# Beginning Apache Pig

Big Data Processing Made Easy

Balaswamy Vaddeman

Apress®

# Beginning Apache Pig

## Big Data Processing Made Easy

Balaswamy Vaddeman

Apress®

***Beginning Apache Pig: Big Data Processing Made Easy***

Balaswamy Vaddeman
Hyderabad, Andhra Pradesh, India

ISBN-13 (pbk): 978-1-4842-2336-9 ISBN-13 (electronic): 978-1-4842-2337-6

DOI 10.1007/978-1-4842-2337-6

Library of Congress Control Number: 2016961514



Managing Director: Welmoed Spahr
Lead Editor: Celestin Suresh John
Technical Reviewer: Manoj R. Patil
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Prachi Mehta
Copy Editor: Kim Wimpsett
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper

*The six most important people in my life:*

*The late Kammari Rangaswamy (Teacher)*

*The late Niranjanamma (Mother)*

*Devaiah (Father)*

*Radha (Wife)*

*Sai Nirupam (Son)*

*Nitya Maithreyi (Daughter)*

# Contents at a Glance

# Contents

# About the Author

**Balaswamy Vaddeman** is a thinker, blogger, and serious and self-motivated big data evangelist with 10 years of experience in IT and 5 years of experience in the big data space. His big data experience covers multiple areas such as analytical applications, product development, consulting, training, book reviews, hackathons, and mentoring. He has proven himself while delivering analytical applications in the retail, banking, and finance domains in three aspects (development, administration, and architecture) of Hadoop-related technologies. At a startup company, he developed a Hadoop-based product that was used for delivering analytical applications without writing code.

In 2013 Balaswamy won the Hadoop Hackathon event for Hyderabad conducted by Cloudwick Technologies. Being the top contributor at Stackoverflow.com, he helped countless people on big data topics at multiple web sites such as Stackoverflow.com and Quora.com. With so much passion on big data, he became an independent trainer and consultant so he could train hundreds of people and set up big data teams in several companies.

# About the Technical Reviewer

**Manoj R. Patil** is a big data architect at TatvaSoft, an IT services and consulting firm. He has a bachelor's of engineering degree from COEP in Pune, India. He is a proven and highly skilled business intelligence professional with 17 years of information technology experience. He is a seasoned BI and big data consultant with exposure to all the leading platforms such as Java EE, .NET, LAMP, and so on. In addition to authoring a book on Pentaho and big data, he believes in knowledge sharing, keeps himself busy in corporate training, and is a passionate teacher. He can be reached at on Twitter `@manojrpatil` and at `https://in.linkedin.com/in/manojrpatil` on LinkedIn.

Manoj would like to thank his family, especially his two beautiful daughters, Ayushee and Ananyaa, for their patience during the review process.

# Acknowledgments

Writing a book requires a great team. Fortunately, I had a great team for my first project. I am deeply indebted to them for making this project reality.

I would like to thank the publisher, Apress, for providing this opportunity.

Special thanks to Celestin Suresh John for building confidence in me in the initial stages of this project.

Special thanks to Subha Srikant for your valuable feedback. This project would have not been in this shape without you. In fact, I have learned many things from you that could be useful for my future projects also.

Thank you, Manoj R. Patil, for providing valuable technical feedback. Your contribution added a lot of value to this project.

Thank you, Dinesh Kumar, for your valuable time.

Last but not least, thank you, Prachi Mehta, for your prompt coordination.

CHAPTER 1

# MapReduce and Its Abstractions

In this chapter, you will learn about the technologies that existed before Apache Hadoop, about how Hadoop has addressed the limitations of those technologies, and about the new developments since Hadoop was released.

Data consists of facts collected for analysis. Every business collects data to understand their business and to take action accordingly. In fact, businesses will fall behind their competition if they do not act upon data in a timely manner. Because the number of applications, devices, and users is increasing, data is growing exponentially. Terabytes and petabytes of data have become the norm. Therefore, you need better data management tools for this large amount of data.

Data can be classified into these three types:

- *Small data*: Data is considered small data if it can be measured in gigabytes.
- *Big data*: Big data is characterized by volume, velocity, and variety. *Volume* refers to the size of data, such as terabytes and more. *Velocity* refers to the age of data, such as real-time, near-real-time, and streaming data. *Variety* talks about types of data; there are mainly three types of data: structured, semistructured, and unstructured.
- *Fast data*: Fast data is a type of big data that is useful for the real-time presentation of data. Because of the huge demand for real-time or near-real-time data, fast data is evolving in a separate and unique space.

## Small Data Processing

Many tools and technologies are available for processing small data. You can use languages such as Python, Perl, and Java, and you can use relational database management systems (RDBMSs) such as Oracle, MySQL, and Postgres. You can even use data warehousing tools and extract/transform/load (ETL) tools. In this section, I will discuss how small data processing is done.

**Electronic supplementary material** The online version of this chapter (doi:10.1007/978-1-4842-2337-6_1) contains supplementary material, which is available to authorized users.


B. Vaddeman, *Beginning Apache Pig*, DOI 10.1007/978-1-4842-2337-6_1

Assume you have the following text in a file called `fruits`:

```
Apple, grape
Apple, grape, pear
Apple, orange
```

Let's write a program in a shell script that first filters out the word *pear* and then counts the number of words in the file. Here's the code:

```
cat fruits|tr ',' '\n'|grep -v -i  'pear'|sort -f|uniq  -c -i
```

This code is explained in the following paragraphs.

In this code, `tr` (for "translate" or "transliterate") is a Unix program that takes two inputs and replaces the first set of characters with the second set of characters. In the previous program, the `tr` program replaces each comma (,) with a new line character (`\n`). `grep` is a command used for searching for specific text. So, the previous program performs an inverse search on the word *pear* using the `-v` option and ignores the case using `-i`.

The `sort` command produces data in sorted order. The `-f` option ignores case while sorting.

`uniq` is a Unix program that combines adjacent lines from the input file for reporting purposes. In the previous program, `uniq` takes sorted words from the `sort` command output and generates the word count. The `-c` option is for the count, and the `-i` option is for ignoring case.

The program produces the following output:

```
Apple 3
Grape 2
Orange 1
```

You can divide program functionality into two stages; first is tokenize and filtering, and second is aggregation. Sort is supporting functionality of aggregation. Figure 1-1 shows the program flow.

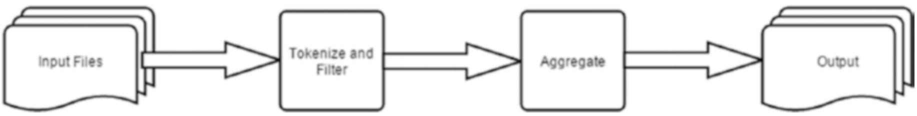


***Figure 1-1.*** *Program flow*

The previous program can be run on a single machine and on small data. Such simple programs can be used to perform simple operations such as searching and sorting on one file at a time. However, writing complex queries involving multiple files and multiple conditions requires better data processing tools. Database management systems (DBMS) and RDBMS technologies were developed to address querying problems with structured data.

## Relational Database Management Systems

RDBMSs were developed based on the relational model founded by E. F. Codd. There are many commercial RDBMS products such as Oracle, SQL Server, and DB2. Many open source RDBMSs such as MySQL, Postgres, and SQLite are also popular. RDBMSs store data in tables, and you can define relations between tables.

Here are some advantages of RDBMSs:

- RDBMS products come with sophisticated query languages that can easily retrieve data from multiple tables with multiple conditions.
- The query language used in RDBMSs is called Structured Query Language (SQL); it provides easy data definition, manipulation, and control.
- RDBMSs also support transactions.
- RDBMSs support low-latency queries so users can access databases interactively, and they are also useful for online transaction processing (OLTP).

RDBMSs have these disadvantages:

- As data is stored in table format, RDBMSs support only structured data.
- You need to define a schema at the time of loading data.
- RDBMSs can scale only to gigabytes of data, and they are mainly designed for frequent updates.

Because the data size in today's organizations has grown exponentially, RDBMSs have not been able to scale with respect to data size. Processing terabytes of data can take days.

Having terabytes of data has become the norm for almost all businesses. And new data types like semistructured and unstructured have arrived. Semistructured data has a partial structure like in web server log files, and it needs to be parsed like Extensible Markup Language (XML) in order to analyze it. Unstructured data does not have any structure; this includes images, videos, and e-books.

## Data Warehouse Systems

Data warehouse systems were introduced to address the problems of RDBMSs. Data warehouse systems such as Teradata are able to scale up to terabytes of data, and they are mainly used for OLAP use cases.

Data warehousing systems have these disadvantages:

- Data warehouse systems are a costly solution.
- They still cannot process other data types such as semistructured and unstructured data.
- They cannot scale to petabytes and beyond.

All traditional data-processing technologies experience a couple of common problems: storage and performance.

Computing infrastructure can face the problem of node failures. Data needs to be available irrespective of node failures, and storage systems should be able to store large volumes of data.

Traditional data processing technologies used a scale-up approach to process a large volume of data. A scale-up approach adds more computing power to existing nodes, so it cannot scale to petabytes and more because the rest of computing infrastructure becomes a performance bottleneck.

Growing storage and processing needs have created a need for new technologies such as parallel computing technologies.

# Parallel Computing

The following are several parallel computing technologies.

## GFS and MapReduce

Google has created two parallel computing technologies to address the storage and processing problems of big data. They are Google File System (GFS) and MapReduce. GFS is a distributed file system that provides fault tolerance and high performance on commodity hardware. GFS follows a master-slave architecture. The master is called Master, and the slave is called ChunkServer in GFS. MapReduce is an algorithm based on key-value pairs used for processing a huge amount of data on commodity hardware. These are two successful parallel computing technologies that address the storage and processing limitations of big data.

## Apache Hadoop

Apache Hadoop is an open source framework used for storing and processing large data sets on commodity hardware in a fault-tolerant manner.

Hadoop was written by Doug Cutting and Mark Cafarella in 2006 while working for Yahoo to improve the performance of the Nutch search engine. Cutting named it after his son's stuffed elephant toy. In 2007, it was given to the Apache Software Foundation.

Initially Hadoop was adopted by Yahoo and, later, by companies like Facebook and Microsoft. Yahoo has about 100,000 CPUs and 40,000 nodes for Hadoop. The largest Hadoop cluster has about 4,500 nodes. Yahoo runs about 850,000 Hadoop jobs every day. Unlike conventional parallel computing technologies, Hadoop follows a scale-out strategy, which makes it more scalable. In fact, Apache Hadoop had set a benchmark by sorting 1.42 terabytes per minute.

Most of Hadoop is written in Java, but it has support for many programming languages such as C, C++, Python, and Scala through its streaming module. Apache Hadoop was initially written for high throughput and batch-processing systems. RDBMS technologies were written for frequent modifications in data, whereas Hadoop has been written for frequent reads.

Moore's law says the processing capability of a machine will double every two years. Kryder's law says the storage capacity of disks will grow faster than Moore's law. The cost of computing and storage devices will go down every day, and these two factors can support more scalable technologies. Apache Hadoop was designed while keeping these things in mind, and parallel computing technologies like this will become more common going forward.

The latest Apache Hadoop contains three modules, as shown in Figure 1-2. They are HDFS, MapReduce, and Yet Another Resource Negotiator (YARN).

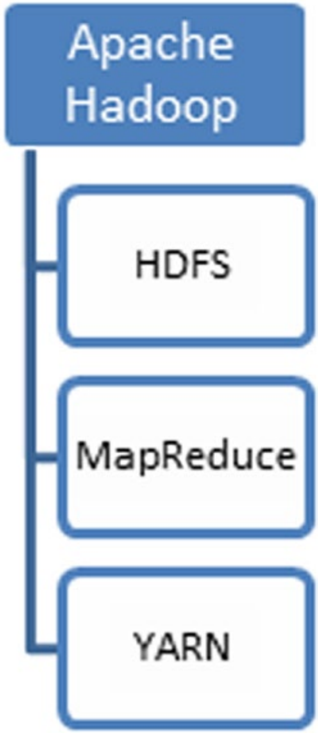


***Figure 1-2.*** *The three components of Hadroop*

## HDFS

The Hadoop distributed file system is used for storing large data sets. It divides files into blocks and stores every block on at least multiple nodes. This is called a *replication factor*, and by default it is 3. HDFS is fault-tolerant because it has more than one replica for every block, so it can handle node failures without affecting data processing. A block of HDFS is the same as an operating system block, but a HDFS block size is larger, such as 64 MB or 128 MB. Unlike traditional storage systems, it is highly scalable. It does not require any special hardware and can work on commodity hardware.

Assume you have a replication factor of 3, a block size of 64 MB, and 640 MB of data needs to be uploaded into HDFS. At the time of uploading the data into HDFS, 640 MB is divided into 10 blocks with respect to block size. Every block is stored on three nodes, which would consume 1920 MB of space on a cluster.

HDFS follows a master-slave architecture. The master is called the *name node*, and the slave is called a *data node*. The data node is fault tolerant because the same block is replicated to two more nodes. The name node was a single point of failure in initial versions; in fact, Hadoop used to go down if the name node crashed. But Hadoop 2.0+ versions have high availability of the name node. If the active name node is down, the standby name node becomes active without affecting the running jobs.

## MapReduce

MapReduce is key-value programming model used for processing large data sets. It has two core functions: Map and Reduce. They are derived from functional programming languages. Both functions take a key-value pair as input and generate a key-value pair as output.

The Map task is responsible for filtering operations and preparing the data required for the Reduce tasks. The Map task will generate intermediate output and write it to the hard disk. For every key that is being generated by the Map task, a Reduce node is identified and will be sent to the key for further processing.

The Map task takes the key-value pair as input and generates the key-value pair as output.

```
(key1, value1) ----------------> Map Task----------------> (Key2, Valu2)
```

The Reduce task is responsible for data aggregation operations such as count, max, min, average, and so on. A reduce operation will be performed on a per-key basis. Every functionality can be expressed in MapReduce.

The Reduce task takes the key and list of values as input and generates the key and value as output.

```
(key2, List (value2))--------> Reduce Task ---------------> (Key3, value3)
```

In addition to the Map and Reduce tasks, there is an extra stage called the *combiner* to improve the performance of MapReduce. The combiner will do partial aggregation on the Map side so that the Map stage has to write less data to disk.

You will now see how MapReduce generates a word count. Figure 1-3 depicts how MapReduce generates the `fruits` word count after filtering out the word *pear*.

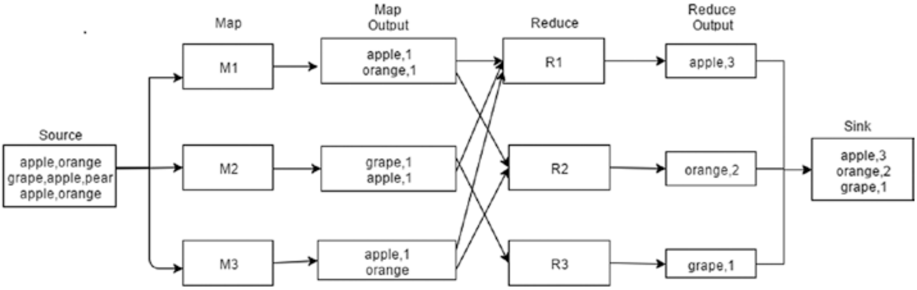


***Figure 1-3.*** *MapReduce generating a word count*

`Source` and `Sink` are HDFS directories. When you upload data to HDFS, data is divided into chunks called *blocks*. Blocks will be processed in a parallel manner on all available nodes.

The first stage is Map, which performs filtering and data preparation after tokenization. All Map tasks (`M1`, `M2`, and `M3`) will do the initial numbering for words that are useful for the final aggregation. And `M2` filters out the word *pear*.

The key and list of its values are retrieved from the Map output and sent to the reducer node. For example, the Apple key and its values (1, 1, 1) are sent to the reducer node `R1`. The reducer aggregates all values to generate the count output.

Between Map and Reduce, there is an internal stage called *shuffling* where the reducer node for the map output is identified.

You will now see how to write the same word count program using MapReduce. You first need to write a mapper class for the Map stage.

## Writing a Map Class

The following is the Map program that is used for the same tokenization and data filtering as in the shell script discussed earlier:

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class FilterMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable offset, Text line, Context context) throws
    IOException, InterruptedException {
        //tokenize line with comma as delimiter
        StringTokenizer itr = new StringTokenizer(line.toString(),",");
        //Iterate all tokens and filter pear word
        while (itr.hasMoreTokens()) {
```

```
            String strToken=itr.nextToken();
            if(!strToken.equals("pear")){
//converting string data type to text data type of mapreduce

            word.set(strToken);
            context.write(word, one);//Map output
            }

        }
    }
}
```

The Map class should extend the Mapper class, which has parameters for the input key, input value, output key, and output value. You need to override the map() method. This code specifies LongWritable for the input key, Text for the input value, Text for the output key, and IntWritable for the output value.

In the map() method, you use StringTokenizer to convert a sentence into words. You are iterating words using a while loop, and you are filtering the word *pear* using an if loop. The Map stage output is written to context.

For every run of the map() method, the line offset value is the input key, the line is the input value, the word in the line will become an output key, and 1 is the output value, as shown in Figure 1-4.


***Figure 1-4.*** *M2 stage*

The map() method runs once per every line. It tokenizes the line into words, and it filters the word *pear* before writing other words with the default of 1.

If the combiner is available, the combiner is run before the Reduce stage. Every Map task will have a combiner task that will produce aggregated output. Assume you have two *apple* words in the second line that is processed by the M2 map task.

The Map output without the combiner will look like Figure 1-5.


***Figure 1-5.*** *Map output without the combiner*