CHAPTER 1: INTROCUTION TO BIG DATA [1]

Big data is a popular term used to describe the exponential growth and availability of data, both structured and unstructured. And big data may be as important to business – and society – as the Internet has become. Why? More data may lead to more accurate analyses.

More accurate analyses may lead to more confident decision making. And better decisions can mean greater operational efficiencies, cost reductions and reduced risk.

1.1 Big data defined

As far back as 2001, industry analyst Doug Laney (currently with Gartner) articulated the now mainstream definition of big data as the three Vs of big data: volume, velocity and variety[1].

- **Volume.** Many factors contribute to the increase in data volume. Transaction-based data stored through the years. Unstructured data streaming in from social media. Increasing amounts of sensor and machine-to-machine data being collected. In the past, excessive data volume was a storage issue. But with decreasing storage costs, other issues emerge, including how to determine relevance within large data volumes and how to use analytics to create value from relevant data.
- **Velocity.** Data is streaming in at unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time. Reacting quickly enough to deal with data velocity is a challenge for most organizations.
- **Variety.** Data today comes in all types of formats. Structured, numeric data in traditional databases. Information created from line-of-business applications. Unstructured text documents, email, video, audio, stock ticker data and financial transactions. Managing, merging and governing different varieties of data is something many organizations still grapple with.

we must also consider two additional dimensions when thinking about big data:

- **Variability.** In addition to the increasing velocities and varieties of data, data flows can be highly inconsistent with periodic peaks. Is something trending in social media? Daily, seasonal and event-triggered peak data loads can be challenging to manage. Even more so with unstructured data involved.
- **Complexity.** Today's data comes from multiple sources. And it is still an undertaking to link, match, cleanse and transform data across systems. However, it is necessary to connect and

correlate relationships, hierarchies and multiple data linkages or your data can quickly spiral out of control.

1.2 Why big data should matter to us

The real issue is not that you are acquiring large amounts of data. It's what you do with the data that counts. The hopeful vision is that organizations will be able to take data from any source, harness relevant data and analyze it to find answers that enable 1) cost reductions, 2) time reductions, 3) new product development and optimized offerings, and 4) smarter business decision making. For instance, by combining big data and high-powered analytics, it is possible to:

- Determine root causes of failures, issues and defects in near-real time, potentially saving billions of dollars annually.
- Optimize routes for many thousands of package delivery vehicles while they are on the road.
- Analyze millions of SKUs to determine prices that maximize profit and clear inventory.
- Generate retail coupons at the point of sale based on the customer's current and past purchases.
- Send tailored recommendations to mobile devices while customers are in the right area to take advantage of offers.
- Recalculate entire risk portfolios in minutes.
- Quickly identify customers who matter the most.
- Use clickstream analysis and data mining to detect fraudulent behavior.

1.3 ISSUES WITH BIG DATA [2]

Analysts estimate that enterprises will spend $34 billion on big data investments in 2013. Clearly, we believe that data holds immense potential. Unfortunately, we are nowhere near ready to unlock its potential as most enterprises are simply applying the same thinking and technologies in newer packages to solve their problems. We most certainly have advanced our abilities to store and process data. What is holding us back are people problems.

There are three cultural issues that we must solve to unlock the latent potential of data: data silos, a lack of data scientists, and broken communication channels between data scientists and business users.

1. Data Silos

In most enterprises, the data generated by a functional area ends up being the property of that group. This leads to two problems. First, it's difficult to get a "complete" view of the data. Consider all the silos and systems that hold data: CRM, ticketing, bug tracking, fulfillment, etc. Getting all the relevant systems to even talk to each other is a huge challenge. Second, there's significant cultural dissonance within organizations. Typically, each group controlling a data silo ends up caring more about their power and place in a department rather than the success of the organization as a whole. Organizations need to pool their data to find the answers to and get a complete view of their data.

2. Data Scientists

A typical enterprise generally has 10x more IT employees than analysts or data scientists. The process of analysis starts with a line of business request. IT collects data from various databases and transfers it to data scientists. Large teams of data scientists are deployed who spend months (or sometimes years) querying the data. Hiring data scientists (with advanced background in statistics, computer science, and some functional expertise) to accelerate the process is difficult because people with these skills are extremely scarce. The demand and interest in data scientists is skyrocketing, as Google Trends can attest, while we are producing fewer of them. What we need is a new class of technologies that amplify the impact data scientists and allow more people to become data scientists.

3. Communication

Finally, the biggest hurdle to fully realize the potential of data science is the lack of communication between data scientists and business users. Said another way, the analytical gap between a data scientist and a business user is so wide that even communicating insights poses a problem. Anything that does not make intuitive sense is often regarded with skepticism, or not fully understood, by business users, which can lead to missed opportunities. Data scientists and business users need to align, work more closely together, and build trust to solve business problems.

Conclusion

It is imperative that we solve these three problems. We need to:

1. Open up data to everyone and break down data silos,

2. Amplify the productivity of data scientists and do more through automation, and

3. Develop a more collaborative culture, which allow business users and data scientists to communicate more effectively.

CHAPTER 2 : INTROCUCTION TO HADOOP

2.1 SQL –NoSQL and HADOOP

Big data is a more complicated world because the scale is much larger. The information is usually spread out over a number of servers, and the work of compiling the data must be coordinated among them. In the past, the work was largely delegated to the database software, which would use its magical **join** mechanism to compile tables, then add up the columns before handing off the rectangle of data to the reporting software that would paginate it. This was often harder than it sounds. Database programmers can tell you the stories about complicated JOIN commands that would lock up their database for hours as it tried to produce a report for the boss who wanted his columns just so.

The game is much different now. Hadoop is a popular tool for organizing the racks and racks of servers, and NoSQL databases are popular tools for storing data on these racks. These mechanism can be much more powerful than the old single machine, but they are far from being as polished as the old database servers. Although SQL may be complicated, writing the JOIN query for the SQL databases was often much simpler than gathering information from dozens of machines and compiling it into one coherent answer. Hadoop jobs are written in Java, and that requires another level of sophistication. The tools for tackling big data are just beginning to package this distributed computing power in a way that's a bit easier to use. Many of the big data tools are also working with NoSQL data stores. These are more flexible than traditional relational databases, but the flexibility isn't as much of a departure from the past as Hadoop. NoSQL queries can be simpler because the database design discourages the complicated tabular structure that drives the complexity of working with SQL. The main worry is that software needs to anticipate the possibility that not every row will have some data for every column.

2.2 THE HADOOP APPROACH

Hadoop is designed to efficiently process large volumes of information by connecting many commodity computers together to work in parallel. The theoretical 1000-CPU machine described earlier would cost a very large amount of money, far more than 1,000 single-CPU or 250 quad-

core machines. Hadoop will tie these smaller and more reasonably priced machines together into a single cost-effective compute cluster.

2.2.1 COMPARISON TO EXISTING TECHNIQUES

Performing computation on large volumes of data has been done before, usually in a distributed setting. What makes Hadoop unique is its simplified programming model which allows the user to quickly write and test distributed systems, and its efficient, automatic distribution of data and work across machines and in turn utilizing the underlying parallelism of the CPU cores.

Grid scheduling of computers can be done with existing systems such as Condor. But Condor does not automatically distribute data: a separate SAN must be managed in addition to the compute cluster. Furthermore, collaboration between multiple compute nodes must be managed with a communication system such as MPI. This programming model is challenging to work with and can lead to the introduction of subtle errors.

2.2.2 DATA DISTRIBUTION

In a Hadoop cluster, data is distributed to all the nodes of the cluster as it is being loaded in. The Hadoop Distributed File System (HDFS) will split large data files into chunks which are managed by different nodes in the cluster. In addition to this each chunk is replicated across several machines, so that a single machine failure does not result in any data being unavailable. An active monitoring system then re-replicates the data in response to system failures which can result in partial storage. Even though the file chunks are replicated and distributed across several machines, they form a single namespace, so their contents are universally accessible.

Data is conceptually record-oriented in the Hadoop programming framework. Individual input files are broken into lines or into other formats specific to the application logic. Each process running on a node in the cluster then processes a subset of these records. The Hadoop framework then schedules these processes in proximity to the location of data/records using knowledge from the distributed file system. Since files are spread across the distributed file system as chunks, each compute process running on a node operates on a subset of the data. Which data operated on by a node is chosen based on its locality to the node: most data is read from the local disk straight into the CPU, alleviating strain on network bandwidth and preventing unnecessary network transfers. This strategy of moving computation to the data, instead of moving the data to the computation allows Hadoop to achieve high data locality which in turn results in high performance.

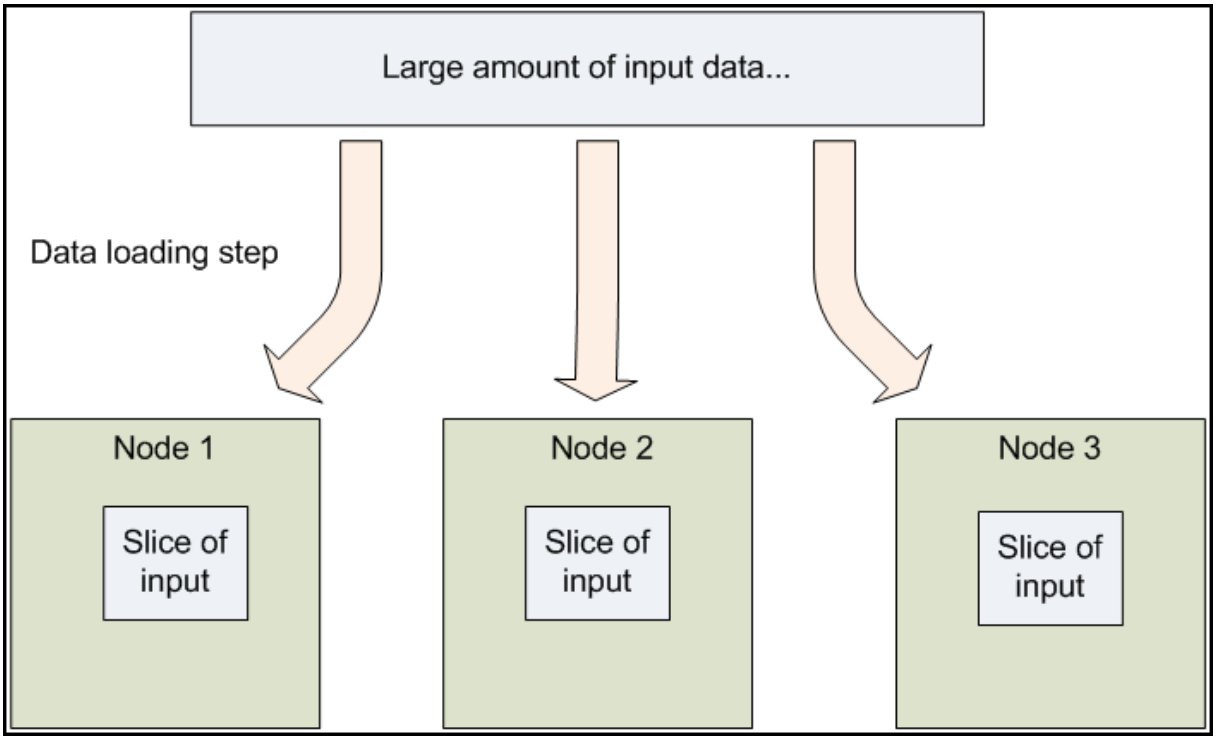


Figure 1.1: Data is distributed across nodes at load time.[3].

2.2.3. MAPREDUCE: ISOLATED PROCESSES

Hadoop limits the amount of communication which can be performed by the processes, as each individual record is processed by a task in isolation from one another. While this sounds like a major limitation at first, it makes the whole framework much more reliable. Hadoop will not run just any program and distribute it across a cluster. Programs must be written to conform to a particular programming model, named "MapReduce."

In MapReduce, records are processed in isolation by tasks called *Mappers*. The output from the Mappers is then brought together into a second set of tasks called *Reducers*, where results from different mappers can be merged together.

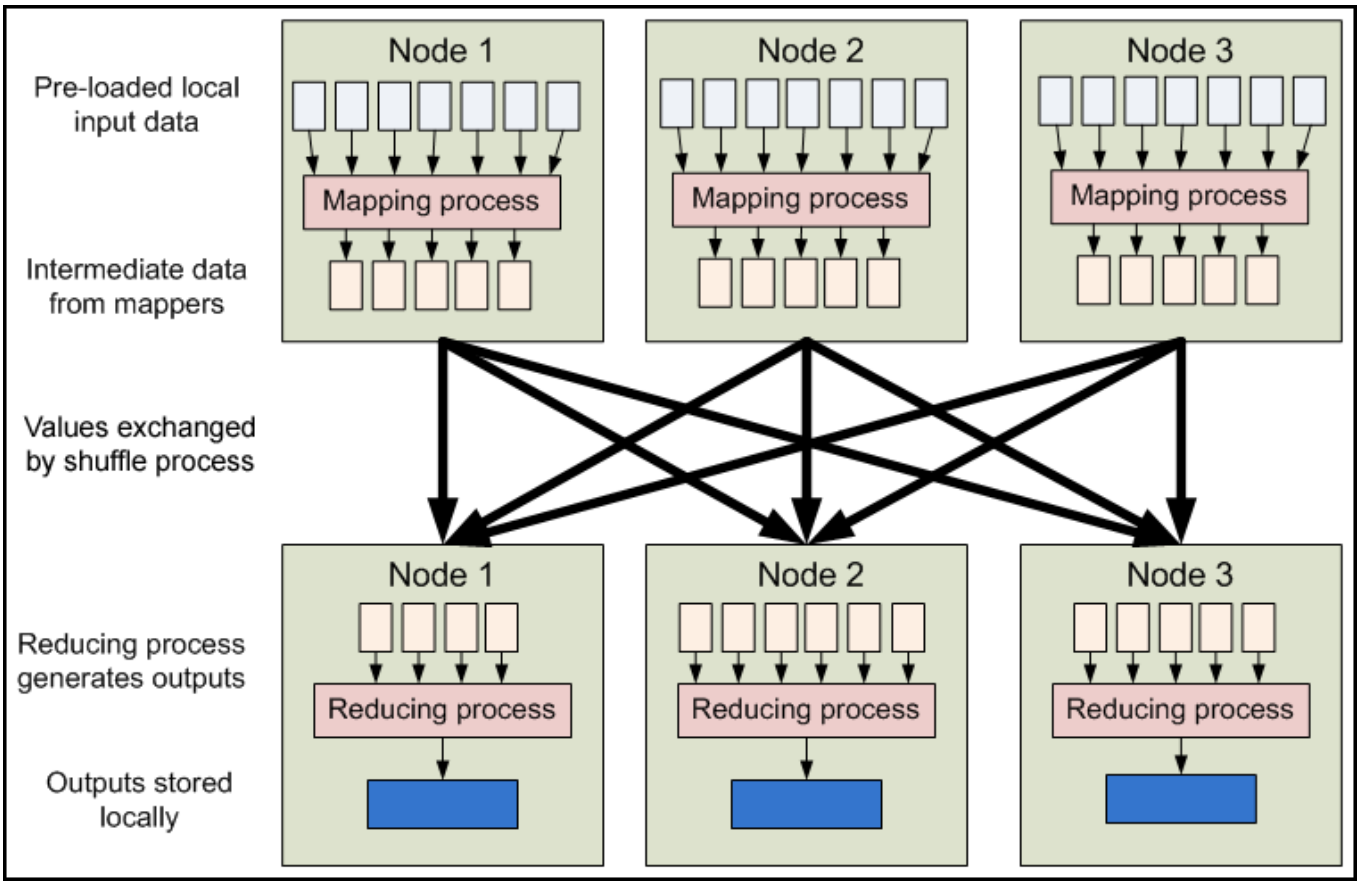


Figure 1.2: Mapping and reducing tasks run on nodes where individual records of data are already present.[3]

Separate nodes in a Hadoop cluster still communicate with one another. However, in contrast to more conventional distributed systems where application developers explicitly marshal byte streams from node to node over sockets or through MPI buffers, communication in Hadoop is performed *implicitly*. Pieces of data can be tagged with key names which inform Hadoop how to send related bits of information to a common destination node. Hadoop internally manages all of the data transfer and cluster topology issues.

By restricting the communication between nodes, Hadoop makes the distributed system much more reliable. Individual node failures can be worked around by restarting tasks on other machines. Since user-level tasks do not communicate explicitly with one another, no messages need to be exchanged by user programs, nor do nodes need to roll back to pre-arranged checkpoints to partially restart the computation. The other workers continue to operate as though nothing went wrong, leaving the challenging aspects of partially restarting the program to the underlying Hadoop layer.

2.2.4 FLAT SCALABILITY

One of the major benefits of using Hadoop in contrast to other distributed systems is its flat scalability curve. Executing Hadoop on a limited amount of data on a small number of nodes may not demonstrate particularly stellar performance as the overhead involved in starting Hadoop programs is relatively high. Other parallel/distributed programming paradigms such as MPI (Message Passing Interface) may perform much better on two, four, or perhaps a dozen machines. Though the effort of coordinating work among a small number of machines may be better-performed by such systems, the price paid in performance and engineering effort (when adding more hardware as a result of increasing data volumes) increases non-linearly.

A program written in distributed frameworks other than Hadoop may require large amounts of refactoring when scaling from ten to one hundred or one thousand machines. This may involve having the program be rewritten several times; fundamental elements of its design may also put an upper bound on the scale to which the application can grow.

Hadoop, however, is specifically designed to have a very flat scalability curve. After a Hadoop program is written and functioning on ten nodes, very little--if any--work is required for that same program to run on a much larger amount of hardware. Orders of magnitude of growth can be managed with little re-work required for your applications. The underlying Hadoop platform will manage the data and hardware resources and provide dependable performance growth proportionate to the number of machines available.

2.3. HADOOP DISTRIBUTED FILE SYSTEM

HDFS, the Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high-throughput access to this information. Files are stored in a redundant fashion across multiple machines to ensure their durability to failure and high availability to very parallel applications.

2.3.1. Distributed File System Basics

A distributed file system is designed to hold a large amount of data and provide access to this data to many clients distributed across a network. There are a number of distributed file systems that solve this problem in different ways.

NFS, the Network File System, is the most ubiquitous distributed file system. It is one of the oldest still in use. While its design is straightforward, it is also very constrained. NFS provides remote access to a single logical volume stored on a single machine. An NFS server makes a portion of its local file system visible to external clients. The clients can then mount this remote

file system directly into their own Linux file system, and interact with it as though it were part of the local drive.

One of the primary advantages of this model is its transparency. Clients do not need to be particularly aware that they are working on files stored remotely. The existing standard library methods like open(), close(),fread(), etc. will work on files hosted over NFS.

But as a distributed file system, it is limited in its power. The files in an NFS volume all reside on a single machine. This means that it will only store as much information as can be stored in one machine, and does not provide any reliability guarantees if that machine goes down (e.g., by replicating the files to other servers). Finally, as all the data is stored on a single machine, all the clients must go to this machine to retrieve their data. This can overload the server if a large number of clients must be handled. Clients must also always copy the data to their local machines before they can operate on it.

HDFS is designed to be robust to a number of the problems that other DFS's such as NFS are vulnerable to. In particular:

- HDFS is designed to store a very large amount of information (terabytes or petabytes). This requires spreading the data across a large number of machines. It also supports much larger file sizes than NFS.
- HDFS should store data reliably. If individual machines in the cluster malfunction, data should still be available.
- HDFS should provide fast, scalable access to this information. It should be possible to serve a larger number of clients by simply adding more machines to the cluster.
- HDFS should integrate well with Hadoop MapReduce, allowing data to be read and computed upon locally when possible.

But while HDFS is very scalable, its high performance design also restricts it to a particular class of applications; it is not as general-purpose as NFS. There are a large number of additional decisions and trade-offs that were made with HDFS. In particular:

- Applications that use HDFS are assumed to perform long sequential streaming reads from files. HDFS is optimized to provide streaming read performance; this comes at the expense of random seek times to arbitrary positions in files.
- Data will be written to the HDFS once and then read several times; updates to files after they have already been closed are not supported. (An extension to Hadoop will provide support for appending new data to the ends of files; it is scheduled to be included in Hadoop 0.19 but is not available yet.)
- Due to the large size of files, and the sequential nature of reads, the system does not provide a mechanism for local caching of data. The overhead of caching is great enough that data should simply be re-read from HDFS source.
- Individual machines are assumed to fail on a frequent basis, both permanently and intermittently. The cluster must be able to withstand the complete failure of several machines, possibly many happening at the same time (e.g., if a rack fails all together). While performance may degrade proportional to the number of machines lost, the system as

a whole should not become overly slow, nor should information be lost. Data replication strategies combat this problem.

The design of HDFS is based on the design of GFS, the Google File System. Its design was described in a paperpublished by Google.

HDFS is a block-structured file system: individual files are broken into blocks of a fixed size. These blocks are stored across a cluster of one or more machines with data storage capacity. Individual machines in the cluster are referred to as DataNodes. A file can be made of several blocks, and they are not necessarily stored on the same machine; the target machines which hold each block are chosen randomly on a block-by-block basis. Thus access to a file may require the cooperation of multiple machines, but supports file sizes far larger than a single-machine DFS; individual files can require more space than a single hard drive could hold.

If several machines must be involved in the serving of a file, then a file could be rendered unavailable by the loss of any one of those machines. HDFS combats this problem by replicating each block across a number of machines (3, by default).

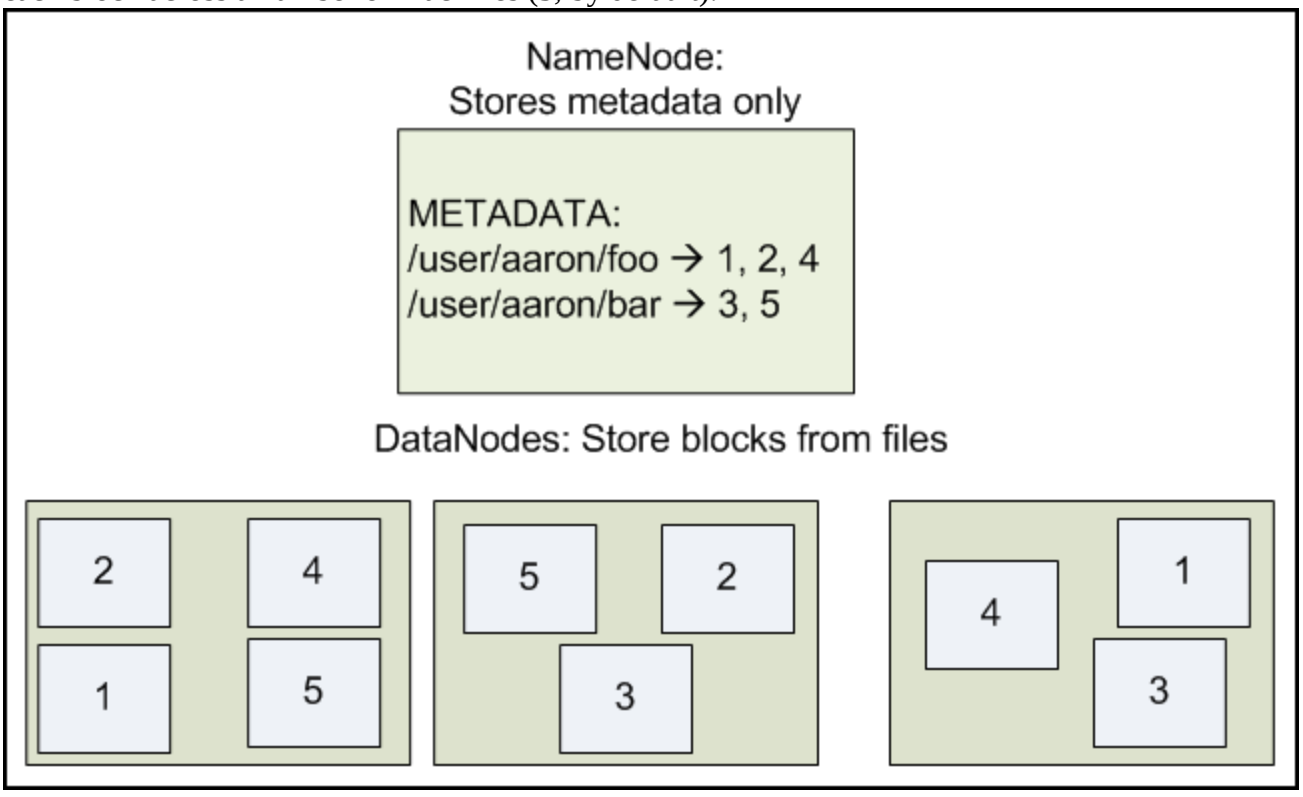


Figure 2.3: DataNodes holding blocks of multiple files with a replication factor of 2. The NameNode maps the filenames onto the block ids.

Most block-structured file systems use a block size on the order of 4 or 8 KB. By contrast, the default block size in HDFS is 64MB -- orders of magnitude larger. This allows HDFS to decrease the amount of metadata storage required per file (the list of blocks per file will be

smaller as the size of individual blocks increases). Furthermore, it allows for fast streaming reads of data, by keeping large amounts of data sequentially laid out on the disk. The consequence of this decision is that HDFS expects to have very large files, and expects them to be read sequentially. Unlike a file system such as NTFS or EXT, which see many very small files, HDFS expects to store a modest number of very large files: hundreds of megabytes, or gigabytes each. After all, a 100 MB file is not even two full blocks. Files on your computer may also frequently be accessed "randomly," with applications cherry-picking small amounts of information from several different locations in a file which are not sequentially laid out. By contrast, HDFS expects to read a block start-to-finish for a program. This makes it particularly useful to the MapReduce style of programming described in Module 4. That having been said, attempting to use HDFS as a general-purpose distributed file system for a diverse set of applications will be suboptimal.

Because HDFS stores files as a set of large blocks across several machines, these files are not part of the ordinary file system. Typing ls on a machine running a DataNode daemon will display the contents of the ordinary Linux file system being used to host the Hadoop services -- but it will not include any of the files stored inside the HDFS. This is because HDFS runs in a separate namespace, isolated from the contents of your local files. The files inside HDFS (or more accurately: the blocks that make them up) are stored in a particular directory managed by the DataNode service, but the files will named only with block ids. You cannot interact with HDFS-stored files using ordinary Linux file modification tools (e.g., ls, cp, mv, etc). However, HDFS does come with its own utilities for file management, which act very similar to these familiar tools. A later section in this tutorial will introduce you to these commands and their operation.

It is important for this file system to store its metadata reliably. Furthermore, while the file data is accessed in a write once and read many model, the metadata structures (e.g., the names of files and directories) can be modified by a large number of clients concurrently. It is important that this information is never desynchronized. Therefore, it is all handled by a single machine, called the NameNode. The NameNode stores all the metadata for the file system. Because of the relatively low amount of metadata per file (it only tracks file names, permissions, and the locations of each block of each file), all of this information can be stored in the main memory of the NameNode machine, allowing fast access to the metadata.

To open a file, a client contacts the NameNode and retrieves a list of locations for the blocks that comprise the file. These locations identify the DataNodes which hold each block. Clients then read file data directly from the DataNode servers, possibly in parallel. The NameNode is not directly involved in this bulk data transfer, keeping its overhead to a minimum.

Of course, NameNode information must be preserved even if the NameNode machine fails; there are multiple redundant systems that allow the NameNode to preserve the file system's metadata

even if the NameNode itself crashes irrecoverably. NameNode failure is more severe for the cluster than DataNode failure. While individual DataNodes may crash and the entire cluster will continue to operate, the loss of the NameNode will render the cluster inaccessible until it is manually restored. Fortunately, as the NameNode's involvement is relatively minimal, the odds of it failing are considerably lower than the odds of an arbitrary DataNode failing at any given point in time.

2.4 HADOOP INSTALLATION

Steps to install Hadoop frame work on eclipse

(All steps are for 64 bit system)

1. Pre-Requisite

--- Jdk 1.7

--- eclipse juno 4.2

2. Install cygwin

---- download cygwin

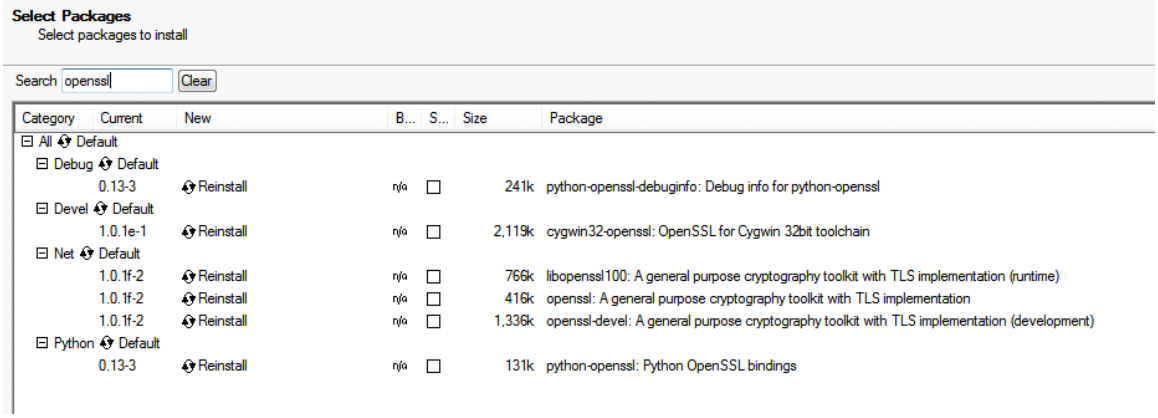
---- install all packages under

--openssh

--openssl

---tcp_warrpers

---- difutills



Setup SSH daemon

Both hadoop scripts and eclipse plugin need password less ssh to operate. This section describes how to set it up in the Cygwin environment.

Configure ssh daemon

Open Cygwin command prompt

Execute the following command

ssh-host-config

When asked if privilege separation should be used, answer **no**.

When asked if sshd should be installed as a service, answer **yes.**

When asked about the value of CYGWIN environment variable enter **ntsec.**

Find my computer icon either on your desktop or in the start-up menu, right-click on it and select Manage from the context menu.

Open Services and Applications in the left-hand panel then select the Services item.

Find the CYGWIN sshd item in the main section and right-click on it.(start cygwin sshd service)

Open cygwin command prompt

Execute the following command to generate keys

ssh-keygen

When prompted for filenames and pass phrases **press ENTER to accept default values**.

After command has finished generating they key, enter the following command to change into your .ssh directory

cd ~/.ssh

Check if the keys where indeed generated by executing the following command

ls -l

You should see two file **id_rsa.pub and id_rsa with the recent creation dates**. These files contain authorization keys.

To register the new authorization keys enter the following command. Note that double brackets, they are very important.

cat id_rsa.pub >> authorized_keys

Now check if the keys where set-up correctly by executing the following command

ssh localhost

Since it is a new ssh installation you warned that authenticity of the host could not be established and will be **prompted whether you really want to connect, answer yes and press ENTER**. You should see the cygwin prompt again, which means that you have successfully connected.

Now execute the command again

ssh localhost

This time you should not be prompted for anything.

Download, Copy and Unpack Hadoop

1)Download hadoop 0.19.1 and place in some folder on your computer such as C:\Java.

2)Open Cygwin command prompt.

Execute the following command

cd .

3)Then execute the following command to get your home directory folder shown in the Windows Explorer window.

explorer .

4)Open another explorer window and navigate to the folder that contains the downloaded hadoop archive.

5)Copy the hadoop archive into your home directory folder.

Unpack Hadoop Installation

1)Open the new cygwin window

2)After new cygwin window appears, execute the following command:

tar -xzf hadoop-0.19.1.tar.gz

3)When you see the new prompt execute the following command:

ls -l

This command will list the contents of your home directory. You should see a newly created directory called hadoop-0.19.1

4)Next execute the following commands

cd hadoop-0.19.1

ls -l

you will see you hadoop files

Configure Hadoop

1)Open a new cygwin window and execute the following commands

cd hadoop-0.19.1

cd conf

explorer .

2)As a result of the last command you will see the explorer window for the 'conf' directory popped up. Minimize it for now or move it to the side.

3)Launch eclipse

4)Bring up the the 'conf' explorer window opened in the step 2 and drug the file hadoop-site to the eclipse main window.

Insert the following lines between **<configuration> and </configuration> tags.**

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9100</value>

</property>

<property>

<name>mapred.job.tracker</name>

<value>localhost:9101</value>

</property>

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

Format the namenode

Open a new cygwin window.

Then execute the following commands

cd hadoop-0.19.1

mkdir logs

bin/hadoop namenode -format

error you will get

1)JAVA_HOME not set

open cygwin window

cd hadoop-0.19.1

explorer .

got to conf folder

open hadoop-env.sh

and

export JAVA_HOME="C:\Program Files\Java\jdk1.7.0_51"

Windows installation fails with "bin/hadoop: line 243: c:\Program: command not found"

soln

this is beacuse of space between Program Files in JAVA_HOME and linux does not understand this and give errors

cd hadoop-0.19.1

explorer .

goto bin -> hadoop

at line 243 and replace with

JAVA_PLATFORM=`CLASSPATH=${CLASSPATH} "${JAVA}" org.apache.hadoop.util.PlatformName | sed -e "s/ /_/g"`

Install Hadoop plugin

Open the new Cygwin window. And execute the following commands

cd hadoop-0.19.1

cd contrib

cd eclipse-plugin

explorer .

2)Shrink the newly popped window and move it to the right side of the screen.

3)Open another explorer window, either through "My Computer" icon or by using the "Start -> Run" menu. Navigate to your Eclipse installation and then open the "plugin" folder of your Eclipse installation.

4)**Copy the file "hadoop-0.19.1-eclipse-plugin.jar, from the Hadoop eclipse plugin folder to the Eclipse plugins folder.**

5)Close both explorer windows

6)Start Eclipse

7)Click on the open perspective icon , which is usually located in the upper-right corner the eclipse application. Then select Other from the menu.

8)Select Map/Reduce from the list of perspectives and press "OK" button.

Start Hadoop Cluster

open 5 cygwind window and in one window open namenode , in 2nd open secondaryname node and so on

Start the namenode in the first window by executing

cd hadoop-0.19.1

bin/hadoop namenode

Start the secondary namenode in the second window by executing

cd hadoop-0.19.1

bin/hadoop secondarynamenode

Start the job tracker the third window by executing

cd hadoop-0.19.1

bin/haoop jobtracker

Start the data node the fourth window by executing

cd hadoop-0.19.1

bin/haoop datanode

Start the task tracker the fifth window by executing

cd hadoop-0.19.1

bin/haoop tasktracker

Setup Hadoop Location in Eclipse

1) the Eclipse environment.

2)Open Map/Reduce perspective by clicking on the open perspective icon (), select "Other" from the menu, and then select "Map/Reduce" from the list of perspectives.

3)After you switched to the Map/Reduce perspective. Select the Map/Reduce Locations tab located at the bottom portion of your eclipse environment. Then right click on the blank space in that tab and select "New Hadoop location...." from the context menu.

4)Fill in the following items, as shown on the figure above.

Location Name -- localhost

Map/Reduce Master

Host -- localhost

Port -- 9101

DFS Master

Check "Use M/R Master Host"

Port -- 9100

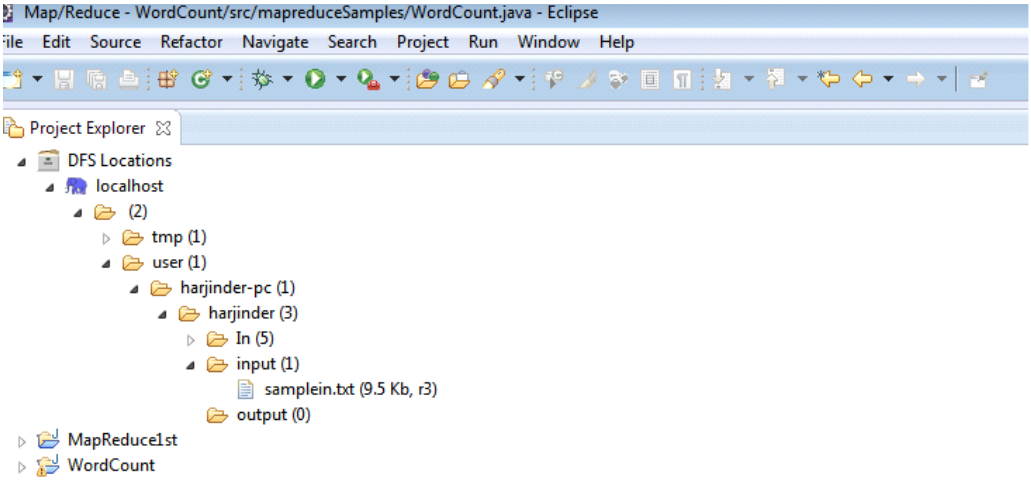
User name -- User

Then press the Finish button.

5)After you closed the Hadoop location settings dialog you should see a new location appearing in the "Map/Reduce Locations" tab.

6)In the Project Explorer tab on the lefthand side of the eclipse window, find the DFS Locations item. Open it up using the "+" icon on the left side of it, inside of it you should see the localhost location

reference with the blue elephant icon. Keep opening up the items below it until you see something like the image below



CHAPTER 3 MAP – REDUCE AND WORD COUNT PROBLEM

3.1 INTRODUCTION

Traditional programming tends to be serial in design and execution. We tackle many problems with a sequential, stepwise approach and this is reflected in the corresponding program. With parallel programming, we break up the processing workload into multiple parts, that can be executed concurrently on multiple processors. Not all problems can be parallelized. The challenge is to identify as many tasks as possible that can run concurrently. Alternatively, we can identify data groups that can be processed concurrently. This will allow us to divide the data among multiple concurrent tasks. The most straightforward situation that lends itself to parallel programming is one where there is no dependency among data. Data can be split into chunks and each process can be assigned a chunk to work on. If we have lots of processors, we can split the data into lots of chunks. A master/worker approach is a design where a master process coordinates overall activity. It identifies the data, splits it up based on the number of available workers, and assigns a data segment to each worker. A worker receives the data segment from the master, performs whatever processing is needed on the data, and then sends results to the master. At the end, the master does whatever final processing (e.g., merging) is needed to the resultant data.

3.1.1 MAP-REDUCE ETYMOLOGY

MapReduce is was created at Google in 2004 by Jeffrey Dean and Sanjay Ghemawat. [4].The name is inspired from map and reduce functions in the LISP programming language. In LISP, the map function takes as parameters a function and a set of values. That function is then applied to each of the values. For example:

(map ‘length ‘(() (a) (ab) (abc)))

applies the length funciton to each of the three items in the list. Since length returns the length of an item, the result of map is a list containing the length of each item:

(0 1 2 3)

The reduce function is given a binary function and a set of values as parameters. It combines all the values together using the binary function. If we use the + (add) function to reduce the list (0 1 2 3):

(reduce #'+ '(0 1 2 3))

we get:

If we think about how the map operation, we realize that each application of the function to a value can be performed in parallel (concurrently) since there is no dependence of one upon another. The reduce operation can take place only after the map is complete.

MapReduce is not an implementation of these LISP functions; they are merely an inspiration and etymological predecessor.

3.1.2 MAP REDUCE

MapReduce is a framework for parallel computing. Programmers get a simple API and do not have to deal with issues of parallelization, remote execution, data distribution, load balancing, or fault tolerance. The framework makes it easy for one to use thousands of processors to process huge amounts of data (e.g., terabytes and petabytes).

From a user's perspective, there are two basic operations in MapReduce: Map and Reduce.

The Map function reads a stream of data and parses it into intermediate (key, value) pairs. When that is complete, the Reduce function is called once for each unique key that was generated by Map and is given the key and a list of all values that were generated for that key as a parameter. The keys are presented in sorted order.

As an example of using MapReduce, consider the task of counting the number of occurrences of each word in a large collection of documents. The user-written Map function reads the document data and parses out the words. For each word, it writes the (key, value) pair of (word, 1). That is, the word is treated as the key and the associated value of 1 means that we saw the word once. This intermediate data is then sorted by MapReduce by keys and the user's Reduce function is called for each unique key. Since the only values are the count of 1,Reduce is called with a list of a "1" for each occurence of the word that was parsed from the document. The function simply adds them up to generate a total word count for that word. Here's what the code looks like:

```
map(String key, String value):
// key: document name,  value: document contents
for each word w in value:
  EmitIntermediate(w, "1");

reduce(String key, Iterator values):
// key: a word;  values: a list of counts
int result = 0;
```

```
for each v in values:
  result += ParseInt(v);
Emit(AsString(result));
```

Let us now look at what happens in greater detail.

3.1.3. MAP REDUCE MORE DEATIL

To the programmer, MapReduce is largely seen as an API: communication with the various machines that play a part in execution is hidden. MapReduce is implemented in a master/worker configuration, with one master serving as the coordinator of many workers. A worker may be assigned a role of either a map worker or a reduce worker.

Step 1. Split input

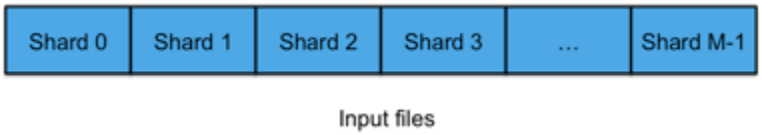


Figure 3.1. Split input into shards[5]

The first step, and the key to massive parallelization in the next step, is to split the input into multiple pieces. Each piece is called a split, or shard. For M map workers, we want to have M shards, so that each worker will have something to work on. The number of workers is mostly a function of the amount of machines we have at our disposal.

The MapReduce library of the user program performs this split. The actual form of the split may be specific to the location and form of the data. MapReduce allows the use of custom readers to split a collection of inputs into shards, based on specific format of the files.

Step 2. Fork processes

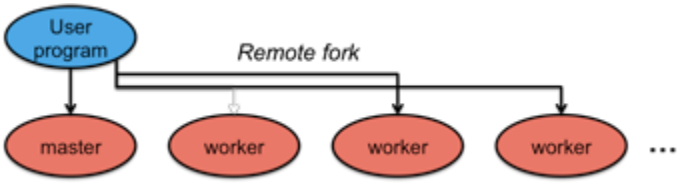


Figure 3.2. Remotely execute worker processes[5]

The next step is to create the master and the workers. The master is responsible for dispatching jobs to workers, keeping track of progress, and returning results. The master picks idle workers and assigns them either a map task or a reduce task. A map task works on a single shard of the original data. A reduce task works on intermediate data generated by the map tasks. In all, there will be M map tasks and Rreduce tasks. The number of reduce tasks is the number of partitions defined by the user. A worker is sent a message by the master identifying the program (map or reduce) it has to load and the data it has to read.

Step 3. Map

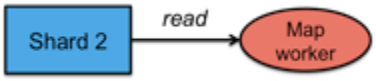


Figure 3.3. Map task[5]

Each map task reads from the input shard that is assigned to it. It parses the data and generates (key, value) pairs for data of interest. In parsing the input, the map function is likely to get rid of a lot of data that is of no interest. By having many map workers do this in parallel, we can linearly scale the performance of the task of extracting data.

Step 4: Map worker: Partition

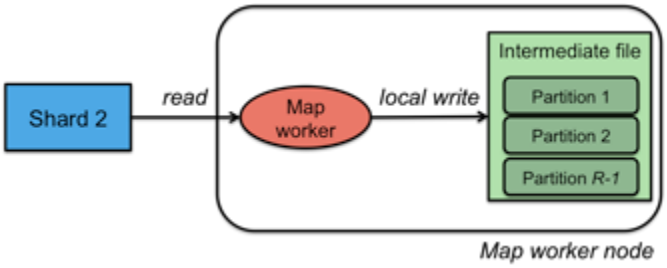


Figure 3.4. Create intermediate files[5]

The stream of (key, value) pairs that each worker generates is buffered in memory and periodically stored on the local disk of the map worker. This data is partitioned into R regions by a partitioning function.

The partitioning function is responsible for deciding which of the R reduce workers will work on a specific key. The default partitioning function is simply a hash of key modulo R but a user can replace this with a custom partition function if there is a need to have certain keys processed by a specific reduce worker.

Step 5: Reduce: Sort (Shuffle)

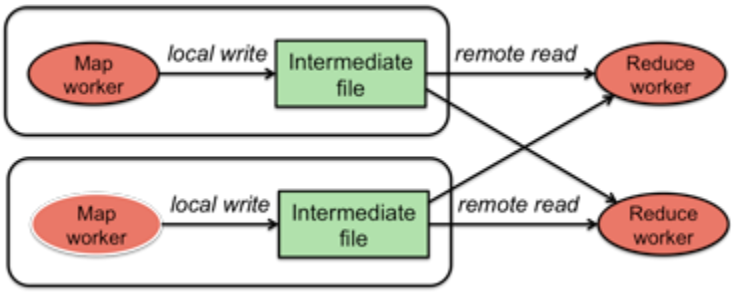


Figure 3.5. Sort and merge partitioned data[5]

When all the map workers have completed their work, the master notifies the reduce workers to start working. The first thing a reduce worker needs to is to get the data that it needs to present to the user'sreduce function. The reduce worker contacts every map worker via remote procedure calls to get the (key, value) data that was targeted for its partition. This data is then sorted by the keys. Sorting is needed since it will usually be the case that there are many occurrences of the same key and many keys will map to the same reduce worker (same partition). After sorting, all occurrences of the same key are grouped together so that it is easy to grab all the data that is associated with a single key.

This phase is sometimes called the shuffle phase.

Step 6: Reduce function

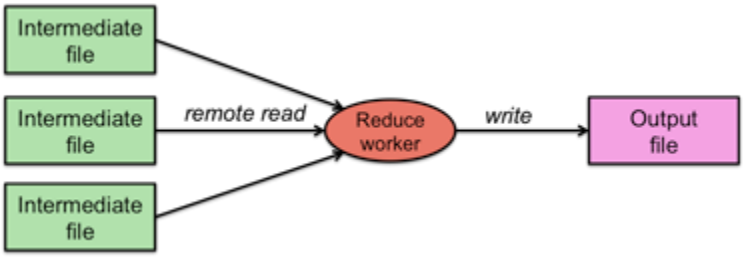


Figure 3.6. Reduce function writes output[5]

With data sorted by keys, the user's Reduce function can now be called. The reduce worker calls theReduce function once for each unique key. The function is passed two parameters: the key and the list of intermediate values that are associated with the key.

The Reduce function writes output sent to file.

Step 7: Done!

When all the reduce workers have completed execution, the master passes control back to the user program. Output of MapReduce is stored in the R output files that the R reduce workers created.

The big picture

Figure 3.7 illustrates the entire MapReduce process. The client library initializes the shards and creates map workers, reduce workers, and a master. Map workers are assigned a shard to process. If there are more shards than map workers, a map worker will be assigned another shard when it is done. Map workers invoke the user's Map function to parse the data and write intermediate (key, value) results onto their local disks. This intermediate data is partitioned into R partitions according to a partioning function. Each of R reduce workers contacts all of the map workers and gets the set of (key, value) intermediate data that was targeted to its partition. It then calls the user's Reduce function once for each unique key and gives it a list of all values that were generated for that key. The Reduce function writes its final output to a file that the user's program can access once MapReduce has completed.

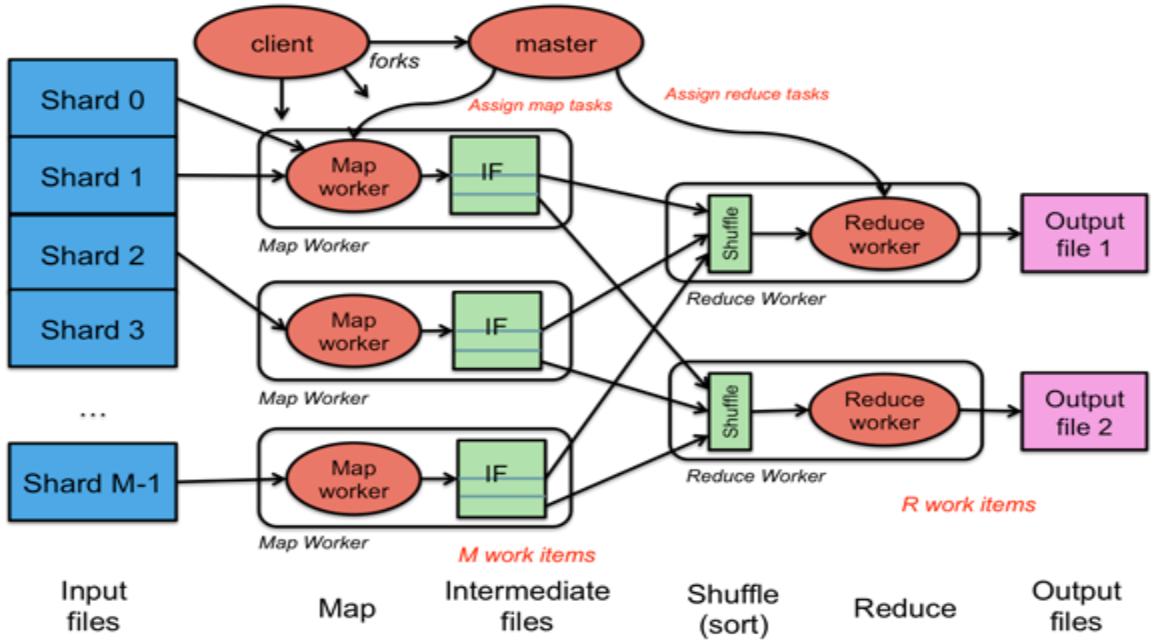


Figure 3.7. MapReduce[5]

3.1.4 DEALING WITH FAILURE

The master pings each worker periodically. If no response is received within a certain time, the worker is marked as failed. Any map or reduce tasks that have been assigned to this worker are reset back to the initial state and rescheduled on other worker

3.2. A Word Count Problem

1)Launch Eclipse

2)Right click on the blank space in the Project Explorer window and select New -> Project.. to create a new project.

3)Select Map/Reduce Project from the list of project types.

4) Press the Next button.

5)Fill in the project name and then click on Configure Hadoop Installation link. Which is located on the right side of the project configuration window. This will bring up the Project preferences

6)When Project preferences window shows up, enter the location of the hadoop directory in the Hadoop Installation Directory

7)After you entered the location close the preferences window by pressing OK button, and then close the Project window by the Finish button.

8)Now you have created your first Hadoop eclipse project. You should see its name in the Project Explorer tab.

and run a word count in that

Code :

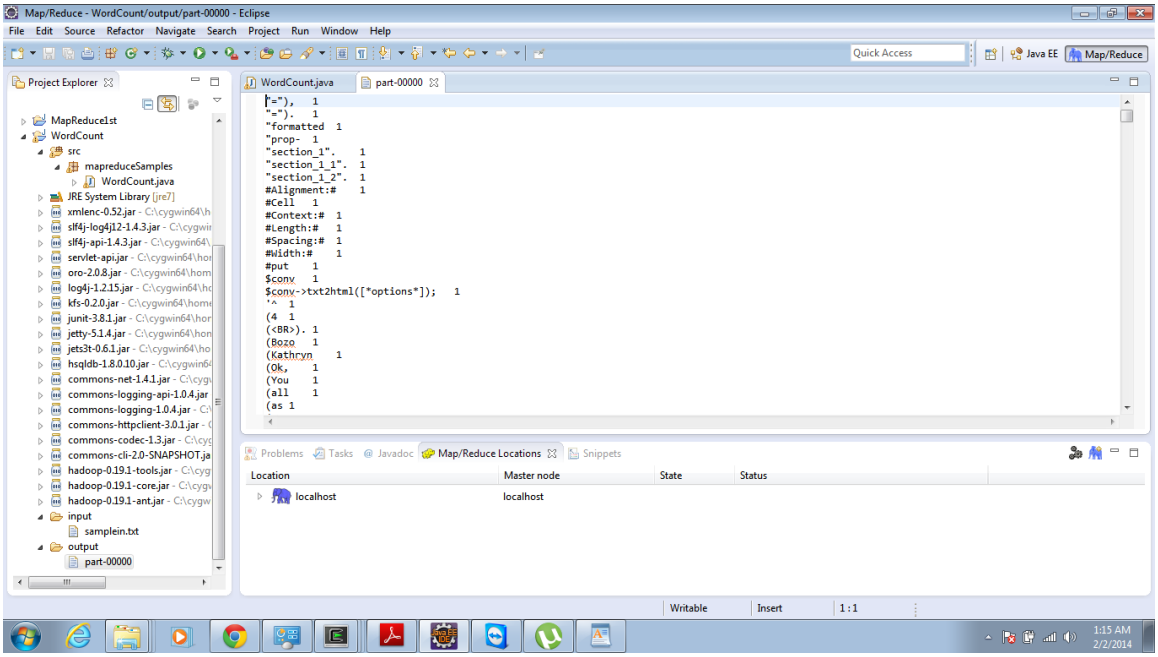
```
package mapreduceSamples;


	import java.io.IOException;

	import java.util.*;


	import org.apache.hadoop.fs.Path;

	import org.apache.hadoop.conf.*;

	import org.apache.hadoop.io.*;

	import org.apache.hadoop.mapred.*;

import org.apache.hadoop.util.*;


	public class WordCount {

   public static class Map extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {

		private final static IntWritable one = new IntWritable(1);

		private Text word = new Text();


		public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException {

	  String line = value.toString();

		StringTokenizer tokenizer = new StringTokenizer(line);

		while (tokenizer.hasMoreTokens()) {

		  word.set(tokenizer.nextToken());

		  output.collect(word, one);

		}
```

```
    }
  }

  public static class Reduce extends MapReduceBase implements Reducer<Text,
IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
      int sum = 0;
      while (values.hasNext()) {
  sum += values.next().get();
      }
      output.collect(key, new IntWritable(sum));
    }
  }

  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path("input"));
    FileOutputFormat.setOutputPath(conf, new Path("output"));
```

```
        JobClient.runJob(conf);
    } }
```

input:samplein.txt(any test file)

snapshot



CHAPTER 4: WORK DONE DURING INDEPENDENT STUDY

4.1 . Reports submitted [Git Hub link]
4.2 Personation on HDFS and Name Node Stability[Git Hub link & Letter from Dr. Yeung attached]
4.3 One day seminar attended on Hadoop and Map Reduce at Library on [04/05/14]
4.4 List of papers and books referred are in Reference section at end .

CHAPTER 5: Final OutCome of Independent Study

This course was a great learning experience of me. Big Data, Hadoop , Map Reduce were one of the topics I always wanted to jump in .My main reason for this course was to explore this field and have some basic knowledge on these topics as I wanted to explore different fields for my final project and this is top priority. This study gave me chance and ample first hand experience on hadoop installation and map-reduce programming basic . Here , I would like to sum up my whole experience and would like to share my idea that how I can take step forward to my final project form here.

1.Basic Knowledge

As I told I am very much interested in Big Data and would like to do my final project in this field , this study helped me having basic knowledge about HDFS, HADOOP, MAPREDUCE, BIG DATA. This is surely helping me thinking about big picture of my intended project and possible issues I can face both technical and resources related .

This study has helped me building confidence that if given sufficient time I can jump in and do an project keeping an eye on possible issues I could face.

2. Networking

Major benefit of this study was I had chance to meet like minded people, who are equally interested in big data. Starting with attending workshop at henry madden library on [--/--/----] and then my self giving an ! hour long lecture to class of CSCI-244 on HDFS, HADOOP and helping people setting up there system for Apache Hadoop. This all was equally fun and excitement. During this I participated in various active groups on FACEBOOK and LINKEDIN to discuss about hadoop and future discourse of the technology.

3. Major Project Ideas : Sharing and Discussions

During course of my independent study through workshop I have attended , I had chance to meet seniors working and interested in this field . It did resulted in lots of valuable discussion and a blue print that what kind of projects people are taking up .

5.1. FUTURE PLANS:

I would like to continue with my learning curve and would like to take hadoop as platform for my final project.

1. During summer I would like to set up an mini- cluster with 5-7 computers and would like to run HADOOP with actual HDFS as almost everybody right now working on HADOOP works on single node systems where actual ability of HDFS is not fully explored .
2. My project idea is to create an dummy data of hospitals with patient reports , like their full medical history from birth to till date, their family history. In simpler words every available information about the patient's life.
3. And there will be an artificial data at some police station of all the criminals and their crimes.
4. Using map reduce I would like to find the criminals and their medical and family history.
5. Then using HIVE I would like to give an dummy survey that "HOW FAMILY AND MEDICAL HISTORY IS RELATED TO CRIME".
6. I feel that if considered with real time data we can find out and take suitable steps to guide people that what kind of social and personal behavior can lead to what kind of future threats to some one's life . I don't see right now how it can help actually what according to me it will be an interesting survey to carry on.

REFRENCES :

[1]. ETA Group. "3D Data Management: Controlling Data Volume, Velocity, and Variety." February 2001.

[2].Availble: http://venturebeat.com/2013/12/04/the-3-big-problems-in-big-data-hint-theyre-all-about-people/

[3]. Yahoo Hadoop wiki .Avaialble: https://developer.yahoo.com/hadoop/tutorial/module1.html[28/04/2014]

[4]. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *OSDI*, 2004, pp. 137–150.

[5] Paul K.,. MApReduce : A Frame Work For Large Scale Parallel Processing. Available : https://www.cs.rutgers.edu/~pxk/417/notes/content/mapreduce.html[01/05/2014]

[6]. Hadoop in Action, Chuk Lam.