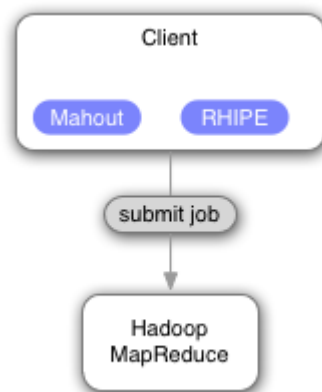


Figure 1.11 Mahout and RHIPE client Architectures

Let's look at how we would distribute these components across hosts in our

environments.

1.1.3 Physical Architecture

The physical architecture lays out where various components are installed and executed. Figure 1.12 shows a typical example of a Hadoop physical architecture involving Hadoop, HBase, ZooKeeper, Pig and Hive, and how they would be distributed across physical hosts. With the current Hadoop stack there is a single master, which we've called "Primary Master", which runs the master HDFS, MapReduce and HBase daemons. Running these masters on the same host is sufficient for small-to-medium Hadoop clusters, but with larger clusters it would be worth considering splitting them onto separate hosts due to the increased load they put on a single server. The "Secondary Master" runs the SecondaryNameNode, and all remaining masters run ZooKeeper, which also runs on the primary and secondary masters. Bear in mind that ZooKeeper requires an odd-numbered quorum⁴, so the recommended practice is to have at least 3 of them in any reasonably-sized cluster.

Footnote 4 A quorum is a high-availability concept which represents the minimum number of members required to be able for a system to still remain online and functioning.

The slave hosts all run the same components, which are the HDFS, MapReduce and HBase slave daemons. A reasonable question may be, why not split them further onto separate hosts? If you were to do this, you would lose out on data locality, the ability to read from local disk, which is a key distributed system property of both the MapReduce and HBase slave daemons.

Client hosts run application code in conjunction with Pig, Hive, RHIPE and Mahout installations. All these Hadoop ecosystem components are client-side libraries that don't need to be installed on your actual Hadoop cluster.

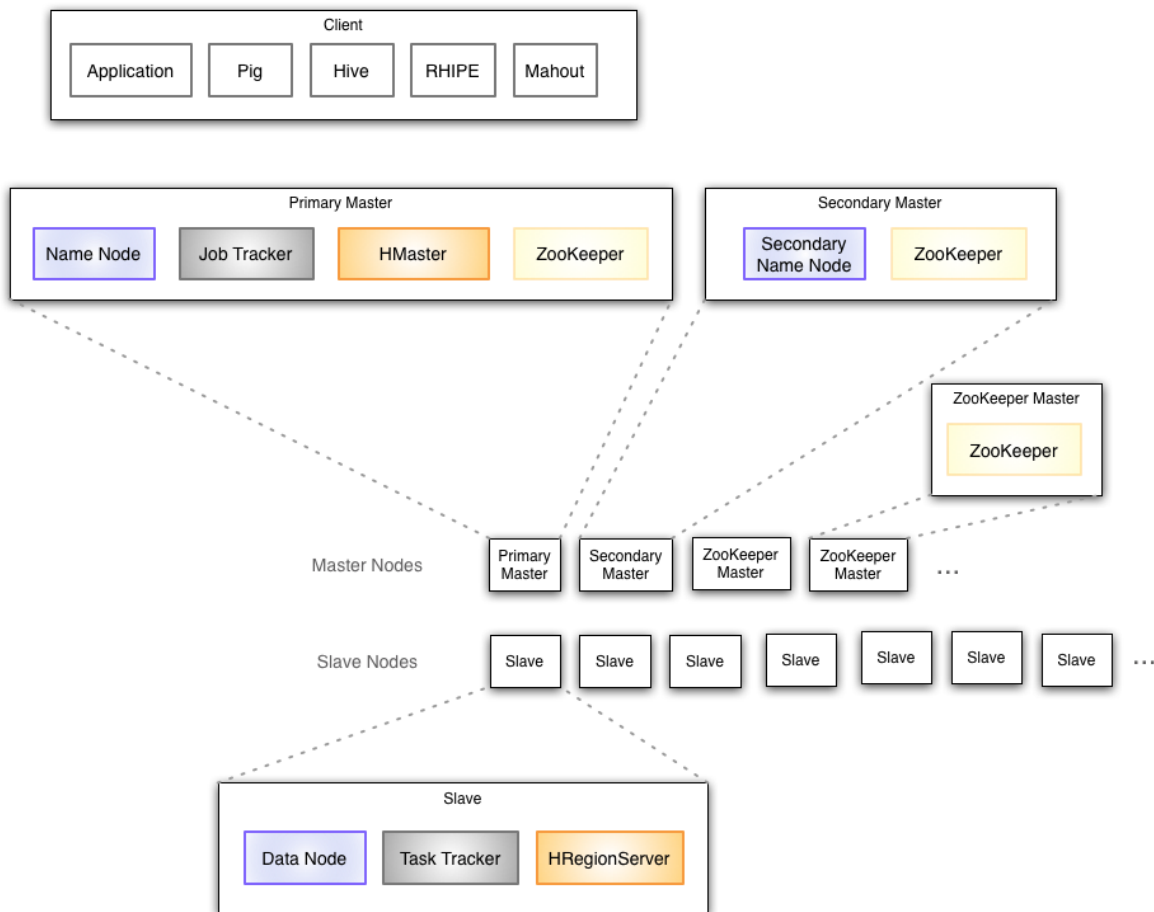


Figure 1.12 Example of Hadoop Physical Architecture

In the case of Hadoop we need to extend our discussion of physical architecture to include CPU, RAM, disk and network, since they all have an impact on the throughput and performance of your cluster.

The term "commodity hardware" is often used when describing Hadoop hardware requirements. While it is true that Hadoop can run on any old servers you can dig up, you still want your cluster to perform well, and not to swamp Operations with diagnosing and fixing hardware issues. Therefore commodity really refers to mid-level rack servers, with dual-sockets, as much error-correcting RAM as is affordable, and SATA drives optimized for RAID storage. Using RAID however isn't necessary, since HDFS already has replication and error-checking built-in.

From a network topology perspective with regards to switches and firewalls, all the master and slave nodes must be able to open connections to each other. For small clusters ideally all the hosts would run 1G network cards connected to a single, good-quality switch. For larger clusters look at 10G top-of-rack switches that have at least multiple 1G uplinks to dual central switches. Client nodes also

need to be able to talk to all the master and slave nodes, but if necessary that access can be from behind a firewall that only permits connection establishment from the client-side.

Now that we understand the functional and architectural aspects of Hadoop, let's look at who is using Hadoop, and in what capacity.

1.1.4 Who is using Hadoop?

Hadoop has a high level of penetration in high-tech companies, and is starting to make inroads across a broad range of sectors including the enterprise (Booz Allen Hamilton, JP Morgan), government (NSA) and health care.

Facebook use Hadoop, Hive and HBase for both data warehousing and for real-time application-serving ⁵. Their data warehousing clusters are petabytes in size with thousands of nodes, and they have separate HBase-driven real-time clusters that are used for messaging and for real-time analytics.

Footnote 5 http://www.facebook.com/note.php?note_id=468211193919

Twitter use Hadoop, Pig and HBase for data analysis, visualization, social graph analysis and machine learning. They LZO-compress all their data, and use Protocol Buffers for serialization purposes, all of which is geared to optimize the use of their storage and computing resources.

Yahoo! use Hadoop for data analytics, machine learning, search ranking, email anti-spam, ad optimization, ETL and more. Combined they have over 40,000 servers running Hadoop with 170 PB of storage.

eBay, Samsung, Rackspace, JP Morgan, Groupon, LinkedIn, AOL, Last.fm and StumbleUpon are just a few organizations that are also heavily invested in Hadoop. Microsoft are also starting to work with Hortonworks to ensure that Hadoop works on its platform.

Google in their MapReduce paper indicate that they use ⁶ their version of MapReduce to create their web index from their crawl data. They also highlight applications of MapReduce to include activities such as a distributed grep, URL access frequency (from log data) and term-vector algorithm which determines popular keywords for a host.

Footnote 6 In 2010 Google moved to a real-time indexing system called Caffeine <http://googleblog.blogspot.com/2010/06/our-new-search-index-caffeine.html>.

The organizations that use Hadoop grow by the day, and if you work at a Fortune 500 company you almost certainly have a Hadoop cluster that's being used in some capacity. It is clear that as Hadoop continues to mature its adoption will

only continue to skyrocket.

As with all technologies a key part to being able to effectively work with them is to understand their shortcomings, and design/architect your solutions to mitigate against them as much as possible.

1.1.5 Hadoop Limitations

Common areas identified as weaknesses across HDFS and MapReduce are availability and security. All of their master processes are single points of failure, although it should be noted that there is active work on High Availability (HA) versions in the community. Security is another area that has its wrinkles, and again another area that is receiving focus.

HIGH AVAILABILITY

Up until the Hadoop 0.23 release, HDFS and MapReduce have both employed single-master models, resulting in single points of failure. Hadoop 0.23 will eventually bring both NameNode and JobTracker High Availability (HA) support. The 0.23 NameNode HA design requires shared storage for NameNode metadata, which may require expensive HA storage (this may change in the future by moving metadata into ZooKeeper). It supports a single standby NameNode, preferably on a separate rack. Both architectures also require ZooKeeper for coordination purposes.

SECURITY

Hadoop does offer a security model, but by default it is disabled. With the security model disabled, the only security feature that exists in Hadoop is HDFS file and directory-level ownership and permissions. However it is simple for malicious users to subvert and assume other user's identities. By default, all other Hadoop services are wide open, allowing any user to perform any kind of operation, such as killing another users's MapReduce jobs.

Hadoop can be configured to run with Kerberos, a network authentication protocol, which requires requires Hadoop daemons to authenticate clients, both user and other Hadoop components. Kerberos can be integrated with an organization's existing Active Directory, and therefore offer a single sign-on experience for users. However the web and proxy Hadoop interfaces require custom code to be written to integrated with Kerberos. Finally, and most importantly for the government sector, there is no storage or wire-level encryption in Hadoop. Overall, configuring Hadoop to be secure has a high pain point due to its complexity.

Let's examine the limitations of some of the individual systems.

HDFS

The weakness of HDFS are mainly around the lack of High Availability, its inefficient handling of small files, and lack of transparent compression. It is not designed to work well with random reads over small files, due to its optimization for sustained throughput. We are waiting for append support for files, which is nearing production readiness.

MAPREDUCE

MapReduce is a batch-based architecture, and therefore does not lend itself to use cases needing real-time data access. Tasks that require global synchronization or sharing of mutable data aren't a good fit for MapReduce, as it is a shared-nothing architecture, which can pose challenges for some algorithms.

ECOSYSTEM VERSION COMPATIBILITIES

There can be version dependency challenges with running Hadoop. For example, at the time of writing HBase only works with a version of Hadoop that's not verified as production-ready ([Subversion branch-0.20-append branch](#)), due to its HDFS sync requirements (sync is a mechanism that ensures that all writes to a stream have been written to disk across all replica's). However it should be noted that the 0.20.205 and 0.23 releases will both work with HBase. There are also challenges with Hive and Hadoop, where Hive may need to be re-compiled to work with versions of Hadoop other than the one it was built against. Similarly Pig has had compatibility issues too, for example the 0.8 Pig release didn't work with Hadoop 0.20.203, requiring manual intervention to make them work together. This is one of the advantages of using a Hadoop distribution other than Apache, as these compatibility problems have been fixed. One development worth tracking is the creation of [BigTop](#) , currently an Apache incubator project, which is a contribution from Cloudera to open-source their automated build, and compliance system. It includes all the major Hadoop ecosystem components, and runs a number of integration tests to ensure they all work in conjunction with each other.

Looking at Hadoop alternatives will help us understand if any of its shortcomings are addressed by other technologies.

1.1.6 Hadoop Alternatives

Prior to Hadoop there were several options available if you needed parallelized storage and/or computation. The earliest examples would be supercomputers, which take parallel processing on extreme levels. Similarly grid computing has existed in various forms since the 90's, and on larger scales have yielded projects such as SETI@home, a distributed computing project spearheaded by U.C. Berkley.

Parallel databases such as Vertica, IBM Netezza and Oracle Exadata can run on shared-nothing clusters and support working with structured data by supporting standard database features such as tables, relationships and SQL. Data is split across nodes in a cluster, and a distributed query optimizer is used to segment work and issue to cluster nodes to be executed in parallel. In a joint paper published by Brown University and Microsoft, they compared parallel databases to Hadoop⁷ and came to the conclusion that parallel databases offered significant performance advantages over Hadoop MapReduce. Google refuted the findings of this study in their own paper, claiming that the comparison was based on flawed assumptions about MapReduce⁸.

Footnote 7 A comparison of parallel databases and MapReduce
<http://database.cs.brown.edu/sigmod09/benchmarks-sigmod09.pdf>

Footnote 8 MapReduce: A Flexible Data Processing Tool
<http://cacm.acm.org/magazines/2010/1/55744-mapreduce-a-flexible-data-processing-tool/fulltext>

Hadoop is frequently used as a cost-effective Data Warehouse platform. However when dealing with sub-terabyte data sets traditional Data Warehouse tools from vendors such as Oracle and IBM can be cost-competitive.

From a framework and architecture standpoint there are few competitors to MapReduce. Microsoft has a research project titled Dryad⁹ which looks to be very similar to MapReduce. They have another project called Daytona¹⁰ which provides a MapReduce mechanism for Windows Azure, their cloud technology. MapReduce as a paradigm is starting to be supported in NoSQL data stores such as Riak and MongoDB. The last few years has seen research into running MapReduce on Graphics Processor Units (GPU's)¹¹.

Footnote 9 <http://research.microsoft.com/en-us/projects/dryad/>

Footnote 10 <http://research.microsoft.com/en-us/downloads/cecba376-3d3f-4eaf-bf01-20983857c2b1/>

Footnote 11 Mars: A MapReduce Framework on Graphics Processor
http://www.cse.ust.hk/gpuqp/Mars_tr.pdf

1.1.7 Where to turn to for Help

There will no doubt be many questions that you will have when working with Hadoop. Luckily between the wiki's and vibrant user community your needs should be well covered.

- The main wiki is at <http://wiki.apache.org/hadoop/>, and contains useful presentations, setup instructions and troubleshooting instructions.
- The Hadoop Common, HDFS and MapReduce mailing lists can all be found on http://hadoop.apache.org/mailling_lists.html.
- Search Hadoop is a useful website that indexes all the Hadoop and ecosystem projects and provides full-text search capabilities. <http://search-hadoop.com/>

Blogs and Twitter

There are many useful blogs that should be subscribed to, to keep on top of current events in Hadoop. We've included a selection of the author's favorites.

- Cloudera are a prolific writer of practical applications of Hadoop. <http://www.cloudera.com/blog/>
- The Hortonworks blog is worth reading, as they discuss application and future Hadoop roadmap items. <http://hortonworks.com/blog/>
- Michael Noll is one of the first bloggers to provide detailed setup instructions for Hadoop, and continues to write about real-life challenges and uses of Hadoop. <http://www.michael-noll.com/blog/>
- There are a plethora of active Hadoop twitter users that should be followed, including Arun Murthy (@acmurthy), Tom White (@tom_e_white), Eric Sammer (@esammer), Doug Cutting (@cutting) and Todd Lipcon (@tlipcon). The Hadoop project itself tweets on @hadoop.

We have looked at Hadoop's architecture, weaknesses and technology alternatives. Our next step is to use the Cloudera Distribution for Hadoop (CDH) to get Hadoop up and running on your system, and throughout the rest of the book. CDH was picked due to its simple installation and operation.

1.2 Running Hadoop

The end goal of this section is to run a MapReduce job on your host. To get there we'll need to install Cloudera's Hadoop distribution, run through some command-line and configuration steps and then write some MapReduce code.

1.2.1 Downloading and Installing Hadoop

CDH includes OS-native installation packages for top-level Linux distributions such as RedHat, Debian and SUSE and their derivatives. There are also tarball and Virtual Machine images with CDH pre-installed. All the available options can be viewed at <http://www.cloudera.com/hadoop/>.

We're going to show instructions for installation on a RedHat-based Linux system (in our case we're using CentOS). Installation instructions for both the CDH tarball as well as the Apache Hadoop tarball are included in Appendix A.

RedHat uses packages called RPM's for installation, and yum as a package installer that can also fetch RPM's from remote yum repositories. Cloudera hosts their own yum repository containing Hadoop RPM's which we'll use for installation.

We're going to follow the pseudo-distributed installation instructions.¹² A pseudo-distributed setup is one where all the Hadoop components are running on a single host. The first thing you need to do is download and install the "bootstrap" RPM, which will update your local yum configuration to include Cloudera's remote yum repository.

Footnote 12

<https://ccp.cloudera.com/display/CDHDOC/Installing+CDH3+on+a+Single+Linux+Node+in+Pseudo-distributed+Mode>

```
$ wget http://archive.cloudera.com/redhat/cdh/cdh3-
repository-1.0-1.noarch.rpm
$ rpm -ivh cdh3-repository-1.0-1.noarch.rpm
```

Next we import Cloudera's RPM signing key so that yum can verify the integrity of the RPM's that it downloads.

```
$ rpm --import \
http://archive.cloudera.com/redhat/cdh/RPM-GPG-KEY-cloudera
```

Our last installation step is to install the pseudo-distributed RPM package which has dependencies on all the other core Hadoop RPM's. We're also installing Pig, Hive and Snappy (which is contained in the Hadoop native package) since we'll be using them in this book.

```
$ yum install hadoop-0.20-conf-pseudo hadoop-0.20-native \  
hadoop-pig hadoop-hive
```

Our installation of Hadoop is complete! For this book we will also be working with Oozie, HBase and other projects, but we'll defer to their respective sections for instructions.

With the basics installed we should look at how we configure Hadoop, and go over some basic commands to start and stop our cluster.

1.2.2 Hadoop Configuration

Having completed the installation instructions in the previous section, our software is ready for use without editing any configuration files. However knowing the basics of Hadoop's configuration is useful so we'll briefly touch upon it here. In CDH the Hadoop configs are contained under `/etc/hadoop/conf`. There are separate configuration files for different Hadoop components, and it's worth providing a quick overview of them.

Table 1.1 Hadoop configuration files

| | |
|-----------------|--|
| hadoop-env.sh | Environment-specific settings go here. If a current JDK is not in the system path then you want to come here to configure your JAVA_HOME. You can also specify JVM options for various Hadoop components here. Customizing directory locations such as the log directory, the locations of the masters and slaves files are also performed here, although by default you shouldn't have to do any of the above in a CDH setup. |
| core-site.xml | System-level Hadoop configuration items such as the HDFS URL, the Hadoop temporary directory, and script locations for rack-aware Hadoop clusters. Settings in this file override the settings in core-default.xml. The default settings can be see at http://hadoop.apache.org/common/docs/current/core-default.html . |
| hdfs-site.xml | Contains HDFS settings such as the default file replication count, the block size and whether permissions are enforced. To view the default settings you can look at http://hadoop.apache.org/common/docs/current/hdfs-default.html . Settings in this file override the settings in hdfs-default.xml. |
| mapred-site.xml | HDFS settings such as the default number of Reduce tasks, default min/max task memory sizes and speculative execution are set here. To view the default settings you can look at http://hadoop.apache.org/common/docs/current/mapred-default.html . Settings in this file override the settings in mapred-default.xml. |
| masters | Contains a list of hosts that are Hadoop "masters". This name is misleading and should really have been called "secondary-masters". When you start Hadoop it will launch the NameNode and JobTracker on the local host that you issued the start command, and then ssh to all the nodes in this file to launch the SecondaryNameNode. |
| slaves | Contains a list of hosts that are Hadoop slaves. When you start Hadoop it will ssh to each host in this file and launch the DataNode and TaskTracker dameons. |

Three of the files identified above (those with "site" in their filenames) have equivalent files that contain Hadoop default settings. In vanilla Apache the default files are co-located in the same directory as the "site" files. CDH chose a different

approach and moved them into the CDH JAR files, most likely to prevent the user from accidentally modifying a default file.

These "site" XML files will grow as you start customizing your Hadoop cluster, and it can quickly become challenging to keep track of what changes you have made, and how they relate to the default configuration values. To help with this the authors have written some code ¹³ which will compare the default and site files and indicate what properties have changed, as well as let you know about properties that you may have mis-spelled. Some example output of our utility is included below showing a few of the difference between the CDH `core-default.xml` and `core-site.xml` files.

Footnote 13 <https://github.com/alexholmes/hadoop-book>

```
Default file:  core-default.xml
Site file:    core-site.xml

      Name      Final  Default      Site
      fs.default.name  false file:///  dfs://localhost:8020
      fs.har.impl.disable.cache  false   true      null
hadoop.proxyuser.oozie.groups    -      -          *
```

The Cloudera team have researched ¹⁴ more advanced techniques using static and dynamic methods to determine what options were supported in Hadoop, and discrepancies between application and Hadoop configurations.

Footnote 14

<http://www.cloudera.com/blog/2011/08/automatically-documenting-apache-hadoop-configuration/>

1.2.3 Basic CLI Commands

Let's rattle through the essentials needed to get you up and running. First up is starting your cluster. You'll need `sudo` access for your user to run this command (as it launches the Hadoop services via `init.d` scripts).

```
$ for svc in /etc/init.d/hadoop-0.20-*; do sudo $svc start; done
```

All the daemon log files are written under `/var/log/hadoop`. For example, the NameNode file is written to `hadoop-hadoop-namenode-cdh.log`, and this can be a useful file to look at if you have problems bringing up HDFS. You can test that things are up and

running in a couple of ways. First try issuing a command to list the files in the root directory in HDFS.

```
$ hadoop fs -ls /
```

NOTE

Pathname expansions

You wouldn't think that the simple Hadoop file system command to list directory contents would have a quirk, but it does, and it's one that has bitten many a user, including the authors on numerous occasions. In bash and other shells it is normal to affix the "*" wildcard to file system commands, and for the shell to expand that prior to running a program. You would therefore (incorrectly) assume that the command `hadoop fs -ls /tmp/*` would work. But if you run this, your shell will expand the path based on the contents of "/tmp" on your local filesystem, pass these filenames into Hadoop, at which point Hadoop attempts to delete files in HDFS which actually reside on your local system! The workaround is to prevent path expansion for occurring by enclosing the path in double-quotes, therefore this would become `hadoop fs -ls "/tmp/*"`.

If this works then HDFS is up and running. To make sure MapReduce is up and running we'll run a quick command to see what jobs are running.

```
$ hadoop job -list
0 jobs currently running
JobId      State      StartTime      UserName      Priority      SchedulingInfo
```

Good, things seem to be in order. If you're curious about what commands you can issue from the command line then take a look at http://hadoop.apache.org/common/docs/current/file_system_shell.html for HDFS commands, http://hadoop.apache.org/common/docs/current/commands_manual.html#job for MapReduce job commands, and http://hadoop.apache.org/common/docs/current/commands_manual.html for an overall list of commands including administrative commands.

Finally, stopping your cluster is similar to starting.

```
$ for svc in /etc/init.d/hadoop-0.20-*; do sudo $svc stop; done
```

With these essentials under our belt our next step is to write a MapReduce job (don't worry, it's not word count) which we can run in our new cluster.

1.2.4 Running a MapReduce Job

Let's say you wanted to build an inverted index. MapReduce would be a good choice for this task as it can work on the creation of indexes in parallel, and as a result is a common MapReduce use case. Your input is a number of text files, and your output is a list of tuples, where each tuple is a word and a list of files that contain the word. Using standard processing techniques this would require you to find a mechanism to join all the words together. A naive approach would be to perform this join in memory, but you may run out of memory if you have large numbers of unique keys. You could use an intermediary datastore such as a database, but this would be inefficient.

A better approach would be to tokenize each line and produce an intermediary file containing a word per line. Each of these intermediary files can then be sorted. The final step would be to open all the sorted intermediary files and call a function for each unique word. This is essentially what MapReduce is doing, albeit in a distributed fashion.

Figure 1.13 walks us through an example of what a very simple inverted index MapReduce. In our example we have two files containing a line each. We have two Mappers, and each mapper is called once for each line in their respective input file. Our Mapper simply splits the line into distinct words, and outputs a key/value pair for each word in the line, where the key is the word, and the value is the name of the file.

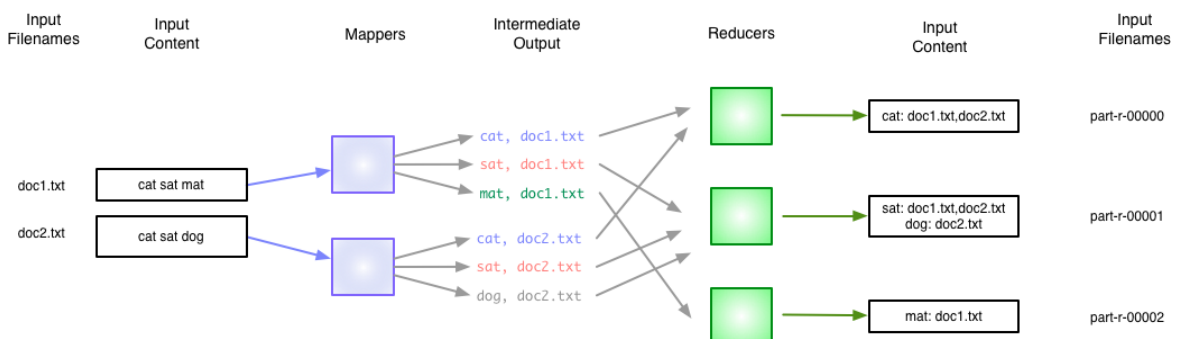


Figure 1.13

MapReduce now performs some partitioning, where it examines the key that the Mapper emitted and determines what Reducer it should go to. The goal here is that the same key emitted across all the Mappers should go to the same Reducer. So in our case, the word "sat" will always go to the same Reducer.

Once the Reducers have all the keys and values from the Mappers they are sorted by key. Then the user's Reducer function is called one per unique key, along with a list of all the values emitted across all the Mappers for that key. In the case of "cat", both Mappers emitted it as a key with their respective document ID's (or filenames in our case), so we get a list containing the two values for both Mappers. Our Reducer would simply collect together all the values and produce a single line of output for each key containing the list of filenames.

We'll start with defining our Mapper. Our Reducers need to be able to generate a line for each word in our input, so our Map output key should be each word in the input files, so that MapReduce can join them all together. The value for each key will be the containing filename, which is our document ID.

```
public static class Map
    extends Mapper<LongWritable, Text, Text, Text> {           ❶

    private Text documentId;                                  ❷
    private Text word = new Text();                          ❸

    @Override
    protected void setup(Context context) {                  ❹
        String filename =                                   ❺
            ((FileSplit) context.getInputSplit()).getPath().getName();
        documentId = new Text(filename);
    }

    @Override
    protected void map(LongWritable key, Text value,          ❻
        Context context)
        throws IOException, InterruptedException {
        for (String token :
            StringUtils.split(value.toString())) {           ❼
            word.set(token);                                  ❽
            context.write(word, documentId);
        }
    }
}
```

- ❶ When we extend the MapReduce Mapper class we specify the key/value types for our inputs and outputs. We're using the MapReduce default InputFormat for our

job, which supplies keys as byte offsets into the input file, and values as each line in the file. Our Map emits Text key/value pairs.

- ② A Text object to store the document Id (filename) for our input.
- ③ To cut down on object creation we create a single Text object which we will reuse.
- ④ This method is called once at the start of the Map and prior to the map method being called. We're using this opportunity to store the input filename for this Map.
- ⑤ Extract the filename from the context.
- ⑥ This map method is called once per input line. Map tasks are run in parallel over subsets of the input files.
- ⑦ Our value contains an entire line from our file. We tokenize the line using StringUtils (which is far faster than using String.split).
- ⑧ For each word our Map outputs the word as the key and the document ID as the value.

The goal of our Reducer is to create an output line for each word, and a list of the document ID's that the word appears in. The MapReduce framework will take care of calling our Reducer once per unique key outputted by the Mappers, along with a list of document ID's. All we need to do in our Reducer is combine all the document ID's together and output them once in the reducer.

```
public static class Reduce
    extends Reducer<Text, Text, Text, Text> {
    private Text docIds = new Text();
    public void reduce(Text key, Iterable<Text> values,
        Context context)
        throws IOException, InterruptedException {
        HashSet<Text> uniqueDocIds = new HashSet<Text>();
        for (Text docId : values) {
            uniqueDocIds.add(new Text(docId));
        }
        docIds.set(new Text(StringUtils.join(uniqueDocIds, ",")));
        context.write(key, docIds);
    }
}
```

- ① Much like our Map class we need to specify both the input and output key/value classes when we define our Reducer.
- ② The reduce method is called once per unique Map output key. The Iterable allows you to iterate over all the values that were emitted for the given key.
- ③ Keep a set of all the document ID's that we encounter for the key.
- ④ Iterate over all the document ID's for the key.
- ⑤ Add the document ID to our set.
- ⑥ Our Reduce outputs the word, and a CSV-separated list of document ID's that contained the word.

The last step in our code is to write the controlling code which will set all the necessary properties to configure our MapReduce job to run. We need to let the framework know what classes should be used for the Map and Reduce functions, and also let it know where our input and output data is located. By default MapReduce assumes we're working with text; if we were working with more complex text structures, or altogether different data storage technologies then we would need to tell MapReduce about how it should read and write from these data sources and sinks.

```

public static void main(String... args) throws Exception {

    runJob(
        Arrays.copyOfRange(args, 0, args.length - 1),
        args[args.length - 1]);
}

public static void runJob(String[] input, String output)
    throws Exception {
    Configuration conf = new Configuration();

    Job job = new Job(conf);
    job.setJarByClass(InvertedIndexMapReduce.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);

    Path outputPath = new Path(output);

    FileInputFormat.setInputPaths(job, StringUtils.join(input, ","));

    FileOutputFormat.setOutputPath(job, outputPath);

    outputPath.getFileSystem(conf).delete(outputPath, true);

    job.waitForCompletion(true);
}

```

- ❶ Our input is 1 or more files, so create a sub-array from our input arguments, excluding the last item of the array which is the MapReduce job output directory.
- ❷ The Configuration container for your job configs. Anything that is set here is available to your Map and Reduce classes.
- ❸ The Job class setJarByClass method determines the JAR that contains the class that's passed-in, which beneath the scenes is copied by Hadoop into the cluster and subsequently set in the Task's classpath so that your Map/Reduce classes are available to the Task.

❹

- 5 Set the Reduce class that should be used for the job.
- 6 If the Map output key/value types differ from the input types you must tell Hadoop what they are. In our case our Map will output each word and file as the key/value pairs, and both are Text objects.
- 7 Set the Map output value class.
- 8 Set the HDFS input files for our job. Hadoop expects multiple input files to be separated with commas.
- 9 Set the HDFS output directory for the job.
- 10 Delete the existing HDFS output directory if it exists. If you don't do this and the directory already exists the job will fail.
- 11 Tell the JobTracker to run the job and block until the job has completed.

Let's see how our code works. We're going to work with two simple files. First we need to copy the files into HDFS.

```
$ hadoop fs -put \
  test-data/ch1/file1.txt test-data/ch1/file2.txt /
$ hadoop fs -cat /file1.txt
cat sat mat
$ hadoop fs -cat /file2.txt
cat sat dog
```

- 1 Copy file1.txt and file2.txt into HDFS. Note that we had to split this command across two lines, so we use the "\" character to escape the newline.
- 2 Dump the contents of the HDFS file /file1.txt to the console.
- 3 Dump the contents of the HDFS file /file2.txt to the console.

Now we run our MapReduce code. We'll use a shell script to run our code, supplying the two input files as arguments, along with the job output directory.

```
$ bin/run.sh com.manning.hip.ch1.InvertedIndexMapReduce \
  /file1.txt /file2.txt output
```

Once our job completes we can examine HDFS for the job output files, and also view their contents.

```
$ hadoop fs -ls output/
Found 3 items
output/_SUCCESS
output/_logs
output/part-r-00000

$ hadoop fs -cat output/part-r-00000
```

```

cat    file2.txt,file1.txt
dog    file2.txt
mat    file1.txt
sat    file2.txt,file1.txt

```

You may be curious about where the actual Map and Reduce log files go. For that we need to know the job's ID, which will take us to the logs directory in the local file system. When you run your job from the command line part of the output is the job ID, as follows.

```

...
INFO mapred.JobClient: Running job: job_201110271152_0001
...

```

With this ID in hand we can navigate to the directory on our local filesystem which contains a directory for each map and reduce task, which can be differentiated by the "m" and "r" in the directory names.

```

$ pwd
/var/log/hadoop-0.20/userlogs/job_201110271152_0001
$ ls -l
attempt_201110271152_0001_m_000000_0
attempt_201110271152_0001_m_000001_0
attempt_201110271152_0001_m_000002_0
attempt_201110271152_0001_m_000003_0
attempt_201110271152_0001_r_000000_0

```

Within each of those directories there are three files, corresponding to standard out, standard error and the system log (output from both the infrastructure task code, as well as any of your own log4j logging).

```

$ ls attempt_201110271152_0001_m_000000_0
stderr  stdout  syslog

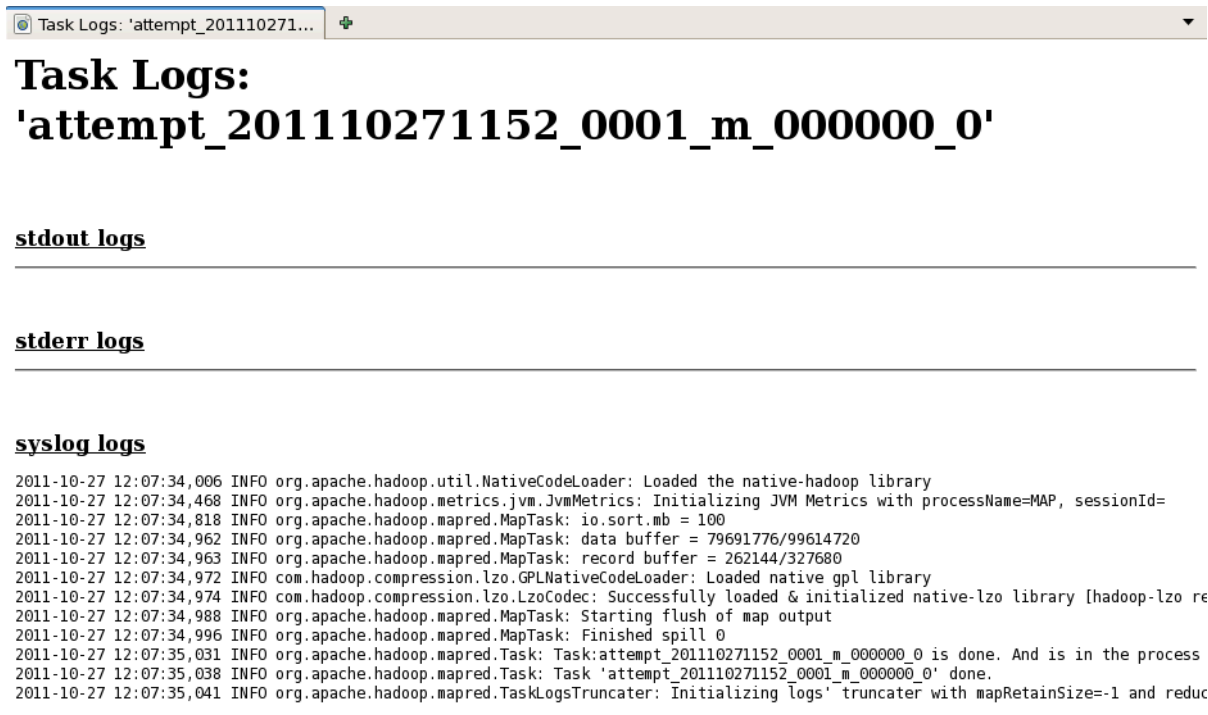
```

Remember that in our pseudo-distributed setup everything's running on our local host, so it's easy to see everything in one place. On a true distributed cluster these logs will be local to the remote TaskTracker nodes, which can make it harder

to get to them. This is where the JobTracker and TaskTracker UI step in to provide easy access to the logs. Figures 1.14 and 1.15 show some screenshots of the JobTracker summary page for our job, and the TaskTracker UI for one of the Map tasks. In CDH the JobTracker UI can be accessed at <http://localhost:50030/jobtracker.jsp>.



Figure 1.14 The Hadoop JobTracker User Interface



The screenshot shows a web browser window with the title 'Task Logs: 'attempt_201110271152_0001_m_000000_0''. Below the title, there are three sections: 'stdout logs', 'stderr logs', and 'syslog logs'. The 'syslog logs' section contains a list of log entries with timestamps and messages from various Hadoop components.

```

Task Logs: 'attempt_201110271152_0001_m_000000_0'

stdout logs

stderr logs

syslog logs
2011-10-27 12:07:34,006 INFO org.apache.hadoop.util.NativeCodeLoader: Loaded the native-hadoop library
2011-10-27 12:07:34,468 INFO org.apache.hadoop.metrics.jvm.JvmMetrics: Initializing JVM Metrics with processName=MAP, sessionId=
2011-10-27 12:07:34,818 INFO org.apache.hadoop.mapred.MapTask: io.sort.mb = 100
2011-10-27 12:07:34,962 INFO org.apache.hadoop.mapred.MapTask: data buffer = 79691776/99614720
2011-10-27 12:07:34,963 INFO org.apache.hadoop.mapred.MapTask: record buffer = 262144/327680
2011-10-27 12:07:34,972 INFO com.hadoop.compression.lzo.GPLNativeCodeLoader: Loaded native gpl library
2011-10-27 12:07:34,974 INFO com.hadoop.compression.lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo re
2011-10-27 12:07:34,988 INFO org.apache.hadoop.mapred.MapTask: Starting flush of map output
2011-10-27 12:07:34,996 INFO org.apache.hadoop.mapred.MapTask: Finished spill 0
2011-10-27 12:07:35,031 INFO org.apache.hadoop.mapred.Task: Task:attempt_201110271152_0001_m_000000_0 is done. And is in the process
2011-10-27 12:07:35,038 INFO org.apache.hadoop.mapred.Task: Task 'attempt_201110271152_0001_m_000000_0' done.
2011-10-27 12:07:35,041 INFO org.apache.hadoop.mapred.TaskLogsTruncater: Initializing 'logs' truncater with mapRetainSize=-1 and reduc

```

Figure 1.15 The Hadoop TaskTracker User Interface

This completes our whirlwind tour of running Hadoop!

1.3 Summary

Hadoop is a distributed system designed to process, generate and store large data sets. Its MapReduce implementation provides us with a fault-tolerant mechanism for large-scale data analysis. It excels at working with heterogeneous structured and unstructured data sources at scale.

We examined Hadoop from a functional and physical architectural standpoints, and also compared it to other distributed systems that offer similar capabilities. We also installed Hadoop and ran a MapReduce job.

The remainder of this book is dedicated to providing real-world techniques to solve common problems encountered when working with Hadoop. We cover a broad spectrum of subject areas starting with HDFS and MapReduce, Pig, Hive and then looking at data analysis techniques with technologies such as Mahout and RHIPE.

Our first stop on our journey is a look at how to bring data into (and out of) Hadoop, so without further ado, let's get started.