HADOOP *Kichidi*

Fundamentals of Big Data Analytics

Dr. Kamakshaiah Musunuru

dr.m.kamakshaiah@gmail.com

Foreword

Hi!
This is just a teaching learning material for certain course called *Big data analytics* which I teach to BBA and MBA students in India. [1]

Author,
Kamakshaiah Musunuru.
https://github.com/Kamakshaiah

[1] This material is a development version. The production version is yet to be out. Completely no guarantee/warranty. User's caution is required :)

Contents

Chapter 1

Introduction

Big data is a term used to refer to the study and applications of data sets that are too complex for traditional data-processing application software to adequately deal with. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. Big data was originally associated with three key concepts: *volume, variety*, and *velocity*. Other concepts later attributed with big data are veracity (i.e., how much noise is in the data) and value. [1]

Modern usage of the term "big data" tends to refer to the use of predictive analytics, user behavior analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set. "There is little doubt that the quantities of data now available are indeed large, but that's not the most relevant characteristic of this new data ecosystem." Analysis of data sets can find new correlations to "spot business trends, prevent diseases, combat crime and so on." Scientists, business executives, practitioners of medicine, advertising and governments alike regularly meet difficulties with large data-sets in areas including Internet search, fintech, urban informatics, and business informatics. Scientists encounter limitations in e-Science work, including meteorology, genomics, connectomics, complex physics simulations, biology and environmental

research.

Data sets grow rapidly in part because they are increasingly gathered by cheap and numerous information sensing Internet of Things (IoT) devices such as mobile devices, aerial (remote sensing), software logs, cameras, microphones, radio-frequency identification (RFID) readers and wireless sensor networks. The world's technological per-capita capacity to store information has roughly doubled every 40 months since the 1980s; as of 2012, every day 2.5 exabytes (2.5×10e18) of data are generated. Based on an IDC report prediction, the global data volume will grow exponentially from 4.4 zettabytes to 44 zettabytes between 2013 and 2020. By 2025, IDC predicts there will be 163 zettabytes of data. One question for large enterprises is determining who should own big-data initiatives that affect the entire organization. See table 1.1 for more details on data measures in bytes.

Value	Symbol	SI
1000	k	kilo
1000^2	M	mega
1000^3	G	giga
1000^4	T	tera
1000^5	P	peta
1000^6	E	exa
1000^7	Z	zetta
1000^8	Y	yotta

Table 1.1: for multiples of bits (bit) or bytes (B)

Relational database management systems, desktop statistics and software packages used to visualize data often have difficulty handling big data. The work may require "massively parallel software running on tens, hundreds, or even thousands of servers". What qualifies as being "big data" varies depending on the capabilities of the users and their tools, and expanding capabilities make big data a moving target. "For some organizations, facing hundreds of gigabytes of data for the first time may trigger a need to reconsider data management options. For others, it may take tens or hundreds of terabytes before data size becomes a significant consideration."

1.1 Definition

The term has been in use since the 1990s, with some giving credit to *John Mashey* for popularizing the term. Perhaps, in simple terms; we might be able to define Big data as

> "Anything that includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time".

Big data philosophy encompasses unstructured, semi-structured and structured data, however the main focus is on unstructured data. Big data "size" is a constantly moving target, as of 2012 ranging from a few dozen terabytes to many exabytes of data. Big data requires a set of techniques and technologies with new forms of integration to reveal insights from datasets that are diverse, complex, and of a massive scale.

In 2016 De Mauro and *et al* defines that [2]

> "It represents the information assets characterized by such a high volume, velocity and variety to require specific technology and analytical methods for its transformation into value".

Additionally, a new *V, veracity,* is added by some organizations to describe it, revisionism challenged by some industry authorities. The three Vs (volume, variety and velocity) have been further expanded to other complementary characteristics of big data:

1. *Machine learning*: big data often doesn't ask why and simply detects patterns.
2. *Digital footprint*: big data is often a cost-free byproduct of digital interaction.

A 2018 definition states

> "Big data is where parallel computing tools are needed to handle data", and notes, "This represents a distinct and clearly defined change in the computer science used, via parallel programming theories, and losses of some of

the guarantees and capabilities made by Codd's relational model". [3]

1.2 Characteristics

Hilbert, Martin and others mentions below mentioned characteristics: [4]

1. *Volume*:The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.

2. *Variety*: The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.

3. *Velocity*: In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time.

4. *Veracity*: The data quality of captured data can vary greatly, affecting the accurate analysis.

1.3 Big data in Action

1.3.1 Marketing

Pandora, Spotify, Apple Music, Tidal... These are a few of the biggest names in streaming music. Livestreaming has forever changed how we listen to music, and the disruption these platforms are spreading through the music industry is only just starting. The next area that's ripe for sweeping change is the use of big data and music personalization for live-stream listeners.

How are streaming services using data to understand what their customers want to hear? How are they using a wide array of data sources

to make better listener recommendations? What insights do their personalization efforts provide other consumer-focused businesses?

Livestreaming "On a Scale That We've Never Seen""

Live-streaming music is the platform consumers demand, and it's rapidly displacing physical sales (CDs) and downloads (via services such as iTunes). The Nielsen Music U.S. Mid-Year Report noted only halfway through 2017 that on-demand audio streams reached over 184 billion streams, a 62.4-percent increase over the same time period in 2016. David Bakula, the senior vice president of music industry insights for Nielsen, noted, "The rapid adoption of streaming platforms by consumers has generated engagement with music on a scale that we've never seen before."

The key phrase to note is "engagement with music." Subscription-based streaming has made discovery of new music easier. For lesser-known artists, it's easier to find an audience and less costly to produce and distribute music. For consumers, it allows them to listen to as much music as they like for as long as they want. Their options aren't limited to a download. They can listen to anything, anywhere.

Distributors have stopped selling ownership, and instead sell access. The barriers to engagement have dropped between musicians and listeners. How are streaming services using big data to improve customer engagement with their services?

Spotify's Approach to the Personalization of Streaming

There are different approaches to combining big data and music personalization. For streaming leader Spotify, it was necessary to grow the personalization team. Within a few short years, the company went from having one tiny team to multiple teams located in New York, Boston, and Stockholm.

Spotify notes that platform features are "but the tip of a huge personalization iceberg." Discover Weekly is one personalization feature that delivers a two-hour compilation of custom-made music to Spotify users, which is "based both on your own listening as well as what others are playlisting and listening to around the songs you love … It's

like having your best friend make you a personalized mix tape every single week."

Release Radar is a feature that Spotify added in 2016. It provides users with a weekly playlist with songs from new albums. The company admits that for new albums, there isn't data about streaming or playlists to draw from. Instead, it relies on "audio research" that uses deep learning to curate similarities between music. By comparing this research to a user's listening history, new tracks that might pique a listener's interest can be determined. Pandora's Big Data Music Project

Spotify's audio research approach is similar to Pandora's Music Genome Project, which began in 1999. In this project, teams of trained musicians listen to and categorize music. From that work, Pandora has built a collection of 450 musical attributes that describe a song's melody, harmony, rhythm, form, instrumentation, lyrics, and voice.

That work informs the platform's machine learning and algorithms. Pandora combines the data compiled in the project with data about what users listened to, as well as the listener's thumbs-up or thumbs-down choices when they are presented with a new song. With the Music Genome Project, Pandora hopes to deliver the "most comprehensive music analysis ever undertaken" and provide listeners with "personalized radio that plays what you love and continually evolves with your tastes."

What Lessons Can Be Learned?

The sheer volume of music and other audio being streamed by listeners reveals that streaming services have tapped into a strong consumer need. This popularity is not something they're taking for granted. The lessons being applied to big data and music offer insight that other consumer-focused retailers should heed.

- Give consumers a reason to stay. Spotify grew their music personalization team for a reason: engagement matters. Use data to drive loyalty and inform the features and options you create for customers.

- Recognize that consumers want to be known by businesses they use. More than 50 percent of surveyed millennials, Gen Xers, and Gen Zers appreciate receiving product and service recommendations based on their purchase history.
- Understand the importance of being data-driven. Retailers can no longer survive operating on instinct or gut feel. The world is being driven by data, and companies must commit to creating the systems that support their data operations.

Live-streaming platforms continue to explore more ways for big data to drive customer engagement and loyalty. The growth and customer satisfaction being driven by these efforts should be music to the ears of retailers of all kinds.

1.3.2 Manufacturing

If you work in manufacturing, you know how quickly competitors can gain an advantage: a plant improvement here, a supply chain tweak there—it all adds up. Additionally, the challenges facing manufacturers are driving them to transform the way they manufacture, distribute, sell, and support their offerings across the entire product life cycle. Using big data in manufacturing can give you the upper hand by making you more competitive.

Gaining an Innovative Edge in Productivity

Competition in the manufacturing industry is becoming highly unpredictable as innovation comes from an increasingly wide array of regions and markets. This can leave companies struggling to keep up, making efficiency in supply chains and manufacturing plants more important than ever. Exacerbating this competition is the accelerating innovation cycle. Improved technical capabilities enable competitors to design, engineer, manufacture and distribute products more quickly than ever before, reducing their time to market.

So, can using big data in manufacturing help businesses meet these competitive challenges? Absolutely. Companies with a more holistic view of their enterprise can spot potential issues and optimize their processes to retain a competitive edge.

For example, big data can also drive efficiency in product manufacturing processes through machine learning. This branch of artificial intelligence, which analyzes historical data to understand "normal" process performance, enables companies to monitor plant processes to detect anomalies, and optimize product quality and production yield. This, along with the ability to leverage machine learning-enabled predictive maintenance models, can dramatically minimize plant equipment downtime, driving down overall manufacturing costs even further.

Upgrading Your Supply Chains

Another problem manufacturers face is that their competitors are taking advantage of more complex, rapidly evolving supply chains. Digital supply chains respond faster to changes in demand due to a real-time flow of information, complete visibility, and the ability to respond quickly.

As manufacturers come under pressure to navigate these supply chains for better pricing and delivery terms, they must also ensure that they meet regulatory compliance challenges around environmental and corporate quality controls. Sourcing nonoptimal components that don't meet environmental standards could create compliance problems further down the line.

Using Data to Understand Your Needs

To prevent these problems while making the best use of the supply chain, manufacturers need a holistic understanding of conditions throughout the delivery pipeline. They must extend their visibility from the factory to the supply chain, drawing data from a range of stakeholders, including suppliers and logistics firms. This can create a complete view of the supply chain that will help manufacturers predict and adapt for emerging issues.

The more data sources that you can consume, the more you can understand and refine your manufacturing process. Pulling in supply chain data from logistics partners and suppliers can help you further refine your products by looking for patterns that affect quality and production times. Seeing patterns in supply chain behavior can help you

avoid disruptions in materials supply, further enhancing your manufacturing efficiency.

Addressing Challenges With Customers

At the other end of the supply chain lies the consumer. There, too, companies face new challenges. Consumers are increasingly empowered, enjoying an unprecedented range of product and service options. Manufacturers that don't anticipate their customers' needs risk losing them. It's all about growth in this hyper-competitive marketplace and growth demands excellence in customer experience with customer-centric interactions, and innovative products and services throughout the product life cycle.

The smartest vendors turn these customer challenges into opportunities. They use big data to gather and analyze customer requirements and feed those insights back into the product design cycle in a process of continuous improvement. They can use a variety of sources for this, including customer relationship management and marketing systems, sales databases, and technical support feeds.

Gaining Insights to Achieve Long-Term Success

Companies that master big data can meet these industry challenges and set themselves up for longer-term success. Instead of aiming only at short-term goals, such as increasing production volume and lowering costs, they can use data from this unified supply chain and manufacturing ecosystem to gain insights into new market opportunities.

Rather than blindly turning the crank to hit the next quarter's sales target, smart companies can use this data to identify new geographical and product markets, and plan targeted business expansions, paving the way for structured, healthy growth.

This ultimate use of big data will come after manufacturers use data sources to create a more cohesive view of their product life cycles. Along the way, they can improve their competitive advantage by realizing tangible improvements in productivity and quality.

The data is already latent in the manufacturing plant and in your

logistical supply chains. By using big data in manufacturing to power your decision-making, you can unlock its value.

1.3.3 Big data and Sustainability

Global spending on big data and business analytics will grow to more than $187 billion in the next few years. At the same time, fears over the use of big data continue to grow—according to research from the University of Cambridge, 71 percent of consumers believe brands that access their personal data use it unethically.

But that's not always true. Smart organizations, in both the private and public sectors, are instead utilizing big data for social good. From addressing complex social issues to fighting hunger, big data is making it easier to create lasting, positive change in society.

Understanding the Need for Change

Various academics have voiced their concerns. Cathy O'Neil, former director of The Lede Program at Columbia University, says algorithms and mathematical models should be used to help reduce bias and inequality. However, the reverse is often true—big data models can sometimes help reinforce discrimination, and automated data–led decisions can leave unfortunate individuals further exposed to poverty.

Yet while the potential exists for big data to create inequality, there's also the opportunity to improve society. Using big data for social good, therefore, should be the aim for all organizations that utilize big data—and examples of that positive use of information already exist.

Sponsoring Joined-Up Thinking

Some experts believe public and private partnerships are the best way to help use big data for social good. Large, blue-chip organizations have the skills and technologies that are developing game-changing big data models. Lending this capability to nonprofit organizations could help change the world for the better.

There are examples of this link-up taking place around the globe. The nonprofit DataKind brings together leading data scientists with high-

impact social organizations: take its recent partnership with Pollinate Energy to address the detection of urban poor communities in Bangalore via satellite images and other data points.

Academic institutions are forging partnerships, too. The University of Chicago runs a summer program called The Data Science for Social Good Fellowship, which helps train aspiring data scientists to work on projects with a positive impact. Working closely with governments and nonprofits, research fellows in the program take on real-world problems in key areas, including education, health, and public safety.

Creating Multidisciplinary Connections

Mobile operator representation organization GSMA recently launched an initiative to use vendors' big data capabilities to address humanitarian crises. The program, known as Big Data for Social Good, aims to meet significant crises—such as epidemics and natural disasters—head on, and is being launched alongside 19 of the world's leading mobile operators.

Key executives in these organizations are helping to drive change. Nuria Oliver, director of data science research at Vodafone, has spent the past decade exploring how big data can be used to improve society. She says multidisciplinary approaches work best: bringing together data scientists and epidemiologists creates a confluence of talent to help solve social problems.

Another example from the mobile space comes in the form of LUCA, the big data unit within Telefonica. The organization—which is running a Big Data for Social Good initiative—is using an Apache Hadoop–based data platform to collect and store information in a data repository to undertake data transformation and analysis.

Making More Informed Decisions

What all parties recognize is that big data is about more than simply improving customer experiences and business profits. There is growing recognition of the game-changing power of big data from key organizations, such as the United Nations, the World Economic Forum, and the World Bank. In short, the right insight at the right time

can improve society.

Take the United Network for Organ Sharing (UNOS), which is the private, nonprofit organization that runs the U.S. organ transplant system. UNOS manages the requirements of a network of hundreds of hospitals, transplant centers, organ procurement professionals, and thousands of volunteers. The decisions UNOS makes change peoples' lives.

UNOS uses a connected data platform solution and 100 percent open-source tools to create a self-service reporting system for centers and organ procurement organizations. The organization's data warehouse gives doctors a past and present view across patients and transplanted organs. The result is informed decisions that have the patients' best interests in mind.

The amount of data being collected—and the money spent on technologies to analyze this information—continues to rise. While experts are concerned by potential risks to individuals, some organizations are already using their investment to help sponsor an improvement in societal conditions. At a time of huge change, we all have a responsibility to find ways to use big data for social good.

1.3.4 Consumer Finance

Anyone who's bought a house without paying the full price upfront knows that applying for a mortgage is anything but simple and straightforward. Even the customers with the most solid financial footings go through lengthy and inconvenient processes to borrow for a home. However, financial institutions and other money services businesses are starting to investigate using big data analytics to cut down on the red tape in mortgage applications and make it easier for their customers to apply for loans.

How Big Data Changes Mortgage Applications

Mortgage banking has long been about modeling loan performance appropriately, based on a financial institution's business goals, appetite for and ability to take risks, and the overall economy. Risk departments have used models to drive decisions about how many

loans should be issued, what types of pricing and mitigation should be applied to individual loans, and what demographics and segments of the economy represent good loans for a bank or credit union.

More Data, Greater Assessments

The key difference with the emergence of big data analytics is the amount of data at companies' disposal. Models with more data have gotten much better at assessing risk and providing an accurate idea of the behavior of a portfolio of mortgages. These models succeed because they use a much broader base of data types—including factors such as a prospective customer's geolocation and transaction history—and are able to correlate things like comparable people who behave in a particular manner.

Data is also more available. The amount of data an organization can collect on one person is vaster than what used to be possible and, once assembled, the data can build a dramatically different picture. Data sets consisting of anything from geolocation to credit card transactions and store sales are starting to be available for sale to lending institutions and other outfits, and that data can be correlated with existing models to generate better decisions.

The types of mathematical models being used today have gotten better in recent years because more data input equals better tweaking and adjusting, and exponentially better training over much larger amounts of data. Simply put, there are more types of input, and that leads to better decision-making.

Data Leads to Faster Decision-Making

The speed of mortgage decisions is also improving, thanks to several factors. Today's big data–oriented infrastructure is more robust and generally runs in near-real time. Most big data implementations have infrastructure that allows a lot of data—of different types or different functions—to be stored very effectively and very cheaply. Additionally, this infrastructure allows easy access to historical data in order to tweak models and validate them, providing analysts the ability to run models effectively and very quickly.

Key to these new capabilities is a computational engine that allows models to run in parallel and very fast, along with a streaming capability to be able to run a model on the fly as new data is coming in.

Potential Pitfalls to Be Aware Of

Ultimately, the idea behind drawing conclusions from big data is to build models and automation that drive better business decisions. But there are pitfalls to watch out for.

Losing the Human Touch

From a lending standpoint, as these models get better, the decision on whether someone should be approved can be driven more by the computer and less by a human decision. The removal of the human element in decision-making has advantages and disadvantages.

If a model is making a choice, it's typically only programmed to examine risk factors, therefore leaving it blind to other factors a human might consider. On the other hand, a human's decision can be influenced by biases extraneous to a purely risk-based decision. Removing biases can be an excellent attribute, but you also remove the ability to make judgment calls, which can mean making a borderline good loan go bad or vice versa.

Thus, a good model will always have two quality assurance steps: additional safety checks to ensure you don't come up with the wrong answer because of a faulty model, and a review of decisions so that a human can assess the overall picture and either sustain or override the decision.

Getting Overwhelmed by Increased Loan Volumes

Another risk of utilizing data analytics as part of your mortgage and lending efforts is volume. With better models and decision-making, an organization can acquire an unwieldy appetite for loans. When you increase volume, however, risk can expand quickly, and the negative consequences can occur much more quickly over a larger base of people.

Being Outsmarted by Fraud

Finally, there's a risk in how intelligent and tolerant models can be in driving lending decisions. In particular, fraud prevention needs to be factored in early on. Models need to be smart enough to prevent gaming. If nefarious actors know a model works with, say, three factors, they may then attempt to force those factors to drive a desired result. Modelers and analysts must understand this important issue and ensure that their models are smart enough to combat it.

Ultimately, the use of big data analytics for mortgages drives better institutional profitability—more loans that are closed in less time, and that perform better overall and have a reduced risk of loss. As long as you're mindful of the common trouble areas, you can capitalize on data's value and see the results you're looking for.

Notes

[1]Laney, Doug (2001). "3D data management: Controlling data volume, velocity and variety". META Group Research Note. 6 (70).

[2]Read more about this definition at https://www.emeraldinsight.com/doi/full/10.1108/LR-06-2015-0061

[3]Fox, Charles (2018-03-25). Data Science for Transport. Springer.

[4]Hilbert, Martin. "Big Data for Development: A Review of Promises and Challenges. Development Policy Review". martinhilbert.net. Retrieved 7 October 2015.

1.4 Questions

1.4.1 Short answer questions

1. Define big data.
2. Define big data analytics.
3. What are the characteristics of big data?

1.4.2 Essay questions

1. Illustrate 3V's big data using supporting/relevant information.
2. Write about various applications in the following areas of business.
   (a) Marketing
   (b) Finance
   (c) HRM
   (d) Operations

Chapter 2

About Linux

2.1 Know about your computer!

Console is Character User Interface (CUI) of OS. Many advanced operations can be done using console but not through graphic user interface (GUI). I found the below definition for *operating system* in Wikipedia:

> An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.

All computer programs, excluding firmware, require an operating system to function. The operating system acts as an intermediary between programs and the computer hardware. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers. The dominant desktop operating system is Microsoft Windows with a market share of around 85%. OS X by Apple Inc. is in second place (9%), and Linux is in third position (1.5%). Microsoft is leading in desktop computing. In the mobile sector Android by Google is dominant with 63% and iOS by Apple is placed second with around 25%. Linux is dominant in the server and supercomputing sectors. Other specialized classes of operating systems, such as embedded and real-time systems,

exist for many applications.

Parts of OS

In this Chapter I will be dealing with following aspects of OS.

- Processing
- Memory
- File System (FS)
- Networking

These are few essential aspects which constitutes a computer. Processing deals with hardware and software that transforms inputs into outputs. In every computer there is one processor (today as more than one called multi-core processors). The whole typical job of the processor is to convert input data into *meaningful* information. The *meaning* is what is the expectation of the programmer. Programmers use software called OS and languages in order to achieve the aforesaid *transformation*. Logic is defined through writing software program which consists of few or many statements which in turn consisting of language specific commands. For instance, it is possible to use few commands like *if, for* together with data types and definitions to achieve arithmetic calculations. Following is the sample program that performs division in Linux Terminal.

```
a=2
b=1
echo $b/$a | bc -l
```

First two lines has the variables called *a, b*. The third line has the logic which essentially deals with division. We will know more about BASH programming in forthcoming sections.

Memory is the second most important aspect of OS. There are two types of memory known as RAM and ROM. RAM is the temporary also known as volatile, whereas ROM is not. Linux has few very informative commands to know and process memory in the computer. The third aspect of the OS is File System also known as FS. These commands are highly important to organize data in memory using

structures known as trees. For instance, it is possible to know the whole directory structre in Linux using the command `tree` inside any directory.

```
###:/home\$ tree
.
|__mk

1 directory, 0 files
```

By the way you need to install the utility `tree` using `sudo apt install tree` in Ubuntu like Debian systems. The above output shows that I have only one directory with a name `mk` inside another directory `home`.

```
###:~\$ pwd
/home/mk
```

`pwd` is the command used for *present working directory* in Linux Terminal.

2.2 Linux

Linux is a Unix-like and mostly POSIX-compliant computer operating system (OS) assembled under the model of free and open-source software development and distribution. The defining component of Linux is the Linux kernel, an operating system kernel first released on October 5, 1991 by Linus Torvalds.The Free Software Foundation uses the name GNU/Linux to describe the operating system, which has led to some controversy. [5]

Linux was originally developed as a free operating system for personal computers based on the Intel x86 architecture, but has since been ported to more computer hardware platforms than any other operating system. The development of Linux is one of the most prominent examples of free and open-source software collaboration. The underlying source code may be used, modified and distributed—commercially or non-commercially—by anyone under the terms of its respective licenses, such as the GNU General Public License. Typically, Linux is packaged in a form known as a Linux distribution (or distro for short)

for both desktop and server use. Some of the most popular mainstream Linux distributions are Arch Linux, CentOS, Debian, Fedora, Gentoo Linux, Linux Mint, Mageia, openSUSE and Ubuntu, together with commercial distributions such as Red Hat Enterprise Linux and SUSE Linux Enterprise Server. Distributions include the Linux kernel, supporting utilities and libraries, many of which are provided by the GNU Project, and usually a large amount of application software to fulfil the distribution's intended use.

2.2.1 History

The history of Linux is more interestingly dramatic. The Unix operating system was conceived and implemented in 1969 at ATT's Bell Laboratories in the United States by Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna. First released in 1971, Unix was written entirely in assembly language as it was common practice at the time. Later, in a key pioneering approach in 1973, it was rewritten in the C programming language by Dennis Ritchie (with exceptions to the kernel and I/O). The availability of a high-level language implementation of Unix made its porting to different computer platforms easier.

Due to an earlier antitrust case forbidding it from entering the computer business, ATT was required to license the operating system's source code to anyone who asked. As a result, Unix grew quickly and became widely adopted by academic institutions and businesses. In 1984, ATT divested itself of Bell Labs; freed of the legal obligation requiring free licensing, Bell Labs began selling Unix as a proprietary product.

The GNU Project, started in 1983 by Richard Stallman, has the goal of creating a "complete Unix-compatible software system" composed entirely of free software. Work began in 1984. Later, in 1985, Stallman started the Free Software Foundation and wrote the GNU General Public License (GNU GPL) in 1989. By the early 1990s, many of the programs required in an operating system (such as libraries, compilers, text editors, a Unix shell, and a windowing system) were completed, although low-level elements such as device drivers, daemons, and the kernel were stalled and incomplete. In 1991, while attending the Uni-

versity of Helsinki, Torvalds became curious about operating systems and frustrated by the licensing of MINIX (which was originally developed by Andrew S. Tanenbaum) which at the time limited it to educational use only. [6] He began to work on his own operating system kernel, which eventually became the Linux kernel.

Torvalds began the development of the Linux kernel on MINIX and applications written for MINIX were also used on Linux. Later, Linux matured and further Linux kernel development took place on Linux systems. GNU applications also replaced all MINIX components, because it was advantageous to use the freely available code from the GNU Project with the fledgling operating system; code licensed under the GNU GPL can be reused in other computer programs as long as they also are released under the same or a compatible license. Torvalds initiated a switch from his original license, which prohibited commercial redistribution, to the GNU GPL. Developers worked to integrate GNU components with the Linux kernel, making a fully functional and free operating system.

2.2.2 Linux OS Design

A Linux-based system is a modular Unix-like operating system, deriving much of its basic design from principles established in Unix during the 1970s and 1980s. Such a system uses a monolithic kernel, the Linux kernel, which handles process control, networking, access to the peripherals, and file systems. Device drivers are either integrated directly with the kernel, or added as modules that are loaded while the system is running.

2.2.3 POSIX

The Portable Operating System Interface (POSIX) is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines both the system- and user-level application programming interfaces (API), along with command line shells and utility interfaces, for software compatibility (portability) with variants of Unix and other operating systems. POSIX is also a trademark of the IEEE. POSIX is intended to be

used by both application and system developers. POSIX has the following standard tools for computing.

- Command Line Interface (CLI)
- Scripting Interface
- Services and Utilities
- Message Passing
- Shared Memory
- Asynchronous and Synchronous I/O
- Priority Scheduling
- Real-Time Signals
- Clocks and Timers
- and many more ...

Few OSes like Android, Linux, MINIX, FreeBSD, NetBSD, OpenBSD, OpenSolaris, Xenix etc. are POSIX compliance.

2.3 SHELL

The most generic sense of the term shell means any program that users employ to type commands. A shell hides the details of the underlying operating system and manages the technical details of the operating system kernel interface, which is the lowest-level, or "inner-most" component of most operating systems. In Unix-like operating systems, users typically have many choices of command-line interpreters for interactive sessions. The Unix shell is both an interactive command language as well as a scripting programming language, and is used by the operating system as the facility to control (shell script) the execution of the system. Shells created for other operating systems often provide similar functionality.

The first Unix shell was the Thompson shell, sh, written by Ken Thompson at Bell Labs and distributed with Versions 1 through 6 of Unix, from 1971 to 1975. Though rudimentary by modern standards,

it introduced many of the basic features common to all later Unix shells, including piping, simple control structures using if and goto, and filename wildcarding. Thompson shell is not in use currently. Mashey shell, sh, was an upward-compatible version of the Thompson shell, augmented by John Mashey and others and distributed with the Programmer's Workbench UNIX, circa 1975-1977. It focused on making shell programming practical, especially in large shared computing centers. It added shell variables (precursors of environment variables, including the search path mechanism that evolved into $PATH), user-executable shell scripts, and interrupt-handling. Control structures were extended from *if/goto* to *if/then/else/endif, switch/breaksw/endsw*, and *while/end/break/continue*. As shell programming became widespread, these external commands were incorporated into the shell itself for performance. But the most widely distributed and influential of the early Unix shells were the *Bourne shell* and the *C shell*. Both shells have been used as the coding base and model for many derivative and work-alike shells with extended feature sets.

The Bourne shell (sh) is a shell, or command-line interpreter, for computer operating systems. The Bourne shell was the default shell for *Unix Version 7*. Most Unix-like systems continue to have */bin/sh* - which will be the Bourne shell, or a symbolic link or hard link to a compatible shell even when other shells are used by most users. Developed by *Stephen Bourne* at Bell Labs, it was a replacement for the Thompson shell, whose executable file had the same name *sh*. It was released in 1977 in the Version 7 Unix release distributed to colleges and universities. Although it is used as an interactive command interpreter, it was also intended as a scripting language and contains most of the features that are commonly considered to produce structured programs.

2.3.1 BASH

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. Bash is a command processor that typically runs in a text window, where the user types commands that cause actions. Bash can also read commands from a file, called a script. Like all Unix shells, it sup-

ports filename globbing (wildcard matching), piping, here documents, command substitution, variables and control structures for condition-testing and iteration. The keywords, syntax and other basic features of the language were all copied from sh. Other features, e.g., history, were copied from csh and ksh. Bash is a POSIX shell, but with a number of extensions. The shell's name is an acronym for Bourne-again shell, punning on the name of the Bourne shell that it replaces and on the term "born again" that denotes spiritual rebirth in contemporary American Christianity.

Brian Fox began coding Bash on January 10, 1988 after Richard Stallman became dissatisfied with the lack of progress being made by a prior developer. Stallman and the Free Software Foundation (FSF) considered a free shell that could run existing shell scripts so strategic to a completely free system built from BSD and GNU code that this was one of the few projects they funded themselves, with Fox undertaking the work as an employee of FSF. Fox released Bash as a beta, version .99, on June 8, 1989 and remained the primary maintainer until sometime between mid-1992 and mid-1994, when he was laid off from FSF and his responsibility was transitioned to another early contributor, Chet Ramey. [7] Since then, Bash has become by far the most popular shell among users of Linux, becoming the default interactive shell on that operating system's various distributions. [8]

2.3.2 Installing Linux Terminal in Windows

There are multiple ways to get POSIX like environment in Windows. Windows provides Windows Subsystem for Linux (WSL) for Linux based software. Windows Subsystem for Linux (WSL) is a compatibility layer for running Linux binary executables (in ELF format) natively on Windows 10, Windows 11, and Windows Server 2019. In May 2019, WSL 2 was announced, introducing important changes such as a real Linux kernel, through a subset of Hyper-V features. Since June 2019, WSL 2 is available to Windows 10 customers through the Windows Insider program, including the Home edition. WSL is not available to all Windows 10 users by default. It can be installed either by joining the Windows Insider program or manual install. Go to Windows CMD and execute the following statement.

```
wsl --install
```

This command takes rather a bit of time to accomplish installation. Once after installing *wsl*, it can be verified by using

```
wsl uname --all
wsl sudo apt-get update
```

The second command from the above snippet updates Ubuntu subsystem in Windows.

2.4 Basics

The section is going to cover few very basics related to system level information. Following are few commands that help you understand your computer.

- `whoami`: gives the user name.
- `date`: gives the current date along with time.
- `uname -a`: gives all details of your computer.
- `lsb_release - a`: gives information about your OS release.
- `cat /etc/lsb-release`: gives information about OS.
- `cat /etc/issue.net`: gives OS info.
- `cat /etc/debian_version`: gives the distribution name.

For MAC address use `ifconfig -a` find the number adjacent to **HWaddr**. We need to install a utility called *nettools* using `sudo apt install nettools` in Debian based Ubuntu like Linux distributions.

2.4.1 Help

There are many ways to get help in Linux (Ubuntu).

- `man -k search`
- `apropos disk`
- `info ls`

- `man ls`

All the above commands accomplish same goal but provides or retrieves different information. The following code snippet shows as how to search for USB devices in Ubuntu like OS.

```
apropos partition
df
lsblk
usb-devices
```

2.5 Directory Structure

2.5.1 The `root` system

1. The entire Linux directory structure starting at the top (/) root directory.
2. A specific type of data storage format, such as EXT3, EXT4, BTRFS, XFS, and so on. Linux supports almost 100 types of filesystems, including some very old ones as well as some of the newest. Each of these filesystem types uses its own metadata structures to define how the data is stored and accessed.
3. A partition or logical volume formatted with a specific type of filesystem that can be mounted on a specified mount point on a Linux filesystem.

`ls` to list the directory

The command ls is highly useful command that lists the ingredients in the directory. This command is also available in Windows PowerShell but not in CMD.

```
ls
ls > a.txt
rm -f a.txt
```

The above code first lists the contents in *home* directory and *pipe* the output to a file named *a* (which is a text file) later deletes with the help of `rm`.

Name	Desc.
`/` (root filesystem)	The root filesystem is the top-level directory of the filesystem.
`/bin`	The /bin directory contains user executable files.
`/boot`	Contains the static *bootloader* and kernel executable and configuration files required to boot a Linux computer.
`/dev`	This directory contains the device files for every hardware device attached to the system.
`/etc`	Contains the local system configuration files for the host computer.
`/home`	Home directory storage for user files. Each user has a sub-directory in `/home`.
`/lib`	Contains shared library files that are required to boot the system.
`/media`	A place to mount external removable media devices such as USB thumb drives that may be connected to the host.
`/mnt`	A temporary mountpoint for regular filesystems (as in not removable media) that can be used while the administrator is repairing or working on a filesystem.
`/opt`	Optional files such as vendor supplied application programs should be located here.
`/root`	This is not the root (/) filesystem. It is the home directory for the root user.
`/sbin`	System binary files. These are executables used for system administration.
`/tmp`	Temporary directory. Used by the operating system and many programs to store temporary files. Users may also store files here temporarily. Note that files stored here may be deleted at any time without prior notice.
`/usr`	These are shareable, read-only files, including executable binaries and libraries, man files, and other types of documentation.
`/var`	Variable data files are stored here. This can include things like log files, MySQL, and other database files, web server data files, email inboxes, and much more.

Table 2.1: Directory structure in Linux

Working in Directories

- `cd`: takes you to home.
- `cd /`: takes you to root.
- `cd .`: current directory.
- `cd ..`: one step back from exiting directory.

Make & remove directories

- `mkdir`: to create a directory.
- `rm -r`: to remove directory.

2.6 Working with files

2.6.1 Creating and editing files

`touch a.txt` - crates a simple text file (with name *a*). Try with `vi` and `nano` in stead of `touch`.

```
nano ~/Desktop/a.txt
cat ~/Desktop/a.txt
rm ~/Desktop/a.txt
ls ~/Desktop/
```

The above code creates a simple *text* file known as *a* and then verify the contents through `cat` and then later removes the file with the help of `rm`.

2.6.2 Creating multiple files

To create multiple files at a time and editing a given file. Following statement creates *simple* text file.

```
sudo nano simple.txt
```

Suppose you want to make multiple text files, say *file1, file2,..., filen.* You may do as below.

```
sudo touch file1..5.txt
```

```
sudo touch a{0..10}.txt
ls
nano a0.txt
```

2.6.3 Cut, Copy & Paste Files

- `cp {file name or path for file}`: makes replica of the file
- `mv {file name or path for file}`: to cut and create same copy

2.6.4 Find and locate files in directories

`grep`

- `sudo find "name.ext"`: find the files that match file name *name* with extension *.ext*.
- `sudo find / -type f -name "*.xls"`: finds all those files ends with *.xls* extension.
- `sudo find / -size [+/-]700 [c, b, M, G, k]`: finds the files by size.

Use `find --help` for options.

`grep`

- `egrep --color "Mem|Cache" /proc/meminfo` : to call by multiple items.
- `cat /proc/meminfo | grep "Mem"`: to call only by an item.

2.6.5 Contents of files

`cat ipsum.txt | [less/more]`

2.7 Memory Management

- `free`: most simple and easy to use command to check memory usage on linux. Eg. `free -m`. The `m` option displays all data in MBs. You can try with other flags viz., *k, g*.
- `cat /proc/meminfo`: the `/proc` file system does not contain real files. They are rather virtual files that contain dynamic information about the kernel and the system.
- `vmstat -s` : with the `s` option, lays out the memory usage statistics much like the `proc` command. *Try with out option.*
- `top`: generally used to check memory and cpu usage per process. However it also reports total memory usage and can be used to monitor the total RAM usage. The header on output has the required information. The most effective command used to manage processes. For instance, use `k` switch to kill any task or operation.
- `htop`: another command used for process management. You need to install using `sudo apt install htop`.
- `sudo dmidecode -t 10`: give you few of the items.

2.8 Processor Management

- `nproc`: simplest command to know number of processors.
- `lscpu`: a very simple command that list most of the CPU properties.
- `hardinfo | less`: needs installation [`sudo apt install hardinfo`].
- `cpuid`: needs installation [`sudo apt install cpuid`].
- `inxi`: needs installation [`sudo apt install inxi`].

2.8.1 /proc/ directory

- `[less/more] /proc/cpuinfo`

- `cat /proc/cpuinfo | grep processor | wc -l`
- `cat /proc/cpuinfo | grep processor`

2.9 Networking

Every computer is connected to some other computer through a network whether internally or externally to exchange some information. This network can be small as some computers connected in your home or office, or can be large or complicated as in large University or the entire Internet. Maintaining a system's network is a task of System/Network administrator. Their task includes network configuration and troubleshooting. Table 2.2 has few Networking and Troubleshooting commands:

We will learn more about Linux networking in one of the forthcoming sections in this Chapter.

2.10 BASH Programming

The Bash command syntax is a superset of the Bourne shell command syntax. Bash supports brace expansion, command line completion (Programmable Completion), basic debugging and signal handling (using trap) since bash 2.05a among other features. Bash can execute the vast majority of Bourne shell scripts without modification, with the exception of Bourne shell scripts stumbling into fringe syntax behavior interpreted differently in Bash or attempting to run a system command matching a newer Bash builtin, etc. Bash command syntax includes ideas drawn from the KornShell (ksh) and the C shell (csh) such as command line editing, command history (`history` command), the directory stack, the `$RANDOM` and `$PPID` variables, and POSIX command substitution syntax $(...).

When a user presses the tab key within an interactive command-shell, Bash automatically uses command line completion, since beta version 2.04, to match partly typed program names, filenames and variable names. The Bash command-line completion system is very flexible

and customizable, and is often packaged with functions that complete arguments and filenames for specific programs and tasks.

Bash's syntax has many extensions lacking in the Bourne shell. Bash can perform integer calculations ("arithmetic evaluation") without spawning external processes. It uses the ((...)) command and the $((...)) variable syntax for this purpose. Its syntax simplifies I/O redirection. For example, it can redirect standard output (stdout) and standard error (stderr) at the same time using the & > operator. This is simpler to type than the Bourne shell equivalent 'command >file 2 >&1'. Bash supports process substitution using the <(command) and >(command)syntax, which substitutes the output of (or input to) a command where a filename is normally used.

When using the 'function' keyword, Bash function declarations are not compatible with Bourne/Korn/POSIX scripts (the KornShell has the same problem when using 'function'), but Bash accepts the same function declaration syntax as the Bourne and Korn shells, and is POSIX-conformant. Because of these and other differences, Bash shell scripts are rarely runnable under the Bourne or Korn shell interpreters unless deliberately written with that compatibility in mind, which is becoming less common as Linux becomes more widespread. But in POSIX mode, Bash conforms with POSIX more closely. Bash supports here documents. Since version 2.05b Bash can redirect standard input (stdin) from a "here string" using the «<operator. Bash 3.0 supports in-process regular expression matching using a syntax reminiscent of Perl.

In February 2009, Bash 4.0 introduced support for associative arrays. Associative array indices are strings, in a manner similar to AWK or Tcl. They can be used to emulate multidimensional arrays. Bash 4 also switches its license to GPL-3.0-or-later; some users suspect this licensing change is why MacOS continues to use older versions. Apple finally stopped using Bash in their operating systems with the release of MacOS Catalina in 2019.

2.10.1 Redirections

Below statement cause the *stderr* output of a program to be written to a file.

```
grep da * 2> grep-errors.txt
```

Here, a file called `grep-errors.txt` will be created and it will contain what you would see the stderr portion of the output of the `grep da *` command. Below statement cause the `stderr` output of a program to be written to the same `filedescriptor` than `stdout`.

```
grep da * 1>&2
```

Here, the `stdout` portion of the command is sent to `stderr`, you may notice that in different ways. The below statement cause the `stderr` output of a program to be written to the same `filedescriptor` than `stdout`.

```
grep * 2>&1
```

Here, the `stderr` portion of the command is sent to stdout, if you pipe to less, you'll see that lines that normally "disappear" (as they are written to `stderr`) are being kept now (because they're on `stdout`). This will place every output of a program to a file. This is suitable sometimes for cron entries, if you want a command to pass in absolute silence.

```
rm -f \$(find / -name core) \&> /dev/null
```

This (thinking on the cron entry) will delete every file called "core" in any directory. Notice that you should be pretty sure of what a command is doing if you are going to wipe it's output.

2.10.2 Pipes

This section explains in a very simple and practical way how to use pipes, nd why you may want it.

Pipes let you use (very simple, I insist) the output of a program as the input of another one. This is very simple way to use pipes.

```
ls -l | sed -e "s/[aeio]/u/g"
```

Here, the following happens: first the command `ls -l` is executed, and it's output, instead of being printed, is sent (piped) to the sed program, which in turn, prints what it has to. Probably, the following is a more difficult way to do `ls -l *.txt`, but it is here for illustrating pipes, not for solving such listing dilema.

```
ls -l | grep "\.txt\$"
```

Here, the output of the program `ls -l` is sent to the grep program, which, in turn, will print lines which match the regex "
.txt$".

2.10.3 Data types & Variables

We can use `echo` for printing to *stdio*. For example, the statement `echo "hello! word"`, prints `hello! word` in the console.

```
#!/bin/bash
STR="Hello World!"; mynum=21
echo $STR $mynum
```

To delete any variable use `unset`. You can use variables as in any programming languages. There are no data types. A variable in bash can contain a number, a character, a string of characters. You have no need to declare a variable, just assigning a value to its reference will create it.

Sample: Hello World! using variables

```
#!/bin/bash
STR="Hello World!"
echo \$STR
```

Line 2 creates a variable called `STR` and assigns the string "Hello World!" to it. Then the VALUE of this variable is retrieved by putting the "$" in at the beginning. Please notice (try it!) that if you don't use the "$" sign, the output of the program will be different, and probably not what you want it to be.

Sample: A very simple backup script (little bit better)

```
#!/bin/bash
OF=/var/my-backup-\$(date +\%Y\%m\%d).tgz
tar -cZf \$OF /home/me/
```

This script introduces another thing. First of all, you should be familiarized with the variable creation and assignation on line 2. Notice the expression "$(date +%Y%m%d)". If you run the script you'll notice that it runs the command inside the parenthesis, capturing its output. Notice that in this script, the output filename will be different every day, due to the format switch to the date command `(+%Y%m%d)`. You can change this by specifying a different format.

Local variables can be created by using the keyword local.

```
#!/bin/bash
HELLO=Hello
function hello {
        local HELLO=World
  echo $HELLO
}
echo $HELLO
hello
echo \$HELLO
```

This example should be enought to show how to use a local variable.

2.10.4 Data types

Unlike many other programming languages, Bash does not segregate its variables by "type". Essentially, Bash variables are character strings, but, depending on context, Bash permits arithmetic operations and comparisons on variables. The determining factor is whether the value of a variable contains only digits.

```
a=2334
let "a += 1"
echo "a = \$a "
```

The output will be 2335, which is still an integer. Suppose if we convert this into a string and perform little math:

```
b=${a/23/BB}              # Substitute "BB" for "23".
                          # This transforms $b into a
                          ↪  string.
echo "b = $b"             # b = BB35
declare -i b              # Declaring it an integer
↪  doesn't help.
echo "b = $b"             # b = BB35

let "b += 1"              # BB35 + 1
echo "b = \$b"             # b = 1
echo                      # Bash sets the "integer value"
```

The value of "b" is *one* now, because BASH treats numeric value of a string or character as zero.

Null & Undeclared values

```
e=''                      # ... Or e="" ... Or e=
echo "e = $e"             # e =
let "e += 1"              #
echo "e = $e"             #
echo                      # Null variable transformed into
↪  an integer.
```

Arithmetic operations allowed on a null variable? yes. The value of "e" after addition will be 1.

```
# What about undeclared variables?
echo "f = $f"             # f =
let "f += 1"              #
echo "f = $f"             # f = 1
echo                      # Undeclared variable
↪  transformed into an integer.
```

Arithmetic operations allowed on undeclared values. The value of f after numerical operation will be 1. And the variable will be transformed into integer.

2.10.5 Operators

String comparison operators

(1) s1 = s2

(2) s1 != s2

(3) s1 < s2

(4) s1 > s2

(5) -n s1

(6) -z s1

Interpretation

(1) s1 matches s2

(2) s1 does not match s2

(3) s1 less than s2

(4) s1 greater than s2

(5) s1 is not null (contains one or more characters)

(6) s1 is null

String comparison examples

Comparing two strings.

```
#!/bin/bash
S1='string'
S2='String'
if [ $S1=$S2 ];
then
        echo "S1('$S1') is not equal to S2('$S2')"
else
```

```
        echo "S1('$S1') is not equal to S2('$S2')"
fi
```

Arithmetic operators

+

-

*

/

% (remainder)

Arithmetic relational operators

-lt (<)

-gt (>)

-le (<=)

-ge (>=)

-eq (==)

-ne (!=)

C programmer's should simple map the operator to its corresponding parenthesis.

2.10.6 Conditionals

Conditionals let you decide whether to perform an action or not, this decision is taken by evaluating an expression. Conditionals have many forms. The most basic form is: `if` expression then statement where "statement" is only executed if "expression" evaluates to *true*. "2<1" is an expresion that evaluates to *false*, while "2>1" evaluates to *true*. Conditionals have other forms such as: *if expression then statement1 else statement2*. Here "statement1" is executed if "expression" is true, otherwise "statement2" is executed. Yet another form of conditionals is: *if expression1 then statement1 else if expression2 then statement2*

else statement3. In this form there's added only the *ELSE IF 'expression2' THEN 'statement2'* which makes *statement2* being executed if *expression2* evaluates to *true*. The base for the `if` constructions in bash is this:

```
if [expression];

then

code if 'expression' is true.

fi
```

Sample: Basic conditional example if .. then

```
#!/bin/bash
if [ "foo" = "foo" ]; then
        echo expression evaluated as true
fi
```

The code to be executed if the expression within braces is true can be found after the 'then' word and before 'fi' which indicates the end of the conditionally executed code.

Sample: Basic conditional example if .. then ... else

```
#!/bin/bash
if [ "foo" = "foo" ]; then
        echo expression evaluated as true
else
        echo expression evaluated as false
fi
```

Sample: Conditionals with variables

```
#!/bin/bash
T1="foo"
T2="bar"
if [ "$T1" = "$T2" ]; then
        echo expression evaluated as true
else
```

```
        echo expression evaluated as false
fi
```

2.10.7 Loops for, while and until

The *for loop* is a little bit different from other programming languages. Basically, it let's you iterate over a series of "words" within a string. The *while* executes a piece of code if the control expression is true, and only stops when it is false (or a explicit break is found within the executed code. The *until* loop is almost equal to the while loop, except that the code is executed while the control expression evaluates to false. If you suspect that while and until are very similar you are right.

For sample

```
#!/bin/bash
for i in $( ls ); do
        echo item: $i
done
```

On the second line, we declare i to be the variable that will take the different values contained in $(ls). The third line could be longer if needed, or there could be more lines before the done. "done" indicates that the code that used the value of $i has finished and $i can take a new value. This script has very little sense, but a more useful way to use the for loop would be to use it to match only certain files on the previous example

C-like for

fiesh suggested adding this form of looping. It's a for loop more similar to C/perl... for.

```
#!/bin/bash
for i in `seq 1 10`;
do
        echo \$i
done
```

While sample

```
#!/bin/bash
COUNTER=0
while [  $COUNTER -lt 10 ]; do
        echo The counter is $COUNTER
        let COUNTER=COUNTER+1
done
```

This script "emulates" the well known (C, Pascal, perl, etc) "for" structure.

Until sample

```
#!/bin/bash
COUNTER=20
until [  $COUNTER -lt 10 ]; do
        echo COUNTER $COUNTER
        let COUNTER-=1
done
```

2.10.8 Functions

As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion. Declaring a function is just a matter of writing function `my_func{ my_code }`. Calling a function is just like calling another program, you just write its name.

Functions sample

```
#!/bin/bash
function quit {
        exit
}
function hello {
        echo Hello!
}
hello
```

```
quit
echo foo
```

Lines 2-4 contain the "quit" function. Lines 5-7 contain the "hello" function If you are not absolutely sure about what this script does, please try it!. Notice that a functions don't need to be declared in any specific order. When running the script you'll notice that first: the function 'hello' is called, second the 'quit' function, and the program never reaches line 10.

Functions with parameters sample

```
#!/bin/bash
function quit {
        exit
}
function e {
        echo \$1
}
e Hello
e World
quit
echo foo
```

This script is almost identically to the previous one. The main difference is the funcion "e". This function, prints the first argument it receives. Arguments, within functions, are treated in the same manner as arguments given to the script.

2.11 Data Analysis

BASH is not meant for data analysis but it can. BASH is one of the default programming languages used through Hadoop streaming for data processing. [9] [10]

2.11.1 Arithmetic calculations

```
a=10
b=20
```

```
ans1=$((a+b))
ans2=$((a-b))
ans3=$((a*b))
ans4=$((a/b))
echo "addition of $a + $b = $ans1"
echo "subtraction of $a + $b = $ans2"
echo "multiplication of $a + $b = $ans3"
echo "division of $a + $b = $ans4"
```

Save the above code in any plain text file but save it as `.sh` extension. Then the file becomes an executable script. It is possible to execute it by simply using the very famous `./` followed by the file name (with extension) in the shell.

```
expr 12 + 8
echo \$(( 12 + 8 ))
echo 12 + 8 | bc
echo 12 / 8 | bc -l
```

Notice the spaces before and after the numbers. All the above statements yields same results except the last one. In last statement the division needs to be printed in decimal values that is why `bc` requires an option `-l`. Following is the example code for simple arithmetic calculation.

```
echo "Enter two numbers"
read x
read y
sum=`expr $x + $y`
echo "Sum = \$sum"
```

The command `read` is useful to scan the user inputs in the shell. The output for the above program will be as follows.

```
###:~/bash_scripts\$ sh arith.sh
Enter two numbers
1
2
Sum = 3
```

Following code can find the length of a string.

```
x=kamakshaiah
len=`expr length \$x`
echo \$len
```

The result will be 10.

2.11.2 One Dimensional Arrays

Code sample for printing array elements.

```
a=( 1 2 3 4 5 )
for i in ${a[@]}; do
echo $i
done
```

The following code process the sum of array elements

```
#sum of array shell script
arr=(1 2 3 4 5)
sum=0
for i in ${arr[@]}
do
    sum=`expr $sum + $i`
done
echo $sum
```

The output will be 15. The following code calculates arithmetic average for static array data.

```
a=(1 2 3 4 5 6)
sum=0
cnt=0
for i in ${a[@]}; do
sum=`expr $sum + $i`
let "cnt++"
done
echo $sum/\$cnt | bc -l
```

The above code prints 3.5000... The following code sample *read*s data from user and then calculates arithmetic average.

```
read -a arr
```

```
sum=0
cnt=0
for i in ${arr[@]}; do
sum=`expr $sum + $i`
let "cnt++"
done
echo $sum/\$cnt | bc -l
```

The code prints 3.5000... for numbers 1 to 6.

2.11.3 Multidimensional arrays

Consider a cross tab of 2×2 order such as below:

This can be made or created using the following sample code.

```
declare -A arr
arr[0, 0]=1
arr[0, 1]=2
arr[1, 0]=3
arr[1, 1]=4
echo \${arr[@]}
```

We can use the following code for row-wise and column-wise sums

2.12 Linux Networking

2.12.1 Beowulf cluster

We will see as how to make two node Beowulf cluster using virtualization and GNU/Linux OS. I will be using *VirtualBox* and *Ubuntu OS* for erecting a 2 node cluster. [11]

VirtualBox is a powerful x86 and AMD64/Intel64 virtualization product for enterprise as well as home use. Not only is VirtualBox an extremely feature rich, high performance product for enterprise customers, it is also the only professional solution that is freely available as Open Source Software under the terms of the GNU General Public License (GPL) version 2.

Ubuntu is a Linux distribution based on Debian and composed mostly of free and open-source software. Ubuntu is officially released in three editions: Desktop, Server, and Core for Internet of things devices and robots. All the editions can run on the computer alone, or in a virtual machine. Ubuntu is a popular operating system for cloud computing, with support for OpenStack.

What is Beowulf cluster?

A Beowulf cluster is a computer cluster of what are normally identical, commodity-grade computers networked into a small local area network with libraries and programs installed which allow processing to be shared among them. The result is a high-performance parallel computing cluster from inexpensive personal computer hardware. The name Beowulf originally referred to a specific computer built in 1994 by Thomas Sterling and Donald Becker at NASA. [12] The name "Beowulf" comes from the Old English epic poem of the same name. [13]

There are few misconceptions in the academics and industry such as Beowulf cluster is a cluster with some special and highly computing resources. It is not .... No particular piece of software defines a cluster as a Beowulf. Typically only free and open source software is used, both to save cost and to allow customisation. Most Beowulf clusters run a Unix-like operating system, such as BSD, Linux, or Solaris. Commonly used parallel processing libraries include Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). Both of these permit the programmer to divide a task among a group of networked computers, and collect the results of processing. Examples of MPI software include Open MPI or MPICH. There are additional MPI implementations available.

A description of the Beowulf cluster, from the original "how-to", which was published by Jacek Radajewski and Douglas Eadline under the Linux Documentation Project in 1998: [14]

> "Beowulf is a multi-computer architecture which can be used for parallel computations. It is a system which usually consists of one server node, and one or more client

nodes connected via Ethernet or some other network. It is a system built using commodity hardware components, like any PC capable of running a Unix-like operating system, with standard Ethernet adapters, and switches. It does not contain any custom hardware components and is trivially reproducible. Beowulf also uses commodity software like the FreeBSD, Linux or Solaris operating system, Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). The server node controls the whole cluster and serves files to the client nodes ...

One of the main differences between Beowulf and a Cluster of Workstations (COW) is that Beowulf behaves more like a single machine rather than many workstations. In most cases client nodes do not have keyboards or monitors, and are accessed only via remote login or possibly serial terminal. Beowulf nodes can be thought of as a CPU and memory package which can be plugged into the cluster, just like a CPU or memory module can be plugged into a motherboard.

Beowulf is not a special software package, new network topology, or the latest kernel hack. Beowulf is a technology of clustering computers to form a parallel, virtual supercomputer. Although there are many software packages such as kernel modifications, PVM and MPI libraries, and configuration tools which make the Beowulf architecture faster, easier to configure, and much more usable, one can build a Beowulf class machine using a standard Linux distribution without any additional software. ***If you have two networked computers which share at least the /home file system via NFS, and trust each other to execute remote shells (rsh), then it could be argued that you have a simple, two node Beowulf machine***."

Notes

[5] https://en.wikipedia.org/wiki/Linux

[6] Linus Torvald appeard to have been influenced by Tanenbaum through his books and teachings.

[7] Chet Ramey (October 31, 2010), Dates in your Computerworld interview, retrieved October 31, 2010

[8] Bash has also been ported to Microsoft Windows and distributed with Cygwin and MinGW, to DOS by the DJGPP project, to Novell NetWare and to Android via various terminal emulation applications. Microsoft announced during the 2016 Build Conference that Windows 10 has added a Linux subsystem that fully supports Bash and other Ubuntu binaries running natively in Windows. Please read about this at https://blogs.windows.com

[9] Read more about Hadoop streaming from https://hadoop.apache.org/docs/r1.2.1/streaming.html.

[10] Hadoop steaming is a utility comes up with Hadoop distribution. Hadoop streaming allows users to write MapReduce programs using few programming languages such as Python, PHP, Ruby, Perl, BASH, etc. Read more about language support at https://www.tutorialscampus.com/hadoop/streaming.htm.

[11] Visit https://www.virtualbox.org/ for more information.

[12] Becker, Donald J and Sterling, Thomas and Savarese, Daniel and Dorband, John E and Ranawak, Udaya A and Packer, Charles V (1995). "BEOWULF: A parallel workstation for scientific computation". Proceedings, International Conference on Parallel Processing. 95.

[13] See Francis Barton Gummere's 1909 translation, reprinted (for example) in Beowulf. Francis B. Gummere (translator). Hayes Barton Press (published c. 1910). 1909. p. 20. ISBN 9781593773700. Retrieved 2014-01-16.

[14] Radajewski, Radajewski; Eadline, Douglas (22 November 1998). "Beowulf HOWTO". ibiblio.org. v1.1.1. Retrieved 8 June 2021.

Notes

[5] https://en.wikipedia.org/wiki/Linux

[6] Linus Torvald appeard to have been influenced by Tanenbaum through his books and teachings.

[7] Chet Ramey (October 31, 2010), Dates in your Computerworld interview, retrieved October 31, 2010

[8] Bash has also been ported to Microsoft Windows and distributed with Cygwin and MinGW, to DOS by the DJGPP project, to Novell NetWare and to Android via various terminal emulation applications. Microsoft announced during the 2016 Build Conference that Windows 10 has added a Linux subsystem that fully supports Bash and other Ubuntu binaries running natively in Windows. Please read about this at https://blogs.windows.com

[9] Read more about Hadoop streaming from https://hadoop.apache.org/docs/r1.2.1/streaming.html.

[10] Hadoop steaming is a utility comes up with Hadoop distribution. Hadoop streaming allows users to write MapReduce programs using few programming languages such as Python, PHP, Ruby, Perl, BASH, etc. Read more about language support at https://www.tutorialscampus.com/hadoop/streaming.htm.

[11] Visit https://www.virtualbox.org/ for more information.

[12] Becker, Donald J and Sterling, Thomas and Savarese, Daniel and Dorband, John E and Ranawak, Udaya A and Packer, Charles V (1995). "BEOWULF: A parallel workstation for scientific computation". Proceedings, International Conference on Parallel Processing. 95.

[13] See Francis Barton Gummere's 1909 translation, reprinted (for example) in Beowulf. Francis B. Gummere (translator). Hayes Barton Press (published c. 1910). 1909. p. 20. ISBN 9781593773700. Retrieved 2014-01-16.

[14] Radajewski, Radajewski; Eadline, Douglas (22 November 1998). "Beowulf HOWTO". ibiblio.org. v1.1.1. Retrieved 8 June 2021.

2.13 Questions

2.13.1 Short answer questions

1. What is Linux OS?
2. What is BASH?
3. What is root?

2.13.2 Essay questions

1. Elaborte history of Linux OS.
2. How do you do help on BASH statements in Linux terminal.
3. Explain about installing Linux terminal in Windows.
4. Illustrate root filesystem in Linux.
5. Give an elaborate note on working with file system in Linux terminal.
6. Write about various commands related to memory management in Linux.
7. Write about various commands related to process management in Linux.
8. Write about various commands related to networking in Linux.

ifconfig	Display and manipulate route and network interfaces.
ip	It is a replacement of ifconfig command.
traceroute	Network troubleshooting utility.
tracepath	Similar to traceroute but doesn't require root privileges.
ping	To check connectivity between two nodes.
netstat	Display connection information.
ss	It is a replacement of netstat.
dig	Query DNS related information.
nslookup	Find DNS related query.
route	Shows and manipulate IP routing table.
host	Performs DNS lookups.
arp	View or add contents of the kernel's ARP table.
iwconfig	Used to configure wireless network interface.
hostname	To identify a network name.
curl or wget	To download a file from internet.
mtr	Combines ping and tracepath into a single command.
whois	Will tell you about the website's `whois`.
ifplugstatus	Tells whether a cable is plugged in or not.

Table 2.2: Few network commands

1	2
3	4

Table 2.3: 2 cross tab

Chapter 3

About Hadoop

Apache Hadoop is a collection of open source software utilities that facilitate using a network of many computers to solve problems involving massive amounts of data and computation. It provides a software framework for distributed storage and processing of big data using the MapReduce programming model. Hadoop was originally designed for computer clusters built from commodity hardware, which is still the common use. It has since also found use on clusters of higher-end hardware. All the modules in Hadoop are designed with a fundamental assumption that hardware failures are common occurrences and should be automatically handled by the framework.

The core of Apache Hadoop consists of a storage part, known as Hadoop Distributed File System (HDFS), and a processing part which is a MapReduce programming model. Hadoop splits files into large blocks and distributes them across nodes in a cluster. It then transfers packaged code into nodes to process the data in parallel. This approach takes advantage of data locality, where nodes manipulate the data they have access to. This allows the data set to be processed faster and more efficiently than it would be in a more conventional supercomputer architecture that relies on a parallel file system where computation and data are distributed via high-speed networking.

3.1 History

According to its co-founders, Doug Cutting and Mike Cafarella, the genesis of Hadoop was the Google File System paper that was published in October 2003. This paper spawned another one from Google – "MapReduce: Simplified Data Processing on Large Clusters". Development started on the Apache Nutch project, but was moved to the new Hadoop sub-project in January 2006. Doug Cutting, who was working at Yahoo! at the time, named it after his son's toy elephant.[20] The initial code that was factored out of Nutch consisted of about 5,000 lines of code for HDFS and about 6,000 lines of code for MapReduce. In March 2006, Owen O'Malley was the first committer to add to the Hadoop project; Hadoop 0.1.0 was released in April 2006. It continues to evolve through contributions that are being made to the project.

3.2 Features

Following are few key features which make Hadoop more reliable to use, an industry favorite, and the most powerful Big Data platform.

1. ***Open Source***: Hadoop is open-source, which means it is free to use. Since it is an open-source project the source-code is available online for anyone to understand it or make some modifications as per their industry requirement.

2. ***Highly Scalable Cluster***: Hadoop is a highly scalable model. A large amount of data is divided into multiple inexpensive machines in a cluster which is processed parallelly. the number of these machines or nodes can be increased or decreased as per the enterprise's requirements. In traditional RDBMS(Relational DataBase Management System) the systems can not be scaled to approach large amounts of data.

3. ***Fault Tolerance***: Hadoop uses commodity hardware(inexpensive systems) which can be crashed at any moment. In Hadoop data is replicated on various *DataNodes* in a Hadoop cluster which ensures the availability of data if somehow any of your systems got crashed. You can read all of the data from a single machine

if this machine faces a technical issue data can also be read from other nodes in a Hadoop cluster because the data is copied or replicated by default. By default, Hadoop makes 3 copies of each file block and stored it into different nodes. This replication factor is configurable and can be changed by changing the replication property in the *hdfs-site.xml* file.

4. ***High Availability***: Fault tolerance provides High Availability in the Hadoop cluster. High Availability means the availability of data on the Hadoop cluster. Due to fault tolerance in case if any of the DataNode goes down the same data can be retrieved from any other node where the data is replicated. The High available Hadoop cluster also has 2 or more than two Name Node i.e. Active NameNode and Passive NameNode also known as stand by NameNode. In case if Active NameNode fails then the Passive node will take the responsibility of Active Node and provide the same data as that of Active NameNode which can easily be utilized by the user.

5. ***Cost-Effective***: Hadoop is open-source and uses cost-effective commodity hardware which provides a cost-efficient model, unlike traditional Relational databases that require expensive hardware and high-end processors to deal with Big Data. The problem with traditional Relational databases is that storing the Massive volume of data is not cost-effective, so the company's started to remove the Raw data. which may not result in the correct scenario of their business. Means Hadoop provides us 2 main benefits with the cost one is it's open-source means free to use and the other is that it uses commodity hardware which is also inexpensive.

6. ***Flexibility:*** Hadoop is designed in such a way that it can deal with any kind of dataset like structured(MySql Data), Semi-Structured(XML, JSON), Un-structured (Images and Videos) very efficiently. This means it can easily process any kind of data independent of its structure which makes it highly flexible. It is very much useful for enterprises as they can process large datasets easily, so the businesses can use Hadoop to analyze valuable insights of data from sources like social media,

email, etc. With this flexibility, Hadoop can be used with log processing, Data Warehousing, Fraud detection, etc.

7. ***Easy to Use***: Hadoop is easy to use since the developers need not worry about any of the processing work since it is managed by the Hadoop itself. Hadoop ecosystem is also very large comes up with lots of tools like Hive, Pig, Spark, HBase, Mahout, etc.

8. ***Data Locality:*** The concept of Data Locality is used to make Hadoop processing fast. In the data locality concept, the computation logic is moved near data rather than moving the data to the computation logic. The cost of Moving data on HDFS is costliest and with the help of the data locality concept, the bandwidth utilization in the system is minimized.

9. ***Faster Data Processing:*** Hadoop uses a distributed file system to manage its storage i.e. HDFS(Hadoop Distributed File System). In DFS(Distributed File System) a large size file is broken into small size file blocks then distributed among the Nodes available in a Hadoop cluster, as this massive number of file blocks are processed parallelly which makes Hadoop faster, because of which it provides a High-level performance as compared to the traditional DataBase Management Systems.

3.3 Hadoop software library

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

The project includes these modules:

1. Hadoop Common: The common utilities that support the other Hadoop modules.

2. Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
3. Hadoop YARN: A framework for job scheduling and cluster resource management.
4. Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

Other Hadoop-related projects at Apache include:

1. *Ambari™*: A web-based tool for provisioning, managing, and monitoring Apache Hadoop clusters which includes support for Hadoop HDFS, Hadoop MapReduce, Hive, HCatalog, HBase, ZooKeeper, Oozie, Pig and Sqoop. Ambari also provides a dashboard for viewing cluster health such as heatmaps and ability to view MapReduce, Pig and Hive applications visually alongwith features to diagnose their performance characteristics in a user-friendly manner.
2. *Avro™*: A data serialization system.
3. *Cassandra™*: A scalable multi-master database with no single points of failure.
4. *Chukwa™*: A data collection system for managing large distributed systems.
5. *HBase™*: A scalable, distributed database that supports structured data storage for large tables.
6. *Hive™*: A data warehouse infrastructure that provides data summarization and ad hoc querying.
7. *Mahout™*: A Scalable machine learning and data mining library.
8. *Pig™*: A high-level data-flow language and execution framework for parallel computation.
9. *Spark™*: A fast and general compute engine for Hadoop data. Spark provides a simple and expressive programming model that supports a wide range of applications, including ETL, machine learning, stream processing, and graph computation.

10. $\underline{Tez^{TM}}$:A generalized data-flow programming framework, built on Hadoop YARN, which provides a powerful and flexible engine to execute an arbitrary DAG of tasks to process data for both batch and interactive use-cases. Tez is being adopted by $Hive^{TM}$, Pig^{TM} and other frameworks in the Hadoop ecosystem, and also by other commercial software (e.g. ETL tools), to replace $Hadoop^{TM}$ MapReduce as the underlying execution engine.

11. $\underline{ZooKeeper^{TM}}$: A high-performance coordination service for distributed applications.

3.4 Architecture

Hadoop consists of the Hadoop Common package, which provides file system and operating system level abstractions, a MapReduce engine (either MapReduce/MR1 or YARN/MR2) and the Hadoop Distributed File System (HDFS). The Hadoop Common package contains the Java Archive (JAR) files and scripts needed to start Hadoop.

For effective scheduling of work, every Hadoop-compatible file system should provide location awareness, which is the name of the rack, specifically the network switch where a worker node is. Hadoop applications can use this information to execute code on the node where the data is, and, failing that, on the same rack/switch to reduce backbone traffic. HDFS uses this method when replicating data for data redundancy across multiple racks. This approach reduces the impact of a rack power outage or switch failure; if any of these hardware failures occurs, the data will remain available.

A small Hadoop cluster includes a single master and multiple worker nodes. The master node consists of a Job Tracker, Task Tracker, NameNode, and DataNode. A slave or worker node acts as both a DataNode and TaskTracker, though it is possible to have data-only and compute-only worker nodes. These are normally used only in nonstandard applications. Hadoop requires Java Runtime Environment (JRE) 1.6 or higher. The standard startup and shutdown scripts require that Secure Shell (SSH) be set up between nodes in the cluster.

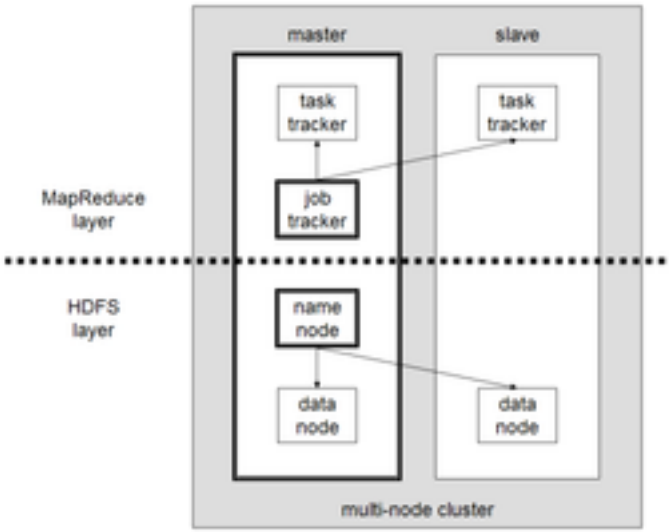


Figure 3.1: Hadoop Architecture

In a larger cluster, HDFS nodes are managed through a dedicated NameNode server to host the file system index, and a secondary NameNode that can generate snapshots of the namenode's memory structures, thereby preventing file-system corruption and loss of data. Similarly, a standalone JobTracker server can manage job scheduling across nodes. When Hadoop MapReduce is used with an alternate file system, the NameNode, secondary NameNode, and DataNode architecture of HDFS are replaced by the file-system-specific equivalents.

3.5 Hadoop - Installation

The present release of Hadoop is 2.7.2 which is in continution to the previous stabel release 2.7.1. There is excellent source for documentation on as how to install hadoop in Linux machines at https://hadoop.apache.org/docs/r2.7.2/index.html. There are mainly three components in Hadoop they are Hadoop-Common, HDFS, YARN.

1. *Hadoop-Common*: Used to access WebHDFS via a proxy server. Also useful along with Hadoop compatible File System (HCFS).
2. *HDFS*: Supports POSIX-style file system. [15]
3. *YARN*: Supoorts administration like write, modify, kill operations. Stores generic and application specific information. Manages queues.

Hadoop can be installed in two different ways, namely *Single Node Cluster (SNC)* and *Multinode Cluster (MC)*. You may visit https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/SingleCluster.html for SNC and https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/ClusterSetup.html for MC.

This manual deals with SNC setup. The software is tested through 2 core, 64 bit Ubuntu OS version 16.04 running through genuine Intel processor of 2.9 GHz speed. The following is the procedure to install Hadoop.

3.6 Installation of JAVA

Java is one of the important prerequisites for setting up Hadoop. There are number of online resources that provide procedures as how to setup Java in Ubuntu. The following is the procedure to install Java in Ubuntu Terminal. [16]

```
sudo apt-get install default-jre default-jdk
java -version
```

Uninstall previous version or use `update-alternatives --config java` to set right or correct Java installation. You may also check Java installation by doing `which java` and `whereis java`.

3.6.1 Update JAVA HOME variable

Test if java is running or not with the help of `java`, `javac` these commands must be able to return certain help information such as how to use commands else your installation is wrong. One of the

reasons might be that, the system might not be able to spot out Java installation directory. In such case, do as below to update Java Home variable.

```
export JAVA_HOME=/usr/lib/jvm/java-7-openjdk
export PATH=\$PATH:/usr/lib/jvm/java-7-openjdk/bin
```

The above code is applicable only for `default-jdk`. There is also another way to install `default-jdk` and it is through *Ubuntu Software Center*. From 14.04 Ubuntu has Unity Desktop Environment, where you may be able to get the quick menu by executing *Alt+F1* the left side Unity task bar appear in which you may be able to find Ubuntu Software Center. Ubuntu Software Center has nice searching arrangements, search for *open jdk* and try to install.

3.7 Creating Hadoop User Group

It is always better to have a seperate user group for Hadoop work for many of the user commands used through Hadoop or more or less similar with certain native BASH commands. So, it is always advisable to have Hadoop installed through a separate working user name. The following is the procedure to create hadoop user.

```
sudo addgroup hadoop
sudo adduser --ingroup hadoop hduser
```

While executing second statement the computer might trigger for password, you might give a simple and easy-to-remember word for that. Anyway, this password will be suppressed through SSH implementation which is explained in the subsequent section. Now this is time to give root privileges for newly created to user (hduser).

Open a separate terminal (Ctl+Alt+T) and do as below:

```
sudo su root
sudo gedit etc/sudoers
```

The above code opens a text file known as *sudoers* in which find the line

```
root ALL=(ALL:ALL) ALL
```

under "#User previlege specification" and insert the following line

```
hduser ALL=(ALL:ALL) ALL
```

3.8 Install and configure *open-ssh-server*

OpenSSH is a freely available version of the Secure Shell (SSH) protocol family of tools for remotely controlling, or transferring files between, computers. OpenSSH can use many authentication methods, including plain password, public key, and *Kerberos* tickets. [17] There is certain source to manage authentication mechanisms through *openssh* at https://help.ubuntu.com/lts/serverguide/openssh-server.html. Visit http://www.openssh.com/ For time being you may do as follows:

```
sudo apt-get install openssh-server
sudo su hduser
ssh-keygen -t rsa -P ""
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

The first statement installs $openssh - server$ in the computer. Second statement lets you change the user (from your default user to *hduser*). Third statement creates *rsa* (public) key [18] and last statement copies the public key from $\$HOME/.ssh$ directory to another forlder $authorized_keys$ in the same directory.

At this point you may reboot the system by simply executing command `reboot` at the Terminal. However, this optional, you may forward without rebooting your machine, as well. There is another task that few technicians do at this point of time i.e. disabling IPV6. This is also optional Hadoop work with both IPV4 and IPV6 and the performance differences are minimal.

3.9 Download & Setup Hadoop

Go to or visit https://hadoop.apache.org/releases.html. Download binaries of your interest. If you click on *binaries* that may redirects to another page just as https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz. You may

find another link, but this time it is not URI, a direct link for Hadoop binaries. Hadoop 3.3.1 can be downloaded at https://dlcdn.apache.org/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz. This *url* was active at the time of writing this manual. [19] After downloading do as below:

1. copy the *.tar.gz* file to Desktop
2. Extract the file
3. Copy the extracted file to */usr/local/hadoop*

Assume that your extracted files are at *Desktop folder* i.e. */home-/Desktop/* in your user account. Then you might execute the below statement in the Terminal.

```
sudo mv /home/Desktop/your..hadoop..folder/
↪  /usr/local/hadoop
```

In the above code `your..hadoop..folder/` is your hadoop folder name (extract). Now we have created a folder under the directory */usr/local* with a name *hadoop* in which our Hadoop files exists. Now since, we had created a folder it is better to give folder permissions to the user *hduser* because whatever that we do later uses the stuff living under this folder.

```
sudo chown hduser:hadoop -R /usr/local/hadoop
```

3.9.1 Working directories for *NameNode* and *DataNode*

The *NameNode* is the centerpiece of an HDFS file system. It keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept. A *DataNode* stores data in the Hadoop File System (HFS), is also known as distributed file system (HDFS). The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. These nodes of file system use certain temporary folder to track, change and

execute user requests. Do as below to create a temporary directory for NameNode and DataNode.

```
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/namenode
sudo mkdir -p /usr/local/hadoop_tmp/hdfs/datanode
sudo chown hduser:hadoop -R /usr/local/hadoop_tmp/
```

The last (third) statement above gives both group, as well as, user privileges to newly created to folder $hadoop_tmp$.

3.9.2 Configuring Hadoop

./bashrc

.bashrc is the file that is executed when you open a terminal window (like gnome terminal) for text-based interaction with the system from the command line. You should open the file .bashrc in your home directory in a text editor program like *vi* or *gedit*. Once, you open *./bashrc* go to the end of the file and put the following code.

```
export JAVA_HOME=/usr/lib/jvm/java-8-oracle
export HADOOP_HOME=/usr/local/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIRECTORY=
↪  $HADOOP_HOME/lib/native
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib"
export PATH=$PATH:/usr/local/hadoop/bin
```

3.9.3 *hadoop-env.sh*

Hadoop need to know as where is Java installed in the computer. So go to the directory */usr/local/hadoop/env/hadoop/* then open *hadoop-env.sh* and find the line $JAVA\_HOME$ update the java installation directory against that line. In the beginning of this chapter, we have installed *default-jdk* so update the following line:

```
JAVA_HOME = \$(JAVA_HOME)
```

You may comment the above line and write your own statement such as $JAVA_HOME = /usr/lib/jvm/java-8-openjdk-amd64/bin/$ and something like that...

3.9.4 *core-site.xml*

Get the file with the help of `sudo gedit core-site.xml` and put the following code in between <configure> and </configure> tags.

```
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
```

3.9.5 *hdfs-site.xml*

Put the below code

```
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/namenode</value>
</property>
<property>
<name>dfs.datanode.data.dir</name>
<value>file:/usr/local/hadoop_tmp/hdfs/datanode</value>
</property>
```

3.9.6 *yarn-site.xml*

Put the following code

```
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
```

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

3.9.7 mapred-site.xml

Execute the below code to keep the old configuration file as it is.

```
cp /usr/local/hadoop/etc/hadoop/mapred-site.xml.template
↪  /usr/local/hadoop/etc/hadoop/mapred-site.xml
sudo gedit mapred-site.xml
```

And put the follwing code in it.

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

3.10 Start the Hadoop

Now it is time to start the Hadoop first by formating *NameNode*

```
hdfs namenode -format
```

You must be able to get the message "namenode has been successfully formatted" from the bottom fifth line (or what-so-ever it is). You must be able to find the output that NameNode is successfully formated. Now you may start the demons as below:

```
start-dfs.sh
start-yarn.sh
```

You must start these demons one by one to keep hadoop running. Now execute `hadoop version` at the console and you must be able to get relevant information.

3.11 Compiling Native Binaries

Once after installing Hadoop try to check native binaries, such as with `hadoop checknative` at console. The result should be as below:

If everything is correct, which means if your precompiled libraries are compatible with your architecture. Otherwise you may get the following message:

```
WARN util.NativeCodeLoader: Unable to load native-hadoop
↪  library for
your platform...using builtin-java classes where
↪  applicable
```

If the response to your command `hadoop checknative` is as above then something is wrong in your machine. The reason could be that the hadoop libraries (pre-compiled) are not correct for your machine. The you got to compile native libraries.

If native libraries are not properly installed user might exeperience problems at every phase of using hadoop such as formating namenode, starting deamons and using DFS resources. One of the potential reasons for encountering such problems is due to installations of precompiled libraries from the archives. Hadoop might be built by using certain resources which might be different from the user resources. So, that might leads to problems while executing commands. To avoid such problems the user may have to compile libraries with in his/her system.

Please read instructions at https://hadoop.apache.org/docs/r2.4.1/hadoop-project-dist/hadoop-common/NativeLibraries.html before compiling hadoop in your machine. The following instructions shows step-by-step procedure to compile hadoop native libraries in Ubuntu.

There are certain prerequisites that need to be installed in your system before triggering compiling. They are as follows:

1. Install *maven*
    - maven (you may download at https://maven.apache.org/download.cgi)

- Go to downloaded directoy and extract with the help of `tar -xf` for more information you may try `tar --help`.
- Go to */.bashrc* file and create environment variable then update the path
- Go to the terminal and try `mvn -version`

2. Install protocol buffer (also known as *protobuf*)
   - Download *protobuf* from https://github.com/google/protobuf or http://protobuf.googlecode.com/files/protobuf-2.5.0.tar.gz
   - Go to the download directory perform the below commands in sequence one after the other.
     - `sudo ./configure`
     - `sudo make`
     - `sudo make check`
     - `sudo make install`
   - Go to the terminal and try `protoc --version`
3. Install *gnu toolchain* such as
   - autoconf, automake, libtool, cmake, gcc, gcc-c++
   - zlib-devel
   - openssl-devel
   - snappy-devel
4. Download hadoop source from *hadoop apche releases* at http://hadoop.apache.org/releases.html
5. Change to downloaded directory
6. Compile with the help of `mvn package -Pdist,native -DskipTests -Dtar` command.

The go to *hadoop-dist/target/hadoop-2.4.1/lib/native* directory and copy *libhadoop.so* to your directory (*lib* directory under hadoop home)

Notes

[15]A set of formal descriptions that provide a standard for the design of operating systems, especially ones which are compatible with Unix. For more information read https://en.wikipedia.org/wiki/POSIX

[16]Use `Ctl+Alt+T` to open Terminal or Console in Ubuntu

[17]Kerberos is a computer network authentication protocol that works on the basis of 'tickets' to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.

[18]Visit http://man.openbsd.org/OpenBSD-current/man1/ssh-keygen.1 for more information.

[19]These URIs tend to change. Reader discretion is suggested while downloading appropriate version of Hadoop.

3.12 Questions

3.12.1 Short answer questions

1. What is Hadoop?
2. What is Hadoop cluster?
3. List various Hadoop daemons.
4. How do you start and stop Hadoop cluster?

3.12.2 Essay questions

1. Write a detail note on Hadoop history.
2. Write a detail note on Hadoop software library.
3. Write a detail note on Hadoop architecture.
4. Explain various steps involved in Hadoop installation in Ubuntu linux using code.
5. Explain various configuration files required for Hadoop.
6. Explain core commands of Hadoop using code.

Chapter 4

YARN

Hadoop is one of the most popular open-source distributed computation frameworks, popularized by the widely used MapReduce computation paradigm. However, recently, Hadoop as been trying to distance itself from a single computation model and provide an abstraction layer over a cluster of machines with which you can easily develop a great variety of fault-tolerant and scalable distributed computation paradigms. In a sense, Hadoop is now trying to become a "kernel" of the distributed cluster, handling the low level details so you don't have to, a kind of Linux for clusters.

Due to this paradigm change in Hadoop, we can identify 2 different Hadoop generations: pre-YARN and post-YARN (2.x). YARN (Yet Another Resource Manager) constitutes a significant improvement over the previous method of handling resources and applications inside Hadoop. To get a better idea of what exactly changed from the previous generation, you can have a look at these presentations by Spotify and Hortonworks. In a nutshell, the main features are:

1. Better scaling due to reduced centralization of responsabilities. Should easily support over 10K nodes, 10K jobs and 100K tasks (previous generation started getting into trouble at 4K nodes and 40K tasks).

2. More flexibility in resource allocations. You can now easily specify the requirements of each individual container directly in your YARN application, specifying needed RAM, CPU cores and even specific hosts on which the container should be allocated. Previously this was limited to global memory and CPU limits for all containers specified in configuration files.

3. More flexibility in computation. Previous Hadoop generations only ran MapReduce jobs. With YARN, Hadoop can run virtually any kind of computation. MapReduce is still possible but is implemented as a backwards-compatible module called MapReducev2 (not enabled by default). Other modules exist such as Giraph for graph processing, Spark for general computations with intermediate results stored in memory, etc.

4. Better failure handling. Previous generations had a single point of failure in the JobTracker losing the entire job queue in such an event. New generation has (or will soon have) recovery of both ApplicationMaster (through a restart by the ResourceManager) and the ResourceManager (through ZooKeeper, YARN-128, YARN-149, YARN-556).

5. Wire-compatible protocol. This should guarantee protocol compatibility even between different versions of Hadoop so you no longer have to worry about having to simultaneously update the entire cluster and can do rolling upgrades.

YARN is the component responsible for allocating containers to run tasks, coordinating the execution of said tasks, restart them in case of failure, among other housekeeping. Just like HDFS, it also has 2 main components: a *ResourceManager* which keeps track of the cluster resources and *NodeManagers* in each of the nodes which communicates with the ResourceManager and sets up containers for execution of tasks. We got to configure *yarn-site.xml* to use YARN. Please see the instructions in the installation section. You may go through the default configuration at `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-common/yarn-default.xml`

The general concept is that an application submission client submits an application to the YARN ResourceManager (RM). This can be

done through setting up a YarnClient object. After YarnClient is started, the client can then set up application context, prepare the very first container of the application that contains the ApplicationMaster (AM), and then submit the application. You need to provide information such as the details about the local files/jars that need to be available for your application to run, the actual command that needs to be executed (with the necessary command line arguments), any OS environment settings (optional), etc. Effectively, you need to describe the Unix process(es) that needs to be launched for your ApplicationMaster. The YARN ResourceManager will then launch the ApplicationMaster (as specified) on an allocated container. The ApplicationMaster communicates with YARN cluster, and handles application execution. It performs operations in an asynchronous fashion. During application launch time, the main tasks of the ApplicationMaster are:

- communicating with the ResourceManager to negotiate and allocate resources for future containers,
- after container allocation, communicating YARN *NodeManager*s (NMs) to launch application containers on them.

The three main protocols for YARN application (ApplicationClientProtocol, ApplicationMasterProtocol and ContainerManagementProtocol) are still preserved. The 3 clients wrap these 3 protocols to provide simpler programming model for YARN applications.

The entire process at clients side can go as shown below. [20]

1. The first step that a client needs to do is to initialize and start a *YarnClient*.
2. Once a client is set up, the client needs to create an application, and get its application id.
3. The response from the *YarnClientApplication* for a new application also contains information about the cluster such as the minimum/maximum resource capabilities of the cluster. This is required so that to ensure that you can correctly set the specifications of the container in which the *ApplicationMaster* would be launched.

4. The main crux of a client is to setup the *ApplicationSubmissionContext* which defines all the information needed by the RM to launch the AM. A client needs to set the following into the context:

    - *Application info*: id, name
    - *Queue, priority info*: Queue to which the application will be submitted, the priority to be assigned for the application.
    - *User*: The user submitting the application
    - *ContainerLaunchContext*: The information defining the container in which the AM will be launched and run. The ContainerLaunchContext, as mentioned previously, defines all the required information needed to run the application such as the local *Resources (binaries, jars, files etc.), Environment settings (CLASSPATH etc.), the Command to be executed and security T*okens (RECT).
    - At this point, the RM will have accepted the application and in the background, will go through the process of allocating a container with the required specifications and then eventually setting up and launching the AM on the allocated container.
    - *There are multiple ways a client can track progress of the actual task.* The ApplicationReport received from the RM consists of the following:
    - *General application information*: Application id, queue to which the application was submitted, user who submitted the application and the start time for the application.
    - *ApplicationMaster details*: the host on which the AM is running, the rpc port (if any) on which it is listening for requests from clients and a token that the client needs to communicate with the AM.
    - *Application tracking information*: If the application supports some form of progress tracking, it can set a tracking

url which is available via ApplicationReport's getTrackingUrl() method that a client can look at to monitor progress.

5. In certain situations, if the application is taking too long or due to other factors, the client may wish to kill the application. YarnClient supports the killApplication call that allows a client to send a kill signal to the AM via the ResourceManager. An ApplicationMaster if so designed may also support an abort call via its rpc layer that a client may be able to leverage.

The entire process at master (AM) side can go as shown below.

1. The AM is the actual owner of the job. It will be launched by the RM and via the client will be provided all the necessary information and resources about the job that it has been tasked with to oversee and complete.

2. As the AM is launched within a container that may (likely will) be sharing a physical host with other containers, given the multi-tenancy nature, amongst other issues, it cannot make any assumptions of things like pre-configured ports that it can listen on.

3. When the AM starts up, several parameters are made available to it via the environment. These include the *ContainerId* for the AM container, the application submission time and details about the NM (NodeManager) host running the *ApplicationMaster*. Ref *ApplicationConstants* for parameter names.

4. All interactions with the RM require an *ApplicationAttemptId* (there can be multiple attempts per application in case of failures). The *ApplicationAttemptId* can be obtained from the AM's container id. There are helper APIs to convert the value obtained from the environment into objects.

5. After an AM has initialized itself completely, we can start the two clients: one to ResourceManager, and one to NodeManagers. We set them up with our customized event handler, and we will talk about those event handlers in detail later in this article.

6. The AM has to emit heartbeats to the RM to keep it informed

that the AM is alive and still running. The timeout expiry interval at the RM is defined by a config setting accessible via *YarnConfiguration.RM_AM_EXPIRY_INTERVAL_MS* with the default being defined by *YarnConfiguration.DEFAULT_RM_AM_EXPIRY_I* The ApplicationMaster needs to register itself with the ResourceManager to start hearbeating.

7. In the response of the registration, maximum resource capability if included. You may want to use this to check the application's request.

8. Based on the task requirements, the AM can ask for a set of containers to run its tasks on. We can now calculate how many containers we need, and request those many containers.

9. In *setupContainerAskForRM()*, the follow two things need some set up:

   - *Resource capability*: Currently, YARN supports memory based resource requirements so the request should define how much memory is needed. The value is defined in MB and has to less than the max capability of the cluster and an exact multiple of the min capability. Memory resources correspond to physical memory limits imposed on the task containers. It will also support computation based resource (vCore), as shown in the code.

   - *Priority*: When asking for sets of containers, an AM may define different priorities to each set. For example, the Map-Reduce AM may assign a higher priority to containers needed for the Map tasks and a lower priority for the Reduce tasks' containers.

10. After container allocation requests have been sent by the application manager, contailers will be launched asynchronously, by the event handler of the *AMRMClientAsync* client. The handler should implement *AMRMClientAsync. CallbackHandler* interface.

11. When there are containers allocated, the handler sets up a thread that runs the code to launch containers. Here we use the name

LaunchContainerRunnable to demonstrate. We will talk about the LaunchContainerRunnable class in the following part of this article.

12. On heart beat, the event handler reports the progress of the application.

13. The container launch thread actually launches the containers on NMs. After a container has been allocated to the AM, it needs to follow a similar process that the client followed in setting up the *ContainerLaunchContext* for the eventual task that is going to be running on the allocated Container. Once the *ContainerLaunchContext* is defined, the AM can start it through the *NMClientAsync*.

14. The *NMClientAsync* object, together with its event handler, handles container events. Including container start, stop, status update, and occurs an error.

15. After the ApplicationMaster determines the work is done, it needs to unregister itself through the AM-RM client, and then stops the client.

Testing

The most important ingridient in Yarn is that of memory and cores. For instance, if your computer has 2GB RAM and 2 cores (`free -g or free -m` to know about RAM in your computer). Use `cat /proc/meminfo` to know entire memory status of your computer. Use `cat /proc/cpuinfo | grep processor | wc -l` to know about number of processors (i.e. cores). The Yarn is all about managing resources. So we might be able to restrict or permit tasks by earmarking or allocating memory and cores. For instance, if you look at the default configuration settings of Yarn , you may be able to find the below values for memory and cores.

Memory requests lower or greater than the specified limits will throw a *InvalidResourceRequestException*. Requests lower or greater than the specified value for cores also will throw a *InvalidResourceRequestException*. So, that is how Yarn try to control the resources by providing

Varibale	Value	Description
yarn.scheduler.minimum-allocation-mb	1024	The minimum allocation for every container request at the RM, in MBs.
yarn.scheduler.maximum-allocation-mb	8192	The maximum allocation for every container request at the RM, in MBs.
yarn.scheduler.minimum-allocation-vcores	1	The minimum allocation for every container request at the RM, in terms of virtual CPU cores.
yarn.scheduler.maximum-allocation-vcores	4	The maximum allocation for every container request at the RM, in terms of virtual CPU cores.

containers to tasks such that they will be executed error-free. So the users need to adapt these resources as per the case.

Now let us see as how to test the cluster with custom values (specified by user). We need the `hadoop-yarn-applications-distributedshell -*.*.*.jar` to execute this test.

The following is the entire statement that need to be executed at the console:

```
hadoop jar
↪ hadoop-yarn-applications-distributedshell-2.7.2.jar
↪ org.apache.hadoop.yarn.applications.distributedshell.
↪ Client --jar
↪ hadoop-yarn-applications-distributedshell-2.7.2.jar
↪ --shell_command date --num_containers 2
↪ --master_memory 1024
```

With this command we are telling hadoop to run the Client class in the `hadoop-yarn-applications-distributedshell-2.2.0.jar`, passing it the jar containing the definition of the *ApplicationMaster* (the same jar), the shell command to run in each of the hosts (date), the number of containers to spawn (2) and the memory used by the ApplicationMaster (1024MB). The value of 1024 was set empirically by trying to run the program several times until it stopped failing due to the ApplicationMaster using more memory than that which had been allocated to it. This throw little output and finally ends with a statement

```
16/09/07 16:55:50 INFO distributedshell.Client:
↪ Application has completed successfully. Breaking
↪ monitoring loop
16/09/07 16:55:50 INFO distributedshell.Client:
↪ Application completed successfully
```

You are successful. Then find a statement like
*application*_1473245156605_0001 and run
`grep "" $HADOOP_PREFIX/logs/userlogs/ <APPLICATION ID>/**/stdout`
in the console. In my case it is...

```
grep "" \$HADOOP_PREFIX/logs/userlogs/<APPLICATION ID>
↪  /**/stdout
```

Notes

[20]Read at https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/WritingYarnApplications.html for full set of instructions.

4.1 Questions

4.1.1 Short answer questions

1.

4.1.2 Essay questions

1.

Chapter 5

HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets. HDFS relaxes a few POSIX requirements to enable streaming access to file system data. HDFS was originally built as infrastructure for the Apache Nutch web search engine project. HDFS is part of the Apache Hadoop Core project. The project URL is http://hadoop.apache.org/. Read more about HDFS at http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

HDFS is the primary distributed storage used by Hadoop applications. A HDFS cluster primarily consists of a NameNode that manages the file system metadata and DataNodes that store the actual data. The HDFS architecture deal with basic interactions among NameNode, the DataNodes, and the clients. Clients contact NameNode for file metadata or file modifications and perform actual file I/O directly with the DataNodes.

The following are some of the salient features of HDFS:

1. Hadoop, including HDFS, is well suited for distributed storage and distributed processing using commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. MapReduce, well known for its simplicity and applicability for large set of distributed applications, is an integral part of Hadoop.

2. HDFS is highly configurable with a default configuration well suited for many installations. Most of the time, configuration needs to be tuned only for very large clusters.

3. Hadoop is written in Java and is supported on all major platforms.

4. Hadoop supports shell-like commands to interact with HDFS directly.

5. The NameNode and Datanodes have built in web servers that makes it easy to check current status of the cluster.

New features and improvements are regularly implemented in HDFS. The following is a subset of useful features in HDFS:

- File permissions and authentication.
- *Rack awareness*: to take a node's physical location into account while scheduling tasks and allocating storage.
- *Safemode*: an administrative mode for maintenance.
- *fsck*: a utility to diagnose health of the file system, to find missing files or blocks.
- *fetchdt*: a utility to fetch DelegationToken and store it in a file on the local system.
- *Balancer*: tool to balance the cluster when the data is unevenly distributed among DataNodes.
- *Upgrade and rollback*: after a software upgrade, it is possible to rollback to HDFS' state before the upgrade in case of unexpected problems.

- *Secondary NameNode*: performs periodic checkpoints of the namespace and helps keep the size of file containing log of HDFS modifications within certain limits at the NameNode.
- *Checkpoint node*: performs periodic checkpoints of the namespace and helps minimize the size of the log stored at the NameNode containing changes to the HDFS. Replaces the role previously filled by the Secondary NameNode, though is not yet battle hardened. The NameNode allows multiple Checkpoint nodes simultaneously, as long as there are no Backup nodes registered with the system.
- *Backup node*: An extension to the Checkpoint node. In addition to checkpointing it also receives a stream of edits from the NameNode and maintains its own in-memory copy of the namespace, which is always in sync with the active NameNode namespace state. Only one Backup node may be registered with the NameNode at once.

5.1 Webinterface

NameNode and DataNode each run an internal web server in order to display basic information about the current status of the cluster. With the default configuration, the *NameNode* front page is at `http://namenode-name:50070/`. It lists the *DataNodes* in the cluster and basic statistics of the cluster. The web interface can also be used to browse the file system (using "Browse the file system" link on the NameNode front page).

5.2 Shell Commands

Hadoop includes various shell-like commands that directly interact with HDFS and other file systems that Hadoop supports. The command bin/hdfs dfs -help lists the commands supported by Hadoop shell. Furthermore, the command `bin/hdfs dfs -help command-name` displays more detailed help for a command. These commands support most of the normal files system operations like copying files, changing

file permissions, etc. It also supports a few HDFS specific operations like changing replication of files. For more information see File System Shell Guide at http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html. To know it in your computer just do as `hadoop fs -help`. You may get a list of functions that are useful to perform operations on HDFS. For instance `-appendToFile` (the very first function or called as *option* is useful to update the content of a local file (.txt or etc.) to destination folder (possibly located at *dfs*). The description to this *option* is given as below.

```
-appendToFile <localsrc> ... <dst> :
  Appends the contents of all the given local files to
  ↪  the given dst file. The dst
  file will be created if it does not exist. If
  ↪  <localSrc> is -, then the input is
  read from stdin.
```

We may practice the code as shown below

```
sudo gedit text.txt # copy or write some data into it
↪  just like "this only a test file"
hadoop fs -mkdir /dir1
hadoop fs -copyFromLocal text.txt /dir1/
hadoop gedit text.txt # this time add few lines of text
↪  like "to test few commands of hadoop"
hadoop fs -appendTofile text.txt /dir1/text.txt # now you
↪  must be able to see both lines together in the file.
```

In the above code `-mkdir, -copyFromLocal` are other two options or functions you may read as well from the help context i.e. `hadoop fs -help`. Following is the list of few other commands.

1. *cat*: Copies source paths to stdout.
2. *checksum*: Returns the checksum information of a file.
3. *chgrp*: Change group association of files. The user must be the owner of files, or else a super-user.
4. *chmod*: Change the permissions of files. With -R, make the change recursively through the directory structure. The user

must be the owner of the file, or else a super-user.

5. *chown*: Change the owner of files. The user must be a super-user.

6. *copyFromLocal*: Similar to put command, except that the source is restricted to a local file reference.

7. *copyToLocal*: Similar to get command, except that the destination is restricted to a local file reference.

8. *count*: Count the number of directories, files and bytes under the paths that match the specified file pattern. The output columns with -count are: $DIR\_COUNT, FILE\_COUNT, CONTENT\_SIZE, PATHNAME$.

9. *cp*: Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory.

10. *createSnapshot & deleteSanpshot*: HDFS Snapshots are read-only point-in-time copies of the file system. Snapshots can be taken on a subtree of the file system or the entire file system. Some common use cases of snapshots are data backup, protection against user errors and disaster recovery. Please read more on http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsSnapshots.html

11. *df*: display free space This command is highly useful. HDFS resources are precious for execution of jobs. The memory gradually reduces each time the job is exexuted. This command helps knowing or verifying memory status. For instance, `hadoop fs -df /` will let you know the memory status. [21]

12. *du*: Displays sizes of files and directories contained in the given directory or the length of a file in case its just a file.

13. *dus*: Displays a summary of file lengths.

14. *expunge*: Empty the Trash. Perhaps the other important command after *du, df* that helps us to make MapReduce jobs rather better.

15. *find*: Finds all files that match the specified expression and applies selected actions to them. If no path is specified then defaults to the current working directory. If no expression is specified then defaults to -print.

    ```
    hduser@hduser:~\$ hadoop fs -find / -name text.txt
    /dir1/text.txt
    ```

16. *get*: Copy files to the local file system. Files that fail the CRC check may be copied with the -ignorecrc option. Files and CRCs may be copied using the -crc option.

    ```
    hadoop fs -get /dir1/text.txt
    ```

    Once after executing the code we can check with native Linux list command *ls* to check if the file is downloaded from the hdfs.

17. *getfacl*: Displays the Access Control Lists (ACLs) of files and directories. If a directory has a default ACL, then getfacl also displays the default ACL.

    ```
    hadoop fs -getfacl /dir1/
    ```

    Will display the following information

    ```
    # file: /dir1
    # owner: hduser
    # group: supergroup
    getfacl: The ACL operation has been rejected.
    ↪  Support for ACLs has been disabled by setting
    ↪  dfs.namenode.acls.enabled to false.
    ```

18. *getfattr*: Displays the extended attribute names and values (if any) for a file or directory.

19. *getmerge*: Takes a source directory and a destination file as input and concatenates files in src into the destination local file. Optionally -nl can be set to enable adding a newline character (LF) at the end of each file.

20. *help*: Perhaps this is foremost and important command that every user must know. This command helps understanding options

associated with any given command. Precisely, this command returns usage output.

21. *ls*: This command is mostly used command in HDFS. This command has three options associated with it, namely *-d, -h, -R*. The option *-R* return entire information of HDFS. Alternatively we can also use *-lsr* in stead of with -R option.

22. *moveFromLocal*: Similar to put command, except that the source localsrc is deleted after it's copied. The difference between *put* and *moveFromLocal* is that *moveFromLocal* deletes the source file once after copying.

23. *moveToLocal*: This command performs just as above but copies files from HDFS to local FS.

24. *mv*: Moves files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across file systems is not permitted. For instance, let us see how *mv* executes the tast with in HDFS.

```
hduser@hduser:~$ hadoop fs -mv /dir1/text.txt /dir2/
hduser@hduser:~$ hadoop fs -ls -R /
drwxr-xr-x   - hduser supergroup          0
↪  2016-09-07 10:43 /dir1
drwxr-xr-x   - hduser supergroup          0
↪  2016-09-07 10:43 /dir2
-rw-r--r--   1 hduser supergroup        272
↪  2016-09-07 10:02 /dir2/text.txt

hduser@hduser:~$ hadoop fs -ls /
Found 2 items
drwxr-xr-x   - hduser supergroup          0
↪  2016-09-07 10:43 /dir1
drwxr-xr-x   - hduser supergroup          0
↪  2016-09-07 10:43 /dir2
hduser@hduser:~$ hadoop fs -ls /dir1
```

25. *put*: Copy single src, or multiple srcs from local file system to the destination file system. Also reads input from stdin and writes

to destination file system.

26. *renameSnapshot*: Rename a snapshot. This operation requires owner privilege of the snapshottable directory.
27. *rm*: Delete files specified as args. The args: *-f, -r, -R, -skipTrash*
28. *rmdir*: Delete a directory.
29. *rmr*: Recursive version of delete.
30. *setfacl*: Sets Access Control Lists (ACLs) of files and directories.
    - -b: Remove all but the base ACL entries. The entries for user, group and others are retained for compatibility with permission bits.
    - -k: Remove the default ACL.
    - -R: Apply operations to all files and directories recursively.
    - -m: Modify ACL. New entries are added to the ACL, and existing entries are retained.
    - -x: Remove specified ACL entries. Other ACL entries are retained.
    - –set: Fully replace the ACL, discarding all existing entries. The acl_spec must include entries for user, group, and others for compatibility with permission bits.
    - acl_spec: Comma separated list of ACL entries. path: File or directory to modify.
31. *setattr*: Sets an extended attribute name and value for a file or directory.
32. *setrep*: Changes the replication factor of a file. If path is a directory then the command recursively changes the replication factor of all files under the directory tree rooted at path.
33. *stat*: Print statistics about the file/directory at <path> in the specified format. Format accepts filesize in blocks (%b), type (%F), group name of owner (%g), name (%n), block size (%o), replication (%r), user name of owner(%u), and modification date

(%y, %Y). %y shows UTC date as $yyyy-MM-ddHH:mm:ss$ and %Y shows milliseconds since $January 1, 1970 UTC$. If the format is not specified, %y is used by default. Eg. `hadoop fs -stat "%F %u:%g %b %y %n" /file`

34. *tail*: Displays last kilobyte of the file to stdout. Eg. `hadoop fs -tail pathname`.

35. *test*: Tests if entity or object in certain path. Emits boolean value as result.

    - -d: f the path is a directory, return 0.
    - -e: if the path exists, return 0.
    - -f: if the path is a file, return 0.
    - -s: if the path is not empty, return 0.
    - -z: if the file is zero length, return 0.

36. *text*: Takes a source file and outputs the file in text format. The allowed formats are *zip* and *TextRecordInputStream*.

37. *touchz*: Create a file of zero length. Eg. `hadoop fs -touchz pathname`

38. *truncate*:Truncate all files that match the specified file pattern to the specified length. The `-w` flag requests that the command waits for block recovery to complete, if necessary. Without `-w` flag the file may remain unclosed for some time while the recovery is in progress. During this time file cannot be reopened for append. The syntax is `hadoop fs -truncate [-w] <length> <paths>`. Eg. `hadoop fs -truncate 55 /user/hadoop/file1 /user/hadoop/file2` or `hadoop fs -truncate -w 127 hdfs://nn1.example.co`

39. *usage*: Return the help for an individual command. Eg. `hadoop fs -usage command`

Notes

[21]There are certain problem that are persistent and also naging while executing jobs. One of the problems is *MapReduce* jobs. At times it is common that MapReduce code exists with out executing besides throwing certain complaints. One of the reasons for these problems is shortage of memory. Commands like *df, du* are greatly helpful while verifying the memory for execution.

5.3 Questions

5.3.1 Short answer questions

1.

5.3.2 Essay questions

1.

Chapter 6

Core JAVA

6.1 What is Java?

Java is a programming language and a platform. [22] Java history is interesting to know. The history of java starts from Green Team. Java team members (also known as Green Team), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc. For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape. Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java. James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team. Originally designed for small, embedded systems in electronic appliances like set-top boxes. Firstly, it was called "Greentalk" by James Gosling and file extension was *.gt*. After that, it was called Oak and was developed as a part of the Green project. Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies. In 1995, Time magazine called Java one of the Ten

Best Products of 1995. The official *JDK 1.0* released in(January 23, 1996). On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL). On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suite various types of platforms. Ex: J2EE for Enterprise Applications, J2ME for Mobile Applications. The new J2 versions were renamed as Java SE, Java EE and Java ME respectively. Java is guaranteed to be Write Once, Run Anywhere.

6.1.1 Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

6.1.2 Requirements for write Java programs

For performing the examples discussed in this tutorial, you will need a Pentium $200 - MHz$ computer with a minimum of 64 MB of RAM (128 MB of RAM recommended).

You also will need the following softwares:

1. Linux 7.1 or Windows xp/7/8 operating system.
2. Java JDK 8
3. Microsoft Notepad or any other text editor

6.2 Installation

This manual uses Ubuntu 16.04, gedit text writer for examples. So in Ubuntu or any other linux machines can accommodate Java very easily. For instance, in Ubuntu go to *Sofware Center* choose OpenJDK and just press install. Otherwise, if you are interested in using *Terminal* just use the following statement in the *Console/Terminal* (Note: in Ubuntu you nay call the Terminal or Console by using shortcut keys *Ctl+Alt+T*).

```
sudo apt-get install default-jdk, default-jre
```

The above statement takes a couple of minutes depending upon the your internet speed. The advantage of Linux (Ubuntu) is that you don't need to worry about setting up path and etc. Just go to the Terminal and verify by executing `java -version`. The Console must be able to retun the following message.

```
openjdk version "1.8.0_91"
OpenJDK Runtime Environment (build
↪  1.8.0_91-8u91-b14-3ubuntu1~16.04.1-b14)
OpenJDK 64-Bit Server VM (build 25.91-b14, mixed mode)
```

`openjdk version "1.8.0_91"` shows that the Java is v8. There are lot of online resouces that has abundant of information as how to install and setup Java in other platforms like Windows and Mac. In case if you have multiple distributions (installations) of Java you may try

`update-alternatives --config java` to switch between versions or types. [23]

6.3 Editors

To write your Java programs, you will need a text editor. There are even more sophisticated IDEs available in the market. But for now, you can consider one of the following:

1. Text Editor like *gedit, vi, nano* in Linux, or Notepad in Windows.
2. Netbeans
3. Eclipse

We will be using *gedit* a free *gnome text editor* available by default in Ubuntu.

6.3.1 Common mistakes

The following are the some of the common mistakes we might perform while executing code.

1. Check the *slash* symbol before comment line, it must be / but not \.
2. If you are using arrays check the index both at declaration and in output script.
3. At output statement *"System.out.println()"*, there must not be comma separating between text statement and variable. For instance, `System.out.println("The out is: ", + a)` is wrong; `System.out.println("The out is: " + a)` is correct.

6.4 Programming

6.4.1 Java is object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

6.4.2 Objects and Classess

Object

Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class. Let us now look deep into what are objects. If we consider the real-world we can find many objects around us, Cars, Dogs, Humans, etc. All these objects have a state and behavior. If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging, running. If you compare the software object with a real world object, they have very similar characteristics. Software objects also have a state and behavior. A software object's state is stored in fields and behavior is shown via methods. So in software development, methods operate on the internal state of an object and the object-to-object communication is done via methods.

Calss

A class is a blue print from which individual objects are created.

```
public class Dog{

   String breed;  // state
   int ageC // state
   String color; // state
```

```
   void barking(){ // behaviour
   }

   void hungry(){
   }

   void sleeping(){
   }
}
```

A class can contain any of the following variable types.

1. *Local variable*: Variables defined inside methods, constructors or blocks are called local variables.
2. *Instance variable*: Instance variables are variables within a class but outside any method.
3. *Class variable*: Class variables are variables declared with in a class, outside any method, with the static keyword.

Constructor

When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

Example of a constructor is given below:

```
public class Puppy{
   public Puppy(){
   }

   public Puppy(String name){
      // This constructor has one parameter, name.
```

```
   }
}
```

The below code shows *instance variables* and *methods* in a class.

```
public class Puppy{

   int puppyAge;

   public Puppy(String name){
      // This constructor has one parameter, name.
      System.out.println("Name chosen is :" + name );
   }

   public void setAge( int age ){
       puppyAge = age;
   }

   public int getAge( ){
       System.out.println("Puppy's age is :" + puppyAge );
       return puppyAge;
   }

   public static void main(String []args){
      /* Object creation */
      Puppy myPuppy = new Puppy( "tommy" );

      /* Call class method to set puppy's age */
      myPuppy.setAge( 2 );

      /* Call another class method to get puppy's age */
      myPuppy.getAge( );

      /* You can access instance variable as follows as well */
      System.out.println("Variable Value :" + myPuppy.puppyAge );
   }
}
```

6.4.3 Your First Java Program

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

1. Install Java (We did this in the previous chapter)
2. Set the path (We did this in the previous chapter)
3. Write the program
4. Compile the program

Now let us write a program in simple text editor in Ubuntu. I will crate a text document from the terminal and start writing the program.

```
sudo mkdir tmp
sudo gedit mfjp.txt
```

The second statement open a simple text application. Write the following code in the file.

```
class MyClass
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

Close the application (text file), and execute the following statements.

```
sudo cp mfjp.txt mfjp.java
javac mfjp.java
```

The second statement create two files in your workding directory (`pwd`) you may find a class file (MyClass) as *MyClass.class*. Then you might try executing the program by this way.

```
java MyClass
```

You may get the output as "Hello Java" in terminal. Now let us learn few basics of Java.

6.4.4 Dos

About Java programs, it is very important to keep in mind the following points.

1. *Case Sensitivity* - Java is case sensitive, which means identifier Hello and hello would have different meaning in Java.

2. *Class Names* - For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example, *MyJavaExample*.

3. *Method Names* - All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example *public void myMethodName()*.

4. *Program File Name* - Name of the program file should exactly match the class name. When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match your program will not compile). Example: Assume *MyFirstJavaProgram* is the class name. Then the file should be saved as *MyFirstJavaProgram.java*. `public static void main(String args[])` - Java program processing starts from the `main()` method which is a mandatory part of every Java program.

6.4.5 Variables and Data Types

In computer science *variable* is a name of memory location.There are three types of variables: *local, instance* and *static*. There are two types of *datatypes* in java, *primitive* and *non-primitive*. The following table shows the details of different data types.

The another important subject that needs attention is *operators*. The following table shows the operators used in Java programming.

Table 6.1: Data Types in Java

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

6.5 Exercises

The following are few exercises.

6.5.1 Ex-1: System Input-Output

The exercise is going to be like this that the console ask for a number and then print later the same number.

```
import java.util.Scanner;

class inOut{

        public static void main(String[] args){

        Scanner scan = new Scanner(System.in);
        int num = 0;
        System.out.println("Enter any number: ");
        num = scan.nextInt();

        System.out.println("Input number is: " + num);
        }
}
```

Table 6.2: Table of operators

Operators	Precedence
postfix	expr++ expr–
unary	++expr –expr +expr -expr ∼ !
multiplicative	* / %
additive	+ -
shift	«»»>
relational	<><= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= + = − = ∗ = / = % = & =^=;= <<=>>=>>>=

6.5.2 Ex-2: If statement

```
public class ifExample
{
        public static void main(String[] args)
        {
                int age = 40;
                if(age <= 50)
                {
                        System.out.println("The age is
                        ↪  less than 50");
                }
        }
}
```

The above code evaluates the *age* of the individuals and returns the result. The following is the code for nested-if.

6.5.3 Ex-3: Nested-If

```
import java.util.Scanner;

class nestIf {
        public static void main(String[] args){

        Scanner scan = new Scanner(System.in);

        int s = 0;

        System.out.println("Enter some number: ");

        s = scan.nextInt();

        if(s % 2 == 0){

                System.out.println("The number is even");

                } else{

                System.out.println("The numbe is not
                ↪  even");

                }

        }

}
```

The expected output is as follows:

```
Enter some number:
100
The number is even

Enter some number:
125
The numbe is not even
```

6.5.4 Ex-4: If-Else-If example

Let us write code that makes student grade for different denominations. For instance, look at the following code.

```
import java.util.Scanner;

class studMarks{

        public static void main(String[] args){

        Scanner scan = new Scanner(System.in);

        int m = 0;

        System.out.println("Input marks: ");
        m = scan.nextInt();

        if(m < 50){
        System.out.println("Fail");
        } else if(m >= 50 && m <= 60){
        System.out.println("C Grade");
        }else if(m >= 60 && m <= 75){
        System.out.println("B Grade");
        }else if(m >= 75 && m <= 100){
        System.out.println("A Grade");
        }else{
        System.out.println("Invalid");
        }
}
}
```

The output for the above code is as follows:

```
Input marks:
58
C Grade

Input marks:
59
```

```
C Grade

Input marks:
75
B Grade

Input marks:
90
A Grade
```

6.5.5 Ex-5: Switch Statement

The Java switch statement is executes one statement from multiple conditions. It is like if-else-if ladder statement. Sometimes, it is easy to use *switch* statement in stead of *if-else-if* statemets. See the below exercise.

```
import java.util.Scanner;

class switchEx{
        public static void main(String[] args){
                Scanner scan = new Scanner(System.in);

                int s = 0;

                System.out.println("Enter your choice:
                ↪  ");
                s = scan.nextInt();

                switch(s){
                        case 1: System.out.println("you
                        ↪  want to try AM"); break;
                        case 2: System.out.println("you
                        ↪  want to try GM"); break;
                        case 3: System.out.println("you
                        ↪  want to try HM"); break;
```

```
                        default: System.out.println("your
                        ↪  input is invalid"); break;
                        }
                }

        }
```

The above code reads the user input and outputs the action intended by the user. Actually the code need to compute the measure of interst and output instead of a just statement. For instance, if your input is 1 the code must be able to retun *arithmetic mean*, but that needs a data distribution (vector). So, we need additional input i.e. array to computer statistical measures. We will try to continue this example in next section on arrays.

6.5.6 Ex-3: for loop

For statements are highly useful to execute loops.

```
class forEx{

        public static void main(String args[]){

        for(int i = 1; i < 10; i++){
        System.out.print(i + "\n");
                }
        }
}
```

The above code prints a linear series of numbers. You may have to observe $"\backslash n"$ in print statement this helps printing numbers vertically in the console, otherwise the code prints numbers one after the other adjacently (horizontally).

6.5.7 Ex-4: While Loop

The above exercise also can be done in the following way:

```
class doEx{
```

```
        public static void main(String[] args){

        int i = 1;

        while(i <= 10){
                System.out.println(i + "\n");
                i++;

                }

        }

}
```

The other way of performing the same action:

```
class doWhileEx{

        public static void main(String[] args){

        int i = 1;

        do{
        System.out.println(i);
        i++;
        }while(i <= 10);

        }
}
```

6.5.8 Ex-5: Random Number Generation (RNG)

Now let us do a small interesting exercise. In data anlaytics, as it was mentioned before, often data analysis try to use code on simulated data such as through Random Number Generation (RNG). There is some certain material on math functions of Java at https://docs.oracle.com/javase/tutorial/java/data/beyondmath.html.

Let us do this step-by-step approach. This example is very important

for data analytics has a lot to do with random values of certain distributions. Data gets generated by certain processes. Mostly, when data analysts gets data, it is all about information regardign some certain process. These processes might be random or non-random. Most of the time data analysts works with random data distributions to test their models (code). So, the data analyst need to know as how to exercise with random processes.

```
import java.util.Random;

class randEx{

        public static void main(String[] args){
                Random rand = new Random();
                double n = rand.nextInt();
                System.out.println(n);
        }

}
```

The above code creates only one data point which generated randomly with the help of *random* function. But we need a vector of values to claim that it is a data distribution (a univarite variable). So, we got to use *loop* to generate a set of values by looping the random number generation process.

```
import java.util.Random;

class randEx{

        public static void main(String[] args){
                Random rand = new Random();

        for(int i = 1; i <= 10; i++){
                double n = rand.nextInt(10)+1;
                System.out.println(n);
                        }
        }
```

```
}
```

The above code gives the following output.

```
8.0
1.0
9.0
8.0
5.0
8.0
8.0
8.0
5.0
7.0
```

If we change the value within the paranthesis i.e. `double n = rand.nextInt(10)+` to `double n = rand.nextInt()*10;`, the output will be different.

```
-9.64653906E8
-1.386765098E9
1.44151986E8
-6.869232E7
-2.17323714E8
-1.277274944E9
-1.300752846E9
-2.39470446E8
-1.75980253E9
1.120968E9
```

If we keep empty paranthesis without multiplying with any value.

```
1.11883961E9
-2.16984983E8
6.50234678E8
2.046700708E9
-1.676107933E9
1.885349562E9
2.097020595E9
2.027852388E9
1.04450492E8
-1.624830023E9
```

Perhaps the right method might be `double n = rand.nextInt(10)+1;`. You can design your process as you wish, but depending on the requirements. Try to change the value in paranthesis and try the same code and observe changes. Moreover, there is certain resource on all those function that are available for mathematical operations at https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html. We might discuss some of these methods in next chapter Java for Statistical Analysis and Packaging.

```
import java.util.Random;

class twoDarr{
        public static void main(String[] args){

        double[][] b = new double[5][5];

        Random rand = new Random();
        double n = rand.nextDouble()*10+1;

        for(int r = 0; r < b.length; r++){
                for(int c = 0; c < b[r].length; c++){
                b[r][c] = n;
                        }
                }

        for(int r = 0; r < b.length; r++){
                for(int c = 0; c < b[r].length; c++){
                        System.out.print(b[r][c] +
                        ↪  "\t");

                        }
                System.out.println();
                }

        }
}
```

See that you use `System.out.print()` in inner loop otherwise you don't get a matrix in stead you may get all the columns listed vertically

one below the other. This code still has a setback. The code produces only one unique number as element in all crosssections. I mean the elements are same. To have different random values we might write the code as below:

```
import java.util.Random;
import java.lang.Math;

class twoDarr{
        public static void main(String[] args){

        double[][] b = new double[5][5];

        Random rand = new Random();


        for(int r = 0; r < b.length; r++){
                for(int c = 0; c < b[r].length; c++){
                double n = Math.round(Math.random()*100);
                ↪
                b[r][c] = n;
                        }
                }

        for(int r = 0; r < b.length; r++){
                for(int c = 0; c < b[r].length; c++){
                        System.out.print("\t" +
                        ↪  b[r][c]);

                        }
                System.out.println();
                }

        }
}
```

The difference is that we used two additional methods i.e. *round* and *random* from *Math* library. Now the statement `double n = Math.round(Math.ra` produces a unique value for every iteration of the loop. The expected

output might be as below:

```
29.0 43.0 8.0 69.0 11.0
77.0 39.0 2.0 5.0 39.0
78.0 15.0 18.0 78.0 15.0
55.0 42.0 28.0 68.0 23.0
61.0 19.0 49.0 65.0 56.0
```

6.6 Maps

The Map interface maps unique keys to values. A key is an object that you use to retrieve a value at a later date. Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key. But you got to be a bit careful while using Maps. Several methods throw a `NoSuchElementException` when no items exist in the invoking map. A `ClassCastException` is thrown when an object is incompatible with the elements in a map. A `NullPointerException` is thrown if an attempt is made to use a null object and null is not allowed in the map. An `UnsupportedOperationException` is thrown when an attempt is made to change an unmodifiable map.

A Map is an interface that maps keys to values. The keys are unique and thus, no duplicate keys are allowed. A map can provide three views, which allow the contents of the map to be viewed as a set of keys, collection of values, or set of key-value mappings. In addition, the order of the map is defined as the order in which, the elements of a map are returned during iteration.

The Map interface is implemented by different Java classes, such as `HashMap, HashTable` and `TreeMap`. Each class provides different functionality and can be either synchronized or not. Also, some implementations prohibit null keys and values, and some have restrictions on the types of their keys.

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys. Map is useful if you have to search, update or delete elements on the basis of key.

6.6.1 Collections

The concept of maps belongs to a parent concept called *collections*. Data can be organized throught these collections in number of ways. Earlier i.e. before Java 2 there used to be adhoc classes like *Dictionary, Vector, Stack* and *Properties*. However, these classes were different in terms of their themes. For instance, the purpose of Vector is different from Property. The collections frameworks serves to fulfill following objectives.

1. The framework had to be high-performance. The implementations for the fundamental collections (dynamic *arrays*, *linked lists*, *trees*, and *hashtables*) are highly efficient.

2. The framework had to allow different types of collections to work in a similar manner and with a high degree of interoperability.

3. Extending and/or adapting a collection had to be easy.

To achieve these goals Java implemented collections framework. Collections framework operates through certain standard interfaces. Few of the standard interfaces were; *LinkedList, HashSet* amd *TreeSet*. The following tables shows the details of these interfaces.

The collection classes

Java provides a set of standard collection classes that implement Collection interfaces. Some of the classes provide full implementations that can be used as-is and others are abstract class, providing skeletal implementations that are used as starting points for creating concrete collections.

The standard collection classes are summarized in the following table:

1. *AbstractCollection* : Implements most of the Collection interface.

2. *AbstractList*: Extends AbstractCollection and implements most of the List interface.

3. *AbstractSequentialList* : Extends AbstractList for use by a collection that uses sequential rather than random access of its elements.

S.No.	Interface with description
The collection interface	This enables you to work with groups of objects; it is at the top of the collections hierarchy.
The List Interface	This extends Collection and an instance of List stores an ordered collection of elements.
The Set	This extends Collection to handle sets, which must contain unique elements
The SortedSet	This extends Set to handle sorted sets
The Map	This maps unique keys to values.
The Map.Entry	This describes an element (a key/value pair) in a map. This is an inner class of Map.
The SortedMap	This extends Map so that the keys are maintained in ascending order.
The Enumeration	This is legacy interface and defines the methods by which you can enumerate (obtain one at a time) the elements in a collection of objects. This legacy interface has been superseded by Iterator.

4. *LinkedList*: Implements a linked list by extending AbstractSequentialList.
5. *ArrayList* : Implements a dynamic array by extending AbstractList.
6. *AbstractSet* : Extends AbstractCollection and implements most of the Set interface.
7. *HashSet* : Extends AbstractSet for use with a hash table.
8. *LinkedHashSet* : Extends HashSet to allow insertion-order iterations.
9. *TreeSet*: Implements a set stored in a tree. Extends AbstractSet.
10. *AbstractMap*: Implements most of the Map interface.
11. *HashMap*: Extends AbstractMap to use a hash table.

12. *TreeMap*: Extends AbstractMap to use a tree.
13. *WeakHashMap* : Extends AbstractMap to use a hash table with weak keys.
14. *LinkedHashMap*: Extends HashMap to allow insertion-order iterations.
15. *IdentityHashMap* : Extends AbstractMap and uses reference equality when comparing documents.

Legacy classes

The following are the *legacy classes* for collections.

1. *Vector*: This implements a dynamic array. It is similar to ArrayList, but with some differences.
2. *Stack*: Stack is a subclass of Vector that implements a standard last-in, first-out stack.
3. *Dictionary*: Dictionary is an abstract class that represents a key/value storage repository and operates much like Map.
4. *Hashtable*: Hashtable was part of the original java.util and is a concrete implementation of a Dictionary.
5. *Properties*: Properties is a subclass of Hashtable. It is used to maintain lists of values in which the key is a String and the value is also a String.
6. *BitSet*: A BitSet class creates a special type of array that holds bit values. This array can increase in size as needed.

6.6.2 Basic Methods

A map has the form Map <K, V >where, K specifies the type of keys maintained in this map. V defines the type of mapped values. Furthermore, the Map interface provides a set of methods that must be implemented. Following are some of the methods.

1. *clear*: Removes all the elements from the map.
2. *containsKey*: Returns true if the map contains the requested key.

3. *containsValue*: Returns true if the map contains the requested value.
4. *equals*: Compares an Object with the map for equality.
5. *get*: Retrieve the value of the requested key.
6. *keySet*: Returns a Set that contains all keys of the map.
7. *put*: Adds the requested key-value pair in the map.
8. *remove*: Removes the requested key and its value from the map, if the key exists.
9. *size*: Returns the number of key-value pairs currently in the map.

6.6.3 Map Classes

Hash Map

The most common class that implements the Map interface is the Java HashMap. A HashMap is a hash table based implementation of the Map interface. It permits null keys and values. Also, this class does not maintain any order among its elements and especially, it does not guarantee that the order will remain constant over time. Finally, a HashMap contains two fundamental parameters: initial capacity and performance. The capacity is defined as the number of buckets in the hash table, while the load factor is a measure that indicates the maximum value the hash table can reach, before being automatically increased.

A simple example that uses a HashMap is shown below:

6.6.4 Ex-6: Map example 1

```
import java.util.*;

public class hashEx{

        public static void main(String[] args){

                Map m = new HashMap();
```

```
                m.put("Kama", "40");
                m.put("Sukanya", "38");
                m.put("Ammu", "11");
                m.put("Chikku", "8");

                System.out.println();
                System.out.println("Map Elements");
                System.out.println("\t" + m);
        }
}
```

The above example executes a hashmap object known as m and later prints the values and keys together known as *tuples*. The output will be as below:

```
Map Elements
{Sukanya=38, Kama=40, Ammu=11, Chikku=8}
```

6.7 Arrays

Arrays are important for data analysts. Data analysts deals with data sets that are essentially matrices. A matrix is anything arranged in rows and columns. So, a data set is basically a data matrix set in rows and columns. The order of matrix is denoted in rows by columns (n), which are referred to as dimensions. The following section illustrates as how to deal with data matrices i.e. arrays.

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. Each item in an array is called an element, and each element is accessed by its numerical index. As shown in the preceding illustration, numbering begins with 0. The following program, ArrayDemo, creates an array of integers, puts some values in the array, and prints each value to standard output.

6.7.1 Ex-1: One Dimensional Array

```
class arrEx {
```

```
public static void main(String[] args){

// declares array
        int[] AnArray;
// initiize the array
        AnArray = new int[10];
// initialize elements
        AnArray[0] = 100;
        AnArray[1] = 200;
        AnArray[2] = 300;
        AnArray[3] = 400;
        AnArray[4] = 500;
        AnArray[5] = 600;
        AnArray[6] = 700;
        AnArray[7] = 800;
        AnArray[8] = 900;
        AnArray[9] = 1000;
// output
        System.out.println("The first element in
        ↪  array: " +
        ↪  AnArray[0]);
        System.out.println("The first element in
        ↪  array: " + AnArray[1]);
        System.out.println("The first element in
        ↪  array: " + AnArray[2]);
        System.out.println("The first element in
        ↪  array: " + AnArray[3]);
        System.out.println("The first element in
        ↪  array: " + AnArray[4]);
        System.out.println("The first element in
        ↪  array: " + AnArray[5]);
        System.out.println("The first element in
        ↪  array: " + AnArray[6]);
        System.out.println("The first element in
        ↪  array: " + AnArray[7]);
        System.out.println("The first element in
        ↪  array: " + AnArray[8]);
```

```
                System.out.println("The first element in
                ↪  array: " + AnArray[9]);
        }
}
```

The expectec out put will be

```
The first element in array: 100
The first element in array: 200
The first element in array: 300
The first element in array: 400
The first element in array: 500
The first element in array: 600
The first element in array: 700
The first element in array: 800
The first element in array: 900
The first element in array: 1000
```

The above code defines an array called *AnArray* and reads certain data then finally outputs the same data as by index. The above example shows how to declare and define an array. Lets see how to make a two dimensional array.

Ex-2: I/O operations on Arrays

```
import java.util.Scanner;

class switchEx {

                public static void main(String[] args){

                        double[] a;
                        a = new double[10];
                        //int s = 0;

                        Scanner scan = new
                        ↪  Scanner(System.in);

                        //int sum = 0;
```

```
                        for(int i = 0; i< 10; i++){

                                System.out.println("Enter
                                ↪  input: ");
                                a[i] = scan.nextInt();
                        }

                        for(int i = 0; i < 10; i++){
                                System.out.println(a[i]);
                        }
                }
        }
```

6.7.2 Ex-2: Two Dimensional Array

```
class twoDarray{

        public static void main(String[] args){
                String[][] names = {{"Mr.", "Mrs.",
                ↪  "Ms.", "Mast."}, {"MK", "MS", "Ammu",
                ↪  "Chikku"}};

                System.out.println(names[0][0]+names[1][0]);
                System.out.println(names[0][1]+names[1][1]);
                System.out.println(names[0][2]+names[1][2]);
                System.out.println(names[0][3]+names[1][3]);
        }
}
```

The above code concatenate the two different elements of the rows.

Ex-3: Copying Array

The System class has an arraycopy method that you can use to efficiently copy data from one array into another:

```
class arrCopyDemo {
```

```
        public static void main(String[] args){
                char[] copyFrom = {'a', 'b', 'c', 'd', 'e'};
                char[] copyTo = new char[3];

        System.arraycopy(copyFrom, 2, copyTo, 0, 3);
        System.out.println(new String(copyTo));
        }
}
```

The above code creates two arrays of *char* data type namely *copyFrom* and *copyTo*. The array *copyTo* is created by taking from third element i.e. with index 3, to the last. The expected output is as below:

```
cde
```

Ex-4: Array Manipulation

Arrays are a powerful and useful concept used in programming. Java SE provides methods to perform some of the most common manipulations related to arrays. For instance, the ArrayCopyDemo example uses the arraycopy method of the System class instead of manually iterating through the elements of the source array and placing each one into the destination array. This is performed behind the scenes, enabling the developer to use just one line of code to call the method.

For your convenience, Java SE provides several methods for performing array manipulations (common tasks, such as copying, sorting and searching arrays) in the java.util.Arrays class. For instance, the previous example can be modified to use the copyOfRange method of the java.util.Arrays class, as you can see in the ArrayCopyOfDemo example. The difference is that using the copyOfRange method does not require you to create the destination array before calling the method, because the destination array is returned by the method:

```
import java.util.*;
import java.io.*;

class arrManip {
        public static void main(String[] args){
```

```
                char[] copyFrom = {'a', 'b', 'c', 'd',
                ↪  'e', 'f', 'g', 'h', 'i', 'j', 'k',
                ↪  'l'};
                int size = copyFrom.length;
                char[] copyTo =
                ↪  java.util.Arrays.copyOfRange(copyFrom,
                ↪  5, + size);
                System.out.println(new String(copyTo));
        }
}
```

The above code shows the method of finding *array length* and printing the second array (part of the first array). The same will be different if we are using *only a string*. For instance

```
class strLen {
        public static void main(String[] args){
                String mystr = new String("This is only a
                ↪  test string");
                System.out.println(+ mystr.length());
        }
}
```

You might see the method `.length` is used with paranthesis, whereas in Ex-4 there is no paranthesis. This might be because `char` works differently from `String`. Let us see for `int`.

6.7.3 Vectors

Vectors are very important data type in the domain of data analytics. In univariate analysis, every set of numbers treated as a unique vector. A data distribution is a set of contiguously arranged instances of values arise out of a unique process. So every data distribution is a vector. The same when used by the characters and strings are known as *factor*.

There are other couple of functions or methods to practice on arrays. For instance if we are working with array of integer data type we might be able to user certain methods like *adding,*

Notes

[22]Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

[23]Use `update-alternatives --display java` for display right candidate for cofiguration.

6.8 Questions

6.8.1 Short answer questions

1. What is Java?
2. What are the various features of Java?
3. What is IDE?
4. What is Object Oriented Programming system (OOPs)?
5. What is encapsulation?
6. What is polymorphism?
7. What is a constructor?
8. What is inheritance?
9. What is abstraction?

6.8.2 Essay questions

1. How do you setup Java in Windows and Ubuntu? Write in detail.
2. What are the various IDEs available for Java coding? Explain features of each IDE.
3. What are the various features of OOPs? Explain each features with the help of code.

Chapter 7

Mapreduce

MapReduce is a programming paradigm that runs in the background of Hadoop to provide scalability and easy data-processing solutions. MapReduce is a programming model specifically implemented for processing large data sets. The model was developed by *Jeffrey Dean* and *Sanjay Ghemawat* at Google. At its core, MapReduce is a combination of two functions - *map()* and *reduce()*, as its name would suggest.

Traditional Enterprise Systems normally have a centralized server to store and process data. The following illustration depicts a schematic view of a traditional enterprise system. Traditional model is certainly not suitable to process huge volumes of scalable data and cannot be accommodated by standard database servers. Moreover, the centralized system creates too much of a bottleneck while processing multiple files simultaneously.

Figure 7.1: Centralized systems

Google solved this bottleneck issue using an algorithm called MapReduce. MapReduce divides a task into small parts and assigns them to many computers. Later, the results are collected at one place and integrated to form the result dataset.

Figure 7.2: Decentralized systems

7.1 How MapReduce Works?

The MapReduce algorithm contains two important tasks, namely *Map* and *Reduce*. The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).

The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples. The reduce task is always performed after the map job. Let us now take a close look at each of the phases and try to understand their significance.

Figure 7.3: MapReduce

Different phases in MapReduce:

1. *Input Phase* − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
2. *Map* − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
3. *Intermediate Keys* − They key-value pairs generated by the mapper are known as intermediate keys.
4. *Combiner* − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
5. *Shuffle and Sort* − The Reducer task starts with the Shuffle and

Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.

6. *Reducer* — The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated, filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

7. *Output Phase* — In the output phase, we have an output formatter that translates the final key-value pairs from the Reducer function and writes them onto a file using a record writer.

7.1.1 MR Example

Let us take a real-world example to comprehend the power of MapReduce. Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.

As shown in the illustration, the MapReduce algorithm performs the following actions:

1. *Tokenize* — Tokenizes the tweets into maps of tokens and writes them as key-value pairs.

2. *Filter* — Filters unwanted words from the maps of tokens and writes the filtered maps as key-value pairs.

3. *Count* — Generates a token counter per word.

4. *Aggregate Counters* — Prepares an aggregate of similar counter values into small manageable units.

The MapReduce algorithm contains two important tasks, namely *Map* and *Reduce*.

The map task is done by means of Mapper Class, the reduce task

is done by means of Reducer Class. Mapper class takes the input, tokenizes it, maps and sorts it. The output of Mapper class is used as input by Reducer class, which in turn searches matching pairs and reduces them.

MapReduce implements various mathematical algorithms to divide a task into small parts and assign them to multiple systems. In technical terms, MapReduce algorithm helps in sending the Map Reduce tasks to appropriate servers in a cluster.

These mathematical algorithms may include the following:

1. Searching
2. Indexing
3. TF-IDF

7.1.2 Sorting

Sorting is one of the basic MapReduce algorithms to process and analyze data. MapReduce implements sorting algorithm to automatically sort the output key-value pairs from the mapper by their keys. Sorting methods are implemented in the mapper class itself. In the Shuffle and Sort phase, after tokenizing the values in the mapper class, the *Context class* (user-defined class) collects the matching valued keys as a collection. To collect similar key-value pairs (intermediate keys), the Mapper class takes the help of *RawComparator* class to sort the key-value pairs. The set of intermediate key-value pairs for a given Reducer is automatically sorted by Hadoop to form key-values $(K2, V2, V2,)$ before they are presented to the Reducer.

7.1.3 Searching

Searching plays an important role in MapReduce algorithm. It helps in the combiner phase (optional) and in the Reducer phase. Let us try to understand how Searching works with the help of an example.

Example

The following example shows how MapReduce employs Searching algorithm to find out the details of the employee who draws the highest salary in a given employee dataset. Let us assume we have employee data in four different files − A, B, C, and D. Let us also assume there are duplicate employee records in all four files because of importing the employee data from all database tables repeatedly. See the following illustration.

The Map phase:

The Map phase processes each input file and provides the employee data in key-value pairs (<k, v>: <emp name, salary>). See the following illustration.

The combiner phase (searching technique) will accept the input from the Map phase as a key-value pair with employee name and salary. Using searching technique, the combiner will check all the employee salary to find the highest salaried employee in each file. See the following snippet.

```
<k: employee name, v: salary>
Max= the salary of an first employee. Treated as max
↪  salary

if(v(second employee).salary > Max){
   Max = v(salary);
}

else{
   Continue checking;
}
```

Reducer phase:

Reducer phase − Form each file, you will find the highest salaried employee. To avoid redundancy, check all the <k, v >pairs and eliminate duplicate entries, if any. The same algorithm is used in between the four <k, v >pairs, which are coming from four input files. The final output should be as follows:

7.1.4 Indexing

Normally indexing is used to point to a particular data and its address. It performs batch indexing on the input files for a particular Mapper.

The indexing technique that is normally used in MapReduce is known as inverted index. Search engines like Google and Bing use inverted indexing technique. Let us try to understand how Indexing works with the help of a simple example.

Example

The following text is the input for inverted indexing. Here $T[0], T[1]$, and $t[2]$ are the file names and their content are in double quotes.

```
T[0] = "it is what it is"
T[1] = "what is it"
T[2] = "it is a banana"
```

After applying the Indexing algorithm, we get the following output:

```
"a": {2}
"banana": {2}
"is": {0, 1, 2}
"it": {0, 1, 2}
"what": {0, 1}
```

Here "a" : 2 implies the term "a" appears in the $T[2]$ file. Similarly, "is" : $0, 1, 2$ implies the term "is" appears in the files $T[0], T[1]$, and $T[2]$.

7.1.5 TF-IDF

TF-IDF is a text processing algorithm which is short for Term Frequency − Inverse Document Frequency. It is one of the common web analysis algorithms. Here, the term 'frequency' refers to the number of times a term appears in a document.

TF

It measures how frequently a particular term occurs in a document. It is calculated by the number of times a word appears in a document

divided by the total number of words in that document.

$$TF(the) = \frac{(Numberoftimestermthetheappearsinadocument)}{(Totalnumberoftermsinthedocument)} \tag{7.1}$$

Inverse Document Frequency (IDF)

It measures the importance of a term. It is calculated by the number of documents in the text database divided by the number of documents where a specific term appears.

While computing TF, all the terms are considered equally important. That means, TF counts the term frequency for normal words like "is", "a", "what", etc. Thus we need to know the frequent terms while scaling up the rare ones, by computing the following:

$$IDF(the) = log_e(\frac{Totalnumberofdocuments}{Numberofdocumentswithtermtheinit}). \tag{7.2}$$

Example

Consider a document containing 1000 words, wherein the word hive appears 50 times. The TF for hive is then $(50/1000) = 0.05$.

Now, assume we have 10 million documents and the word hive appears in 1000 of these. Then, the IDF is calculated as $log(10,000,000/1,000) = 4$.

The $TF-IDF$ weight is the product of these quantities: $0.054 = 0.20$.

MapReduce works only on Linux flavored operating systems and it comes inbuilt with a Hadoop Framework. Please refer to Chapter one on "Installation" to know about installing Hadoop in Linux operating system.

7.2 Exercise

We will take a close look at the classes and their methods that are involved in the operations of MapReduce programming. We will primarily keep our focus on the following:

1. JobContext Interface
2. Job Class
3. Mapper Class
4. Reducer Class

The JobContext interface is the super interface for all the classes, which defines different jobs in MapReduce. It gives you a read-only view of the job that is provided to the tasks while they are running. The following are the sub-interfaces of JobContext interface.

Table 7.1: JobContext Interface

S.No.	Subinterface	Description
1	MapContext < KEYIN, VALUEIN, KEYOUT, VALUEOUT >	Defines the context that is given to the Mapper.
2	ReduceContext < KEYIN VALUEIN KEYOUT VALUEOUT >	Defines the context that is passed to the Reducer.

Job class is the main class that implements the JobContext interface.

7.2.1 Job Class

The Job class is the most important class in the MapReduce API. It allows the user to configure the job, submit it, control its execution, and query the state. The set methods only work until the job is submitted, afterwards they will throw an *IllegalStateException*. Normally, the user creates the application, describes the various facets of the job, and then submits the job and monitors its progress.

Here is an example of how to submit a job.

```
// Create a new Job
Job job = new Job(new Configuration());
job.setJarByClass(MyJob.class);
```

```
// Specify various job-specific parameters
job.setJobName("myjob");
job.setInputPath(new Path("in"));
job.setOutputPath(new Path("out"));

job.setMapperClass(MyJob.MyMapper.class);
job.setReducerClass(MyJob.MyReducer.class);

// Submit the job, then poll for progress until the job
↪  is complete
job.waitForCompletion(true);
```

7.2.2 Constructors

Following are the constructor summary of Job class.

S. No.	Constructor summary
1	Job()
2	Job(Configuration conf)
3	Job(Configuration conf, String JobName)

7.2.3 Methods

Some of the important methods of Job class are as follows:

7.2.4 Mapper Class

The Mapper class defines the Map job. Maps input key-value pairs to a set of intermediate key-value pairs. Maps are the individual tasks that transform the input records into intermediate records. The transformed intermediate records need not be of the same type as the input records. A given input pair may map to zero or many output pairs.

S. No.	Method Description	
1	getJobName()	User-specified job name.
2	getJobState()	Returns the current state of the Job.
3	isComplete()	Checks if the job is finished or not.
4	setInputFormatClass()	Sets the InputFormat for the job.
5	setJobName(String name)	Sets the user-specified job name.
6	setOutputFormatClass()	Sets the Output Format for the job.
7	setMapperClass(Class)	Sets the Mapper for the job.
8	setReducerClass(Class)	Sets the Reducer for the job.
9	setPartitionerClass(Class)	Sets the Partitioner for the job.
10	setCombinerClass(Class)	Sets the Combiner for the job.

Method

map is the most prominent method of the Mapper class. This method is called once for each key-value pair in the input split. The syntax is defined below:

```
map(KEYIN key, VALUEIN value,
↪  org.apache.hadoop.mapreduce.Mapper.Context context)
```

7.2.5 Reducer Class

The Reducer class defines the Reduce job in MapReduce. It reduces a set of intermediate values that share a key to a smaller set of values. Reducer implementations can access the Configuration for a job via the *JobContext.getConfiguration()* method. A Reducer has three primary phases — *Shuffle, Sort,* and *Reduce.*

1. *Shuffle* − The Reducer copies the sorted output from each Mapper using HTTP across the network.

2. *Sort* − The framework merge-sorts the Reducer inputs by keys (since different Mappers may have output the same key). The shuffle and sort phases occur simultaneously, i.e., while outputs are being fetched, they are merged.

3. *Reduce* − In this phase the reduce (Object, Iterable, Context) method is called for each key, (collection of values) >in the sorted inputs.

Method

reduce is the most prominent method of the Reducer class. This method is called once for each key on the collection of key-value pairs. The syntax is defined below

```
reduce(KEYIN key, Iterable<VALUEIN> values,
↪  org.apache.hadoop.mapreduce.Reducer.Context context)
```

7.3 MapReduce Algorithm

Generally MapReduce paradigm is based on sending map-reduce programs to computers where the actual data resides.

1. During a MapReduce job, Hadoop sends Map and Reduce tasks to appropriate servers in the cluster.

2. The framework manages all the details of data-passing like issuing tasks, verifying task completion, and copying data around the cluster between the nodes.

3. Most of the computing takes place on the nodes with data on local disks that reduces the network traffic.

4. After completing a given task, the cluster collects and reduces the data to form an appropriate result, and sends it back to the Hadoop server.

7.3.1 Input Output - Java perspective

The MapReduce framework operates on key-value pairs, that is, the framework views the input to the job as a set of key-value pairs and produces a set of key-value pair as the output of the job, conceivably of different types. The key and value classes have to be serializable by the framework and hence, it is required to implement the Writable interface. Additionally, the key classes have to implement the *WritableComparable* interface to facilitate sorting by the framework. Both the input and output format of a MapReduce job are in the form of key-value pairs.

(Input) <k1, v1 > -> map -> <k2, v2 > -> reduce -> <k3, v3 > (Output).

Table 7.2: MapReduce Execution

Step	Input	Output
Map	<k1, v1 >	list (<k2, v2>)
Reduce	<k2, list(v2)>	list (3, v3>)

7.4 Simple MapReduce Examples

There are built-in mapreduce examples. Usually the programs must be executed through *jar* files. These jar files must be created through Java on java class files which in turn obtained through Java compiler. For instance, imagine that we got a *java* program called "wordcount" this file must processed through the below code.

```
javac wordcount.java
jar wordcount # it is a class
```

These things are explained under the section 5.4 mapreduce implementation. These things are for those who are comfortable with writing programs using *core java*. However, for beginners it might not be possible and also cumbersome at first. Naive users might not be able to understand the algorithms and write programs at the brink of the very practice. So, for that there are beautiful mechanisms to practice and understand such programs i.e. MapReduce routines through certain application or file known as *hadoop-mapreduce-examples-[version].jar* located under *share* folder in *hadoop_home*. You can find a separate *hadoop/mapreduce* in share folder where this file *hadoop-mapreduce-examples-[version].jar* can be found.

```
hduser@hadoop:/usr/local/hadoop/share/hadoop/mapreduce\$ ls
hadoop-mapreduce-client-app-2.7.1.jar
↪  hadoop-mapreduce-client-hs-plugins-2.7.1.jar
↪  hadoop-mapreduce-examples-2.7.1.jar  sources
hadoop-mapreduce-client-common-2.7.1.jar
↪  hadoop-mapreduce-client-jobclient-2.7.1.jar        lib
hadoop-mapreduce-client-core-2.7.1.jar
↪  hadoop-mapreduce-client-jobclient-2.7.1-tests.jar
↪  lib-examples
hadoop-mapreduce-client-hs-2.7.1.jar
↪  hadoop-mapreduce-client-shuffle-2.7.1.jar
↪  part-r-00000
hduser@hadoop:/usr/local/hadoop/share/hadoop/mapreduce\$
```

It is easy to know as what programs are supported by *hadoop-mapreduce-examples.*.*.*.jar* but by executing

```
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.2.jar
```

Following are the other examples that you might find more useful for practice.

1. *aggregatewordcount*: An Aggregate based map/reduce program that counts the words in the input files.
2. *aggregatewordhist*: An Aggregate based map/reduce program that computes the histogram of the words in the input files.
3. *bbp*: A map/reduce program that uses Bailey-Borwein-Plouffe to compute exact digits of Pi.
4. *dbcount*: An example job that count the pageview counts from a database.
5. *distbbp*: A map/reduce program that uses a BBP-type formula to compute exact bits of Pi.
6. *grep*: A map/reduce program that counts the matches of a regex in the input.
7. *join*: A job that effects a join over sorted, equally partitioned datasets.

8. *multifilewc*: A job that counts words from several files.
9. *pentomino*: A map/reduce tile laying program to find solutions to pentomino problems.
10. *pi*: A map/reduce program that estimates Pi using a quasi-Monte Carlo method.
11. *randomtextwriter*: A map/reduce program that writes 10GB of random textual data per node.
12. *randomwriter*: A map/reduce program that writes 10GB of random data per node.
13. *secondarysort*: An example defining a secondary sort to the reduce.
14. *sort*: A map/reduce program that sorts the data written by the random writer.
15. *sudoku*: A sudoku solver.
16. *teragen*: Generate data for the terasort.
17. *terasort*: Run the terasort. *terasort* is a benchmark in computing to evaluate the time taken to sort the randomly distributed data in a given computer. This function is used to assess the performance of Hadoop MR in Cluster.
18. *teravalidate*: Checking results of terasort.
19. *wordcount*: A map/reduce program that counts the words in the input files.
20. *wordmean*: A map/reduce program that counts the average length of the words in the input files.
21. *wordmedian*: A map/reduce program that counts the median length of the words in the input files.
22. *wordstandarddeviation*: A map/reduce program that counts the standard deviation of the length of the words in the input files.

One interesting question is as *what is this hadoop-core-*.*.*.jar* and how it is different from *hadoop-common-*.*.*.jar*?

7.4.1 Processing maximum units of electricity consumption

The following table shows the data regarding the electrical consumption of an organization. The table includes the monthly electrical consumption and the annual average for five consecutive years.

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Avg
1979	23	23	2	43	24	25	26	26	26	26	25	26	25
1980	26	27	28	28	28	30	31	31	31	30	30	30	29
1981	31	32	32	32	33	34	35	36	36	34	34	34	34
1984	39	38	39	39	39	41	42	43	40	39	38	38	40
1985	38	39	39	39	39	41	41	41	0	40	39	39	45

Table 7.3: MapReduce Example - Data of Electrical Consumption

We need to write applications to process the input data in the given table to find the year of maximum usage, the year of minimum usage, and so on. This task is easy for programmers with finite amount of records, as they will simply write the logic to produce the required output, and pass the data to the written application. Let us now raise the scale of the input data. Assume we have to analyze the electrical consumption of all the large-scale industries of a particular state. When we write applications to process such bulk data,

1. They will take a lot of time to execute.
2. There will be heavy network traffic when we move data from the source to the network server.

To solve these problems, we have the MapReduce framework. The above data can be saved in a simple text file (.txt), and can be used as input. The below code is related to MapReduce logic for processing data file.

```
package hadoop;

import java.util.*;
import java.io.IOException;
import java.io.IOException;

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 ProcessUnits
{
   //Mapper class
   public static class E_EMapper extends MapReduceBase
   ↪  implements
   Mapper<LongWritable,          /*Input key Type */
   Text,                           /*Input value Type*/
   Text,                           /*Output key Type*/
   IntWritable>                    /*Output value Type*/
   {
      //Map function
      public void map(LongWritable key, Text value,
      ↪  OutputCollector<Text, IntWritable> output,
      ↪  Reporter reporter) throws IOException
      {
         String line = value.toString();
         String lasttoken = null;
         StringTokenizer s = new
         ↪  StringTokenizer(line,"\t");
         String year = s.nextToken();

         while(s.hasMoreTokens()){
            lasttoken=s.nextToken();
         }

         int avgprice = Integer.parseInt(lasttoken);
         output.collect(new Text(year), new
         ↪  IntWritable(avgprice));
      }
   }

   //Reducer class
```

```
public static class E_EReduce extends MapReduceBase
↪  implements
Reducer< Text, IntWritable, Text, IntWritable >
{
   //Reduce function
   public void reduce(Text key, Iterator <IntWritable>
   ↪  values, OutputCollector>Text, IntWritable>
   ↪  output, Reporter reporter) throws IOException
   {
      int maxavg=30;
      int val=Integer.MIN_VALUE;
      while (values.hasNext())
      {
         if((val=values.next().get())>maxavg)
         {
            output.collect(key, new IntWritable(val));
         }
      }
   }
}

//Main function

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

   conf.setJobName("max_eletricityunits");

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

   conf.setMapperClass(E_EMapper.class);
   conf.setCombinerClass(E_EReduce.class);
   conf.setReducerClass(E_EReduce.class);

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

```
      FileInputFormat.setInputPaths(conf, new
      ↪  Path(args[0]));
      FileOutputFormat.setOutputPath(conf, new
      ↪  Path(args[1]));

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

Save the above program into ProcessUnits.java. The compilation and execution of the program is given below. Compilation and

Execution of *ProcessUnits* Program

Let us assume we are in the home directory of Hadoop user (*e.g./home/hadoop*). Follow the following steps to compile and execute the program.

1. Use the following command to create a directory to store the compiled java classes.

   ```
   mkdir units
   ```

2. Download *Hadoop-core-1.2.1.jar* or *Hadoop-core-2.7.2.jar*, which is used to compile and execute the MapReduce program. Download the jar from https://mvnrepository.com/artifact/org.apache.hadoop/hadoop-core/1.2.1 for *1.2.1* or http://mvnrepository.com/artifact/org.apache.hadoop/hadoop-mapreduce-client-core/2.7.2 for *2.7.2*. Let us assume the download folder is */home/hadoop/*.

3. The following commands are used to compile the ProcessUnits.java program and to create a jar for the program.

   ```
   \$ javac -classpath hadoop-core-1.2.1.jar -d units
   ↪  ProcessUnits.java
   \$ jar -cvf units.jar -C units/ .
   ```

4. The following command is used to create an input directory in HDFS.

   ```
   \$HADOOP_HOME/bin/hadoop fs -mkdir input_dir
   ```

5. The following command is used to copy the input file named sample.txt in the input directory of HDFS.

```
\$HADOOP_HOME/bin/hadoop fs -put
↪ /home/hadoop/sample.txt input_dir
```

6. The following command is used to verify the files in the input directory.

```
\$HADOOP_HOME/bin/hadoop fs -ls input_dir/
```

7. The following command is used to run the *Eleunit_max* application by taking input files from the input directory.

```
\$HADOOP_HOME/bin/hadoop jar units.jar
↪ hadoop.ProcessUnits input_dir output_dir
```

8. The following command is used to verify the resultant files in the output folder.

```
\$HADOOP_HOME/bin/hadoop fs -ls output_dir/
```

9. The following command is used to see the output in Part-r-00000 file. This file is generated by HDFS.

```
\$HADOOP_HOME/bin/hadoop fs -cat
↪ output_dir/part-r-00000
```

10. The following command is used to copy the output folder from HDFS to the local file system.

```
\$HADOOP_HOME/bin/hadoop fs -cat
↪ output_dir/part-00000/bin/hadoop dfs -get
↪ output_dir /home/hadoop
```

Notes

[22]Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

[23]Use `update-alternatives --display java` for display right candidate for cofiguration.

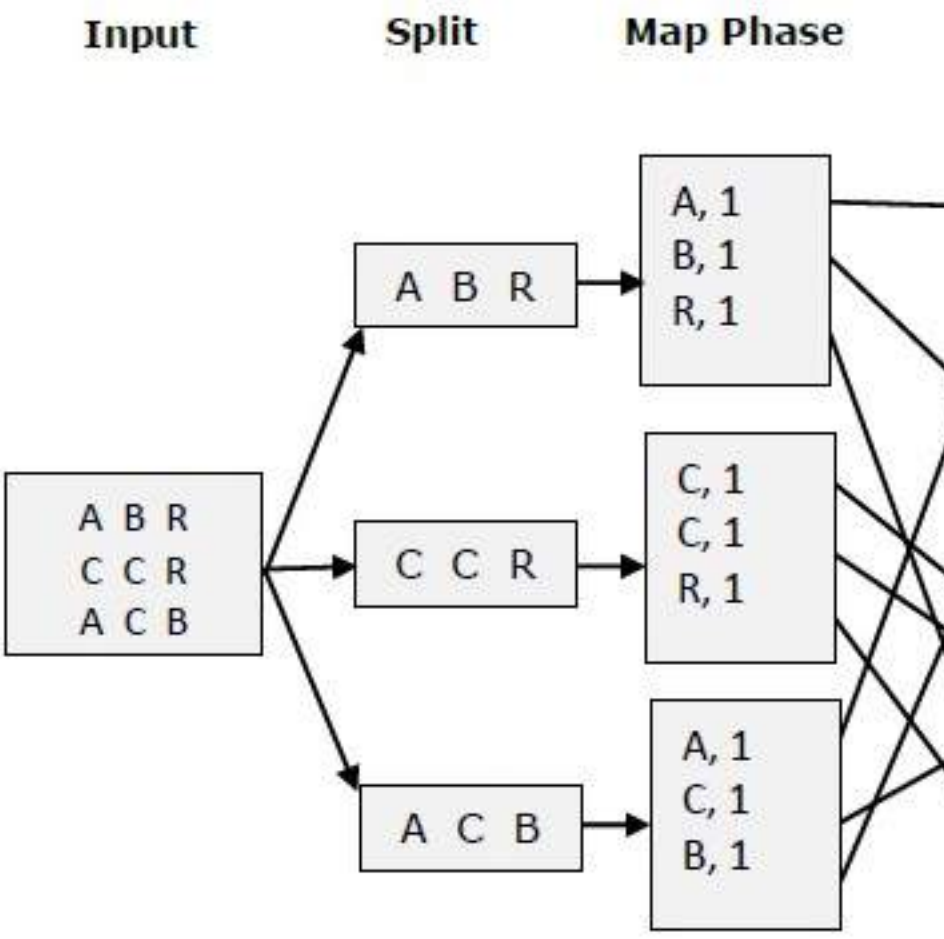
7.5 Questions

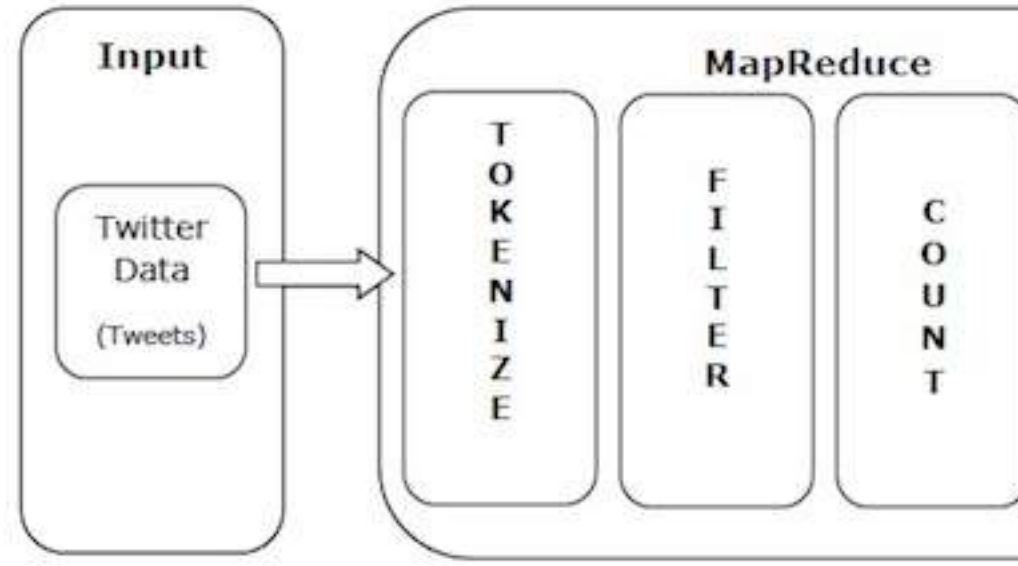
7.5.1 Short answer questions

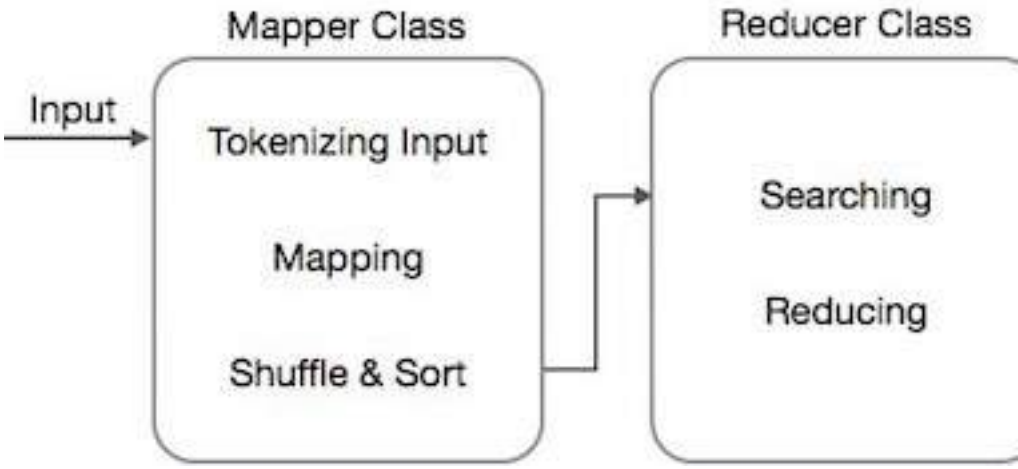
1. What is mapreduce?
2. What is Mapper class?
3. What is Reducer class?
4. What is Combiner class?
5. What is WordCount script?

7.5.2 Essay questions

1. What is mapreduce framework? Elaborate.
2. Write about various classes of mapreduce examples jar.
3. Explain in detail about various classes in side wordcount Java script.


Input
Split
Map Phase
A B R
C C R
A C B
A B R
C C R
A C B
A, 1
B, 1
R, 1
C, 1
C, 1
R, 1
A, 1
C, 1
B, 1


Input
Twitter Data
(Tweets)
MapReduce
TOKENIZE
FILTER
COUNT


Mapper Class
Reducer Class
Input
Tokenizing Input
Mapping
Shuffle & Sort
Searching
Reducing

name, salary
satish, 26000
Krishna, 25000
Satishk, 15000
Raju, 10000

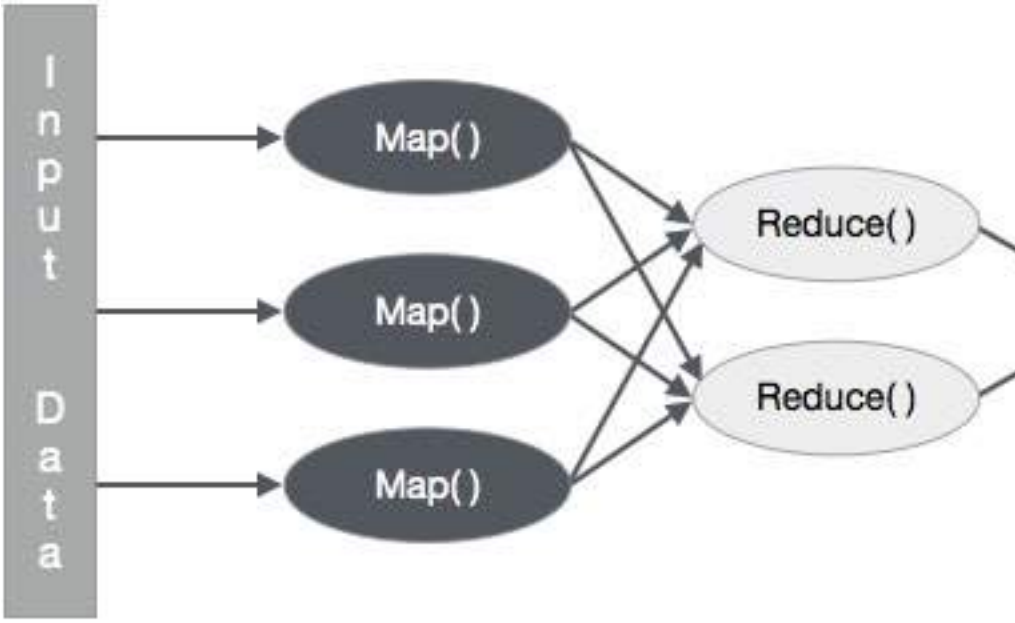
name, salary
gopal, 50000
Krishna, 25000
Satishk, 15000
Raju, 10000

name, salary
satish, 26000
kiran, 45000
Satishk, 15000
Raju, 10000

<satish, 26000>
<Krishna, 25000>
<Satishk, 15000>
<Raju, 10000>

<gopal, 50000>
<Krishna, 25000>
<Satishk, 15000>
<Raju, 10000>

<satish, 26000>
<kiran, 45000>
<Satishk, 15000>
<Raju, 10000>


Input Data
Map()
Map()
Map()
Reduce()
Reduce()

Chapter 8

Pig

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze larger sets of data representing them as data flows. Pig is generally used with Hadoop; we can perform all the data manipulation operations in Hadoop using Pig.To write data analysis programs, Pig provides a high-level language known as *Pig Latin*. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

8.1 History

In 2006, Apache Pig was developed as a research project at Yahoo, especially to create and execute MapReduce jobs on every data set. In 2007, Apache Pig was open sourced via Apache incubator. In 2008, the first release of Apache Pig came out. In 2010, Apache Pig graduated as an Apache top-level project. The language used to analyze

data in Hadoop using Pig is known as Pig Latin. It is a high level data processing language which provides a rich set of data types and operators to perform various operations on the data.

To perform a particular task Programmers using Pig, programmers need to write a Pig script using the Pig Latin language, and execute them using any of the execution mechanisms (Grunt Shell, UDFs, Embedded). After execution, these scripts will go through a series of transformations applied by the Pig Framework, to produce the desired output. Internally, Apache Pig converts these scripts into a series of MapReduce jobs, and thus, it makes the programmer's job easy. The architecture of Apache Pig is shown below.

Why do we need Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.

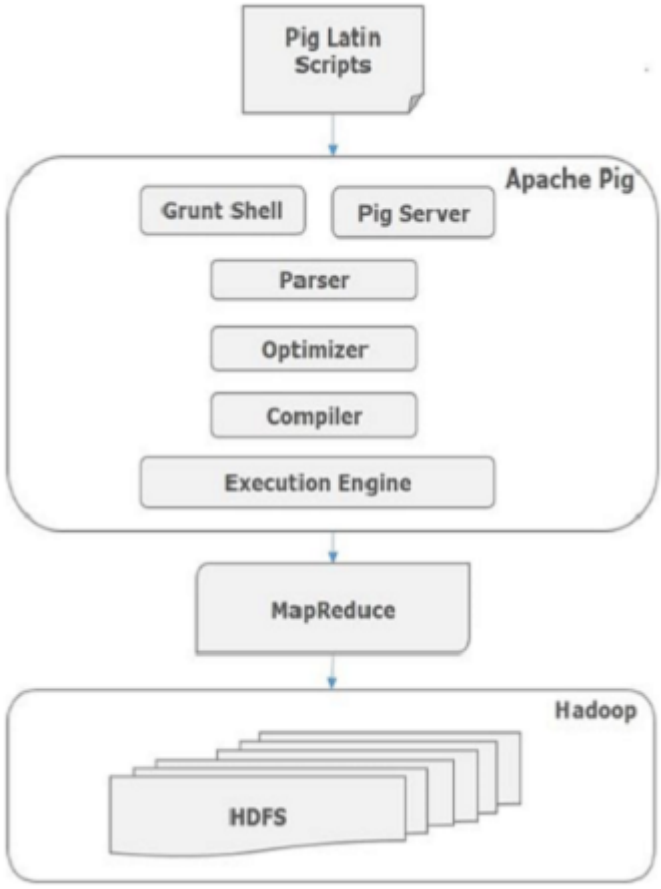
Apache Pig uses multi-query approach, thereby reducing the length of codes. For example, *an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig.* Ultimately Apache Pig reduces the development time by almost 16 times. Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.

Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features.

1. *Rich set of operators* — It provides many operators to perform operations like join, sort, filer, etc.


Pig Latin Scripts
Apache Pig
Grunt Shell
Pig Server
Parser
Optimizer
Compiler
Execution Engine
MapReduce
Hadoop
HDFS

2. *Ease of programming* – Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
3. *Optimization opportunities* – The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
4. *Extensibility* – Using the existing operators, users can develop their own functions to read, process, and write data.
5. *UDF's* – Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.
6. *Handles all kinds of data* – Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

Comparision between Pig and MapReduce

Pig

- Apache Pig is a data flow language.
- It is a high level language.
- Performing a Join operation in Apache Pig is pretty simple.
- Any novice programmer with a basic knowledge of SQL can work conveniently with Apache Pig.
- Apache Pig uses multi-query approach, thereby reducing the length of the codes to a great extent.
- There is no need for compilation. On execution, every Apache Pig operator is converted internally into a MapReduce job.

MapReduce

- MapReduce is a data processing paradigm.
- MapReduce is low level and rigid.
- It is quite difficult in MapReduce to perform a Join operation between datasets.
- Exposure to Java is must to work with MapReduce.

- MapReduce will require almost 20 times more the number of lines to perform the same task.
- MapReduce jobs have a long compilation process.

Comparison between Pig and SQL

Listed below are the major differences between Apache Pig and SQL.

Pig

- Pig Latin is a procedural language.
- In Apache Pig, schema is optional. We can store data without designing a schema (values are stored as $01, $02 etc.)
- The data model in Apache Pig is nested relational.
- Apache Pig provides limited opportunity for Query optimization.

SQL

- SQL is a declarative language.
- Schema is mandatory in SQL.
- The data model used in SQL is flat relational.
- There is more opportunity for query optimization in SQL.

Comparision between Apache Pig Vs Hive

Both Apache Pig and Hive are used to create MapReduce jobs. And in some cases, Hive operates on HDFS in a similar way Apache Pig does. In the following table, we have listed a few significant points that set Apache Pig apart from Hive.

Pig

- Apache Pig uses a language called Pig Latin. It was originally created at Yahoo.
- Pig Latin is a data flow language.
- Pig Latin is a procedural language and it fits in pipeline paradigm.
- Apache Pig can handle

structured, unstructured, and semi-structured data.	• HiveQL is a query processing language.
Hive	• HiveQL is a declarative language.
• Hive uses a language called HiveQL. It was originally created at Facebook.	• Hive is mostly for structured data.

Applications of Apache Pig

Apache Pig is generally used by data scientists for performing tasks involving ad-hoc processing and quick prototyping. Apache Pig is used

1. To process huge data sources such as web logs.
2. To perform data processing for search platforms.
3. To process time sensitive data loads.

8.2 Installation

Installation of Pig is strait forward. Just extract the compressed binaries to certain folder, update the path and configure.

Go to `http://mirror.fibergrid.in/apache/pig/latest/` download both pig-0.16.0-src.tar.gz and pig-0.16.0.tar.gz. But it is always better to visit the main portal `https://pig.apache.org/` for updates and links. You may find *release pages* at this portal which redirects you to a couple of other places so that you may be able to find aforementioned two compressed packages (in *.tar.gz*).

8.2.1 Procedure

Follow the below steps to install Pig in your Linux machine.

1. *Step 1*: Create a directory with the name Pig in the same directory where the installation directories of Hadoop, Java, and other software were installed. (In our tutorial, we have created the Pig directory in the user named Hadoop). `mkdir Pig`

2. *Step 2*: Extract the source file into the directory created in above step. `tar zxvf pig-0.16.0.tar.gz` anywhere you may like to do it.
3. *Step 3*: Move the (extracted) source files to the Pig directory. `mv pig-0.16.0.tar.gz/* /usr/local/Pig`. The path */usr/local/Pig* is my path, you may create some such path anywhere but preferably in directory where hadoop is installed.

Configure

After installing Apache Pig, we have to configure it. To configure, we need to edit two files — *bashrc* and *pig.properties*.

Open *bashrc* file and add the following lines

```
export PIG_HOME = /home/Hadoop/Pig
export PATH  = PATH:/home/Hadoop/pig/bin
export PIG_CLASSPATH = \$HADOOP_HOME/conf
```

Now you may try `souce .bashrc` to save the changes to the file. Go back to parent directy there try `pig -version`. You must be able to see the follwoing response in the terminal.

```
Apache Pig version 0.16.0 (r1746530)
compiled Jun 01 2016, 23:10:49
```

Check the properties (located in $PIG_HOME/conf$ directory with the help of `sudo gedit` $PIG_HOME/conf/pig.properties$.

8.3 Apache Pig Components

As shown in the figure, there are various components in the Apache Pig framework. Let us take a look at the major components.

Parser

Initially the Pig Scripts are handled by the Parser. It checks the syntax of the script, does type checking, and other miscellaneous checks. The output of the parser will be a DAG (directed acyclic graph), which represents the Pig Latin statements and logical operators.

In the DAG, the logical operators of the script are represented as the nodes and the data flows are represented as edges.

A directed acyclic graph (DAG), is a finite directed graph with no directed cycles. That is, it consists of finitely many vertices and edges, with each edge directed from one vertex to another, such that there is no way to start at any vertex v and follow a consistently-directed sequence of edges that eventually loops back to v again. Equivalently, a DAG is a directed graph that has a topological ordering, a sequence of the vertices such that every edge is directed from earlier to later in the sequence.
Though it seems that the performance of algorithm is in random steps, but sicientist found that there exists certain *topological ordering* in such random steps. These methods are covered under *topological sorting algorithms*, one of the oldest application that use such sorting is *PERT*. A. B. Khan (1962), perhaps, the very first person to write about such techniques. His study on *non-empty acyclic graphs* paved a way for topological ordering. The other algorithm, of course, is *depth first search* explained by Tarjan (1976) later Cormen *et al* in 2001. [24]

Optimizer

The logical plan (DAG) is passed to the logical optimizer, which carries out the logical optimizations such as projection and pushdown.

Compiler

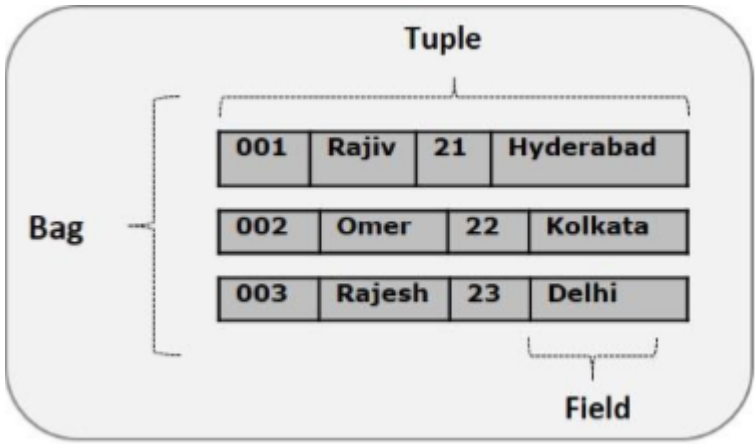
The compiler compiles the optimized logical plan into a series of MapReduce jobs.

Execution engine

Finally the MapReduce jobs are submitted to Hadoop in a sorted order. Finally, these MapReduce jobs are executed on Hadoop producing the desired results.

8.4 Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as map and tuple. Given below is the diagrammatical representation of Pig Latin's data model.



1. *Atom*: Any single value in Pig Latin, irrespective of their data, type is known as an Atom. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a field. Ex. 'Kamakshaiah' or '40' etc.

2. *Tuple*: A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS. Ex. (Sukanya, 38)

3. *Bag*: A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by ''. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type. A bag can be a field in a relation; in that

context, it is known as inner bag. Ex. (Amrutha, 11), (Bhargav, 6)

4. *Map*: A map (or data map) is a set of key-value pairs. The key needs to be of type chararray and should be unique. The value might be of any type. It is represented by '[]'. Ex. [nameKamakshaiah, age40]

5. *Relation*: A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

8.5 Apache Pig Execution Mode

You can run Apache Pig in two modes, namely, Local Mode and HDFS mode.

Local Mode

In this mode, all the files are installed and run from your local host and local file system. There is no need of Hadoop or HDFS. This mode is generally used for testing purpose.

MapReduce Mode

MapReduce mode is where we load or process the data that exists in the Hadoop File System (HDFS) using Apache Pig. In this mode, whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the back-end to perform a particular operation on the data that exists in the HDFS.

8.6 Apache Pig Execution Mechanisms

Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

1. *Interactive Mode (Grunt shell)* – You can run Apache Pig in interactive mode using the Grunt shell. In this shell, you can

enter the Pig Latin statements and get the output (using Dump operator).

2. *Batch Mode (Script)* − You can run Apache Pig in Batch mode by writing the Pig Latin script in a single file with .pig extension.

3. *Embedded Mode (UDF)* − Apache Pig provides the provision of defining our own functions (User Defined Functions) in programming languages such as Java, and using them in our script.

8.7 Invoking methods

8.7.1 *Grunt* Shell

PIG can be invoked in three different modes viz. (1) local mode, (2) mapreduce mode. There is also third mode known as *interactive mode*. You can invoke the Grunt shell in a desired mode (local/MapReduce) using the −x option as shown below.

```
pig -x local
pig or pig -x mapreduce
```

In local mode, all files are installed and run using your local host and file system. Specify local mode using the -x flag (pig -x local). Note that local mode does not support parallel mapper execution with Hadoop 0.20.x and 1.0.0.

8.8 Running modes

8.8.1 Interactive mode

You can run Pig in interactive mode using the Grunt shell. Invoke the Grunt shell using the "pig" command (as shown below) and then enter your Pig Latin statements and Pig commands interactively at the command line. Below Pig Latin statements extract all user IDs from the */etc/passwd* file. First, copy the */etc/passwd* file to your local working directory. Next, invoke the Grunt shell by typing the "pig" command (in local or hadoop mode). Then, enter the Pig Latin statements interactively at the grunt prompt (be sure to include the

semicolon after each statement). The DUMP operator will display the results to your terminal screen.

```
A = load 'passwd' using PigStorage(':');
B = foreach A generate \$0 as id;
dump B;
```

8.8.2 Batch mode

You can run Pig in batch mode using Pig scripts and the "pig" command (in local or hadoop mode). The Pig Latin statements in the Pig script (id.pig) extract all user IDs from the */etc/passwd* file. First, copy the */etc/passwd* file to your local working directory. Next, run the Pig script from the command line (using local or mapreduce mode). The STORE operator will write the results to a file (id.out).

```
/* id.pig */

A = load 'passwd' using PigStorage(':');  -- load the
↪  passwd file
B = foreach A generate \$0 as id;  -- extract the user
↪  IDs
store B into 'id.out';  -- write the results to a file
↪  name id.out
```

8.8.3 Shell Scripts

PIG shell (`grunt>`) has provision to execute the shell scripts. The user need not switch between BASH and GRUNT to and forth. However they are very few in number.

For instance to know

1.

Notes

[24] http://dl.acm.org/citation.cfm?doid=368996.369025

8.9 Questions

8.9.1 Short answer questions

1. What is Pig?
2. What are the prerequisites for Pig?
3. What are the various ways to run Apache Pig?
4. What is grunt shell?

8.9.2 Essay questions

1. What are the various components of Pig?
2. Write about Pig Latin data model.
3. Explain Pig Latin commands with the help of examples.

Chapter 9

Hive

The Apache Hive data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage and queried using SQL syntax. Built on top of Apache Hadoop, Hive provides the following features:

1. Tools to enable easy access to data via SQL, thus enabling data warehousing tasks such as extract/transform/load (ETL), reporting, and data analysis.
2. A mechanism to impose structure on a variety of data formats
3. Access to files stored either directly in Apache HDFS or in other data storage systems such as Apache HBase.
4. Query execution via Apache Tez, Apache Spark, or MapReduce
5. Procedural language with HPL-SQL
6. Sub-second query retrieval via Hive LLAP, Apache YARN and Apache Slider.

Hive provides standard SQL functionality, including many of the later SQL:2003, SQL:2011, and SQL:2016 features for analytics. Hive's SQL can also be extended with user code via user defined functions (UDFs),

user defined aggregates (UDAFs), and user defined table functions (UDTFs).

There is not a single "Hive format" in which data must be stored. Hive comes with built in connectors for comma and tab-separated values (CSV/TSV) text files, Apache Parquet, Apache ORC, and other formats. Users can extend Hive with connectors for other formats. Please see File Formats and Hive SerDe in the Developer Guide for details. Hive is not designed for online transaction processing (OLTP) workloads. It is best used for traditional data warehousing tasks. Hive is designed to maximize scalability (scale out with more machines added dynamically to the Hadoop cluster), performance, extensibility, fault-tolerance, and loose-coupling with its input formats. Components of Hive include HCatalog and WebHCat.

1. HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools — including Pig and MapReduce — to more easily read and write data on the grid.

2. WebHCat provides a service that you can use to run Hadoop MapReduce (or YARN), Pig, Hive jobs. You can also perform Hive metadata operations using an HTTP (REST style) interface.

9.1 Installation and Configuration

Hive can be installed as a stable release of Hive by downloading a tarball, or you can download the source code and build Hive from that.

9.1.1 Requirements

1. Java 1.7. Hive versions 1.2 onward require Java 1.7 or newer. Hive versions 0.14 to 1.1 work with Java 1.6 as well. Users are strongly advised to start moving to Java 1.8.

2. Hadoop 2.x (preferred), 1.x (not supported by Hive 2.0.0 onward). Hive versions up to 0.13 also supported Hadoop 0.20.x,

0.23.x.

3. Hive is commonly used in production Linux and Windows environment. Mac is a commonly used development environment. The instructions in this document are applicable to Linux and Mac. Using it on Windows would require slightly different steps.

9.1.2 Installing Hive from a Stable Release

Start by downloading the most recent stable release of Hive from one of the Apache download mirrors (see Hive Releases). Next you need to unpack the tarball. This will result in the creation of a subdirectory named hive-x.y.z (where x.y.z is the release number):

```
tar -xzvf hive-x.y.z.tar.gz
```

Set the environment variable $HIVE\_HOME$ to point to the installation directory:

```
cd hive-x.y.z
export HIVE_HOME={{pwd}}
```

Finally, add $HIVE\_HOME/bin$ to your PATH:

```
export PATH=$HIVE_HOME/bin:$PATH
```

9.2 Running Hive

Hive uses Hadoop, so:

Hadoop must be in the path OR $exportHADOOP_HOME =< hadoop-install-dir >$. In addition, you must use below HDFS commands to create /tmp and /user/hive/warehouse (aka hive.metastore.warehouse.dir) and set them chmod g+w before you can create a table in Hive.

```
$HADOOP_HOME/bin/hadoop fs -mkdir       /tmp
$HADOOP_HOME/bin/hadoop fs -mkdir       /user/hive/warehouse
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /tmp
$HADOOP_HOME/bin/hadoop fs -chmod g+w   /user/hive/warehouse
```

You may find it useful, though it's not necessary, to set $HIVE\_HOME$:

```
export HIVE_HOME=<hive-install-dir>
```

9.2.1 Running Hive CLI

To use the Hive command line interface (CLI) from the shell:

```
\$HIVE_HOME/bin/hive
```

Starting from Hive 2.1, we need to run the schematool command below as an initialization step. For example, we can use "derby" as db type.

```
\$HIVE_HOME/bin/schematool -dbType <db type> -initSchema
```

HiveServer2 (introduced in Hive 0.11) has its own CLI called Beeline. HiveCLI is now deprecated in favor of Beeline, as it lacks the multi-user, security, and other capabilities of HiveServer2. To run HiveServer2 and Beeline from shell:

```
\$HIVE_HOME/bin/hiveserver2
\$HIVE_HOME/bin/beeline -u jdbc:hive2://$HS2_HOST:$HS2_PORT
```

Beeline is started with the JDBC URL of the HiveServer2, which depends on the address and port where HiveServer2 was started. By default, it will be (localhost:10000), so the address will look like jdbc:hive2://localhost
Or to start Beeline and HiveServer2 in the same process for testing purpose, for a similar user experience to HiveCLI:

```
\$HIVE_HOME/bin/beeline -u jdbc:hive2://
```

9.2.2 Running HCatalog

To run the HCatalog server from the shell in Hive release 0.11.0 and later:

```
\$HIVE_HOME/hcatalog/sbin/hcat_server.sh
```

To use the HCatalog command line interface (CLI) in Hive release 0.11.0 and later:

```
\$HIVE_HOME/hcatalog/bin/hcat
```

For more information, see HCatalog Installation from Tarball and HCatalog CLI in the HCatalog manual.

9.2.3 Running WebHCat (Templeton)

To run the WebHCat server from the shell in Hive release 0.11.0 and later:

```
\$HIVE_HOME/hcatalog/sbin/webhcat_server.sh
```

9.3 Configuration Management Overview

1. Hive by default gets its configuration from $< install-dir > /conf/hive-default.xml$
2. The location of the Hive configuration directory can be changed by setting the $HIVE\_CONF\_DIR$ environment variable.
3. Configuration variables can be changed by (re-)defining them in $< install-dir > /conf/hive-site.xml$ Log4j configuration is stored in $< install-dir > /conf/hive-log4j.properties$
4. Hive configuration is an overlay on top of Hadoop – it inherits the Hadoop configuration variables by default.
5. Hive configuration can be manipulated by:
   - Editing hive-site.xml and defining any desired variables (including Hadoop variables) in it
   - Using the set command (see next section)
   - Invoking Hive (deprecated), Beeline or HiveServer2 using the syntax:
     - `bin/hive --hiveconf x1=y1 --hiveconf x2=y2 //this sets the variables x1 and x2 to y1 and y2 respectively`
     - `bin/hiveserver2 --hiveconf x1=y1 --hiveconf x2=y2 //this sets server-side variables x1 and x2 to y1 and y2 respectively`
     - `bin/beeline --hiveconf x1=y1 --hiveconf x2=y2 //this sets client-side variables x1 and x2 to y1 and y2 respectively.`

Setting the $HIVE\_OPTS$ environment variable to "–hiveconf x1=y1 –hiveconf x2=y2" which does the same as above.

9.3.1 Runtime Configuration

Hive queries are executed using map-reduce queries and, therefore, the behavior of such queries can be controlled by the Hadoop configuration variables. The HiveCLI (deprecated) and Beeline command 'SET' can be used to set any Hadoop (or Hive) configuration variable. For example:

```
beeline> SET mapred.job.tracker=myhost.mycompany.com:50030;
    beeline> SET -v;
```

The latter shows all the current settings. Without the `-v` option only the variables that differ from the base Hadoop configuration are displayed.

9.4 Hive, Map-Reduce and Local-Mode

Hive compiler generates map-reduce jobs for most queries. These jobs are then submitted to the Map-Reduce cluster indicated by the variable:

```
mapred.job.tracker
```

While this usually points to a map-reduce cluster with multiple nodes, Hadoop also offers a nifty option to run map-reduce jobs locally on the user's workstation. This can be very useful to run queries over small data sets – in such cases local mode execution is usually significantly faster than submitting jobs to a large cluster. Data is accessed transparently from HDFS. Conversely, local mode only runs with one reducer and can be very slow processing larger data sets. Starting with release 0.7, Hive fully supports local mode execution. To enable this, the user can enable the following option:

```
hive> SET mapreduce.framework.name=local;
```

In addition, mapred.local.dir should point to a path that's valid on the

local machine (for example /tmp/<username>/mapred/local). (Otherwise, the user will get an exception allocating local disk space.)

Starting with release 0.7, Hive also supports a mode to run map-reduce jobs in local-mode automatically. The relevant options are *hive.exec.mode.local.auto*, *hive.exec.mode.local.auto.inputbytes.max*, and *hive.exec.mode.local.auto.tasks.max*:

```
hive> SET hive.exec.mode.local.auto=false;
```

Note that this feature is disabled by default. If enabled, Hive analyzes the size of each map-reduce job in a query and may run it locally if the following thresholds are satisfied:

1. The total input size of the job is lower than:
   hive.exec.mode.local.auto.inputbytes.max (128MB by default)

2. The total number of map-tasks is less than:
   hive.exec.mode.local.auto.tasks.max (4 by default)

3. The total number of reduce tasks required is 1 or 0.

So for queries over small data sets, or for queries with multiple map-reduce jobs where the input to subsequent jobs is substantially smaller (because of reduction/filtering in the prior job), jobs may be run locally. Note that there may be differences in the runtime environment of Hadoop server nodes and the machine running the Hive client (because of different jvm versions or different software libraries). This can cause unexpected behavior/errors while running in local mode. Also note that local mode execution is done in a separate, child jvm (of the Hive client). If the user so wishes, the maximum amount of memory for this child jvm can be controlled via the option hive.mapred.local.mem. By default, it's set to zero, in which case Hive lets Hadoop determine the default memory limits of the child jvm.

9.5 Data types

In the order of granularity (i.e., many pieces and elements) - Hive data is organized into:

1. *Databases*: Namespaces function to avoid naming conflicts for tables, views, partitions, columns, and so on. Databases can also be used to enforce security for a user or group of users.

2. *Tables*: Homogeneous units of data which have the same schema. An example of a table could be *page_views* table, where each row could comprise of the following columns (schema):

   - *timestamp*—which is of INT type that corresponds to a UNIX *timestamp* of when the page was viewed.
   - *userid* —which is of BIGINT type that identifies the user who viewed the page.
   - *page_url*—which is of STRING type that captures the location of the page.
   - *referer_url*—which is of STRING that captures the location of the page from where the user arrived at the current page.
   - `IP`—which is of STRING type that captures the IP address from where the page request was made.

3. *Partitions*: Each Table can have one or more partition Keys which determines how the data is stored. Partitions—apart from being storage units—also allow the user to efficiently identify the rows that satisfy a specified criteria; for example, a *date_partition* of type STRING and *country_partition* of type STRING. Each unique value of the partition keys defines a partition of the Table. For example, all "US" data from "2009-12-23" is a partition of the *page_views* table. Therefore, if you run analysis on only the "US" data for $2009 - 12 - 23$, you can run that query only on the relevant partition of the table, thereby speeding up the analysis significantly. Note however, that just because a partition is named $2009 - 12 - 23$ does not mean that it contains all or only data from that date; partitions are named after dates for convenience; it is the user's job to guarantee the relationship between partition name and data content! Partition columns are virtual columns, they are not part of the data itself but are derived on load.

4. *Buckets (or Clusters)*: Data in each partition may in turn be divided into Buckets based on the value of a hash function of some column of the Table. For example the *page_views* table may be bucketed by *userid*, which is one of the columns, other than the partitions columns, of the *page_views* table. These can be used to efficiently sample the data. Note that it is not necessary for tables to be partitioned or bucketed, but these abstractions allow the system to prune large quantities of data during query processing, resulting in faster query execution.

Note that it is not necessary for tables to be partitioned or bucketed, but these abstractions allow the system to prune large quantities of data during query processing, resulting in faster query execution.

9.5.1 Type System

Hive supports primitive and complex data types, as described below.

Primitive Types

Types are associated with the columns in the tables. The following Primitive types are supported:

1. Integers
   - TINYINT—1 byte integer
   - SMALLINT—2 byte integer
   - INT—4 byte integer
   - BIGINT—8 byte integer
2. Boolean type
   - BOOLEAN—TRUE/FALSE
3. Floating point numbers
   - FLOAT—single precision
   - DOUBLE—Double precision
4. Fixed point numbers

- DECIMAL—a fixed point value of user defined scale and precision

5. String types
   - STRING—sequence of characters in a specified character set
   - VARCHAR—sequence of characters in a specified character set with a maximum length
   - CHAR—sequence of characters in a specified character set with a defined length
6. Date and time types
   - TIMESTAMP — A date and time without a timezone ("LocalDateTime" semantics)
   - TIMESTAMP WITH LOCAL TIME ZONE — A point in time measured down to nanoseconds ("Instant" semantics)
   - DATE—a date
7. Binary types
   - BINARY—a sequence of bytes

The Types are organized in the following hierarchy, where the parent is a super type of all the children instances (refer 9.1):

This type hierarchy defines how the types are implicitly converted in the query language. Implicit conversion is allowed for types from child to an ancestor. So when a query expression expects type1 and the data is of type2, type2 is implicitly converted to type1 if type1 is an ancestor of type2 in the type hierarchy. Note that the type hierarchy allows the implicit conversion of STRING to DOUBLE.

9.5.2 Complex Types

Complex Types can be built up from primitive types and other composite types using:

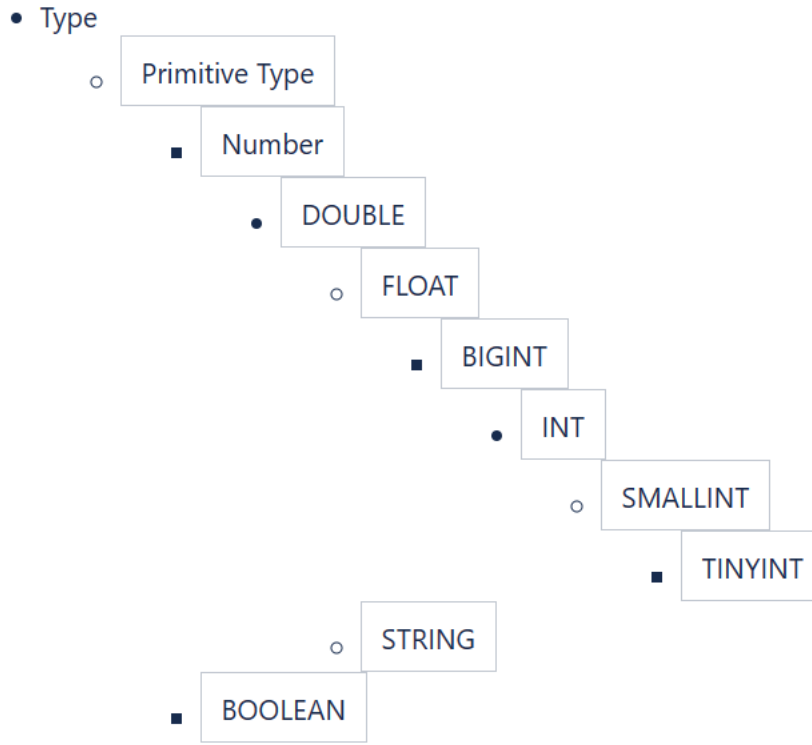


Figure 9.1: Hive data hierarchy

1. Structs: the elements within the type can be accessed using the DOT (.) notation. For example, for a column c of type STRUCT a INT; b INT, the a field is accessed by the expression c.a

2. Maps (key-value tuples): The elements are accessed using ['element name'] notation. For example in a map M comprising of a mapping from 'group' -> gid the gid value can be accessed using M['group']

3. Arrays (indexable lists): The elements in the array have to be in the same type. Elements can be accessed using the [n] notation

where n is an index (zero-based) into the array. For example, for an array A having the elements ['a', 'b', 'c'], A[1] retruns 'b'.

Using the primitive types and the constructs for creating complex types, types with arbitrary levels of nesting can be created. For example, a type User may comprise of the following fields:

1. gender—which is a STRING.
2. active—which is a BOOLEAN.

9.6 Built In Operators and Functions

The operators and functions listed below are not necessarily up to date. In Beeline or the Hive CLI, use these commands to show the latest documentation:

```
SHOW FUNCTIONS;
DESCRIBE FUNCTION <function_name>;
DESCRIBE FUNCTION EXTENDED <function_name>;
```

9.6.1 Built In Operators

1. Relational Operators—The following operators compare the passed operands and generate a TRUE or FALSE value, depending on whether the comparison between the operands holds or not (refer to 9.1).
2. Arithmetic Operators—The following operators support various common arithmetic operations on the operands. All of them return number types (refere to 9.2).
3. Logical Operators — The following operators provide support for creating logical expressions. All of them return boolean TRUE or FALSE depending upon the boolean values of the operands (refer to 9.3).
4. Operators on Complex Types—The following operators provide mechanisms to access elements in Complex Types (refer to 9.4).

Operator	type	desc.
A = B	all primitive types	TRUE or FALSE
A != B	all primitive types	TRUE or FALSE
A < B	all primitive types	TRUE or FALSE
A <= B	all primitive types	TRUE or FALSE
A > B	all primitive types	TRUE or FALSE
A >= B	all primitive types	TRUE or FALSE
A IS NULL	all types	TRUE or FALSE
A IS NOT NULL	all types	TRUE or FALSE
A LIKE B	strings	TRUE or FALSE
A RLIKE B	strings	TRUE or FALSE
A REGEXP B	strings	Same as RLIKE

Table 9.1: Relational opertors

Operator	type	desc.
A + B	all number types	Gives the result of adding A and B.
A - B	all number types	Gives the result of subtracting B from A.
A * B	all number types	Gives the result of multiplying A and B.
A / B	all number types	Gives the result of dividing B from A.
A % B	all number types	Gives the reminder resulting from dividing A by B.
A & B	all number types	Gives the result of bitwise AND of A and B.
A | B	all number types	Gives the result of bitwise OR of A and B.
A ^ B	all number types	Gives the result of bitwise XOR of A and B.
~A	all number types	Gives the result of bitwise NOT of A.

Table 9.2: Arithmetic operators

9.7 Few examples

9.7.1 Browsing Tables and Partitions

For instance, following operations can be performed on existing tables.

```
SHOW TABLES;
```

To list existing tables in the warehouse; there are many of these, likely more than you want to browse.

```
SHOW TABLES 'page.*';
```

To list tables with prefix 'page'. The pattern follows Java regular expression syntax (so the period is a wildcard).

```
SHOW PARTITIONS page_view;
```

Operator	type	desc.
A AND B	boolean	TRUE if both A and B are TRUE, otherwise FALSE
A && B	boolean	Same as A AND B
A OR B	boolean	TRUE if either A or B or both are TRUE, otherwise FALSE
A || B	boolean	Same as A OR B
NOT A	boolean	TRUE if A is FALSE, otherwise FALSE
!A	boolean	Same as NOT A

Table 9.3: Logical operators

Operator	type	desc.
A[n]	A is an Array and n is an int	returns the nth element in the array A.
M[key]	M is a Map<K, V> and key has type K	returns the value corresponding to the key in the map.
S.x	S is a struct	returns the x field of S.

Table 9.4: Logical operators

To list partitions of a table. If the table is not a partitioned table then an error is thrown.

```
DESCRIBE page_view;
```

To list columns and column types of table.

```
DESCRIBE EXTENDED page_view;
```

To list columns and all other properties of table. This prints lot of information and that too not in a pretty format. Usually used for debugging.

```
DESCRIBE EXTENDED page_view PARTITION (ds='2008-08-08');
```

To list columns and all other properties of a partition. This also prints lot of information which is usually used for debugging.

9.7.2 Creating Tables

An example statement that would create the *page_view* table mentioned above would be like:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
STORED AS SEQUENCEFILE;
```

In this example, the columns of the table are specified with the corresponding types. Comments can be attached both at the column level as well as at the table level. Additionally, the partitioned by clause defines the partitioning columns which are different from the data columns and are actually not stored with the data. When specified in this way, the data in the files is assumed to be delimited with ASCII 001(ctrl-A) as the field delimiter and newline as the row delimiter.

The field delimiter can be parametrized if the data is not in the above format as illustrated in the following example:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '1'
STORED AS SEQUENCEFILE;
```

The row delimintor currently cannot be changed since it is not determined by Hive but Hadoop delimiters.

It is also a good idea to bucket the tables on certain columns so that efficient sampling queries can be executed against the data set. If bucketing is absent, random sampling can still be done on the table but it is not efficient as the query has to scan all the data. The following example illustrates the case of the *page_view* table that is bucketed on the userid column:

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '1'
        COLLECTION ITEMS TERMINATED BY '2'
        MAP KEYS TERMINATED BY '3'
```

```
STORED AS SEQUENCEFILE;
```

In the example above, the table is clustered by a hash function of userid into 32 buckets. Within each bucket the data is sorted in increasing order of viewTime. Such an organization allows the user to do efficient sampling on the clustered column—n this case userid. The sorting property allows internal operators to take advantage of the better-known data structure while evaluating queries with greater efficiency.

```
CREATE TABLE page_view(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                friends ARRAY<BIGINT>, properties MAP<STRING, STRING>
                ip STRING COMMENT 'IP Address of the User')
COMMENT 'This is the page view table'
PARTITIONED BY(dt STRING, country STRING)
CLUSTERED BY(userid) SORTED BY(viewTime) INTO 32 BUCKETS
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY '1'
        COLLECTION ITEMS TERMINATED BY '2'
        MAP KEYS TERMINATED BY '3'
STORED AS SEQUENCEFILE;
```

In this example, the columns that comprise of the table row are specified in a similar way as the definition of types. Comments can be attached both at the column level as well as at the table level. Additionally, the partitioned by clause defines the partitioning columns which are different from the data columns and are actually not stored with the data. The CLUSTERED BY clause specifies which column to use for bucketing as well as how many buckets to create. The delimited row format specifies how the rows are stored in the hive table. In the case of the delimited format, this specifies how the fields are terminated, how the items within collections (arrays or maps) are terminated, and how the map keys are terminated. STORED AS SEQUENCEFILE indicates that this data is stored in a binary format (using hadoop SequenceFiles) on hdfs. The values shown for the ROW FORMAT and STORED AS clauses in the above, example represent the system defaults. Table names and column names are case insensitive.

9.7.3 Altering Tables

To rename existing table to a new name. If a table with new name already exists then an error is returned:

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

To rename the columns of an existing table. Be sure to use the same column types, and to include an entry for each preexisting column:

```
ALTER TABLE old_table_name REPLACE COLUMNS (col1 TYPE, ...);
```

To add columns to an existing table:

```
ALTER TABLE tab1 ADD COLUMNS (c1 INT COMMENT 'a new int column',
c2 STRING DEFAULT 'def val');
```

Note that a change in the schema (such as the adding of the columns), preserves the schema for the old partitions of the table in case it is a partitioned table. All the queries that access these columns and run over the old partitions implicitly return a null value or the specified default values for these columns. In the later versions, we can make the behavior of assuming certain values as opposed to throwing an error in case the column is not found in a particular partition configurable.

9.7.4 Dropping Tables and Partitions

Dropping tables is fairly trivial. A drop on the table would implicitly drop any indexes(this is a future feature) that would have been built on the table. The associated command is:

```
DROP TABLE pv_users;
```

To dropping a partition. Alter the table to drop the partition.

```
ALTER TABLE pv_users DROP PARTITION (ds='2008-08-08')
```

Note that any data for this table or partitions will be dropped and may not be recoverable. *

9.7.5 Loading Data

There are multiple ways to load data into Hive tables. The user can create an external table that points to a specified location within HDFS. In this particular usage, the user can copy a file into the specified location using the HDFS put or copy commands and create a table pointing to this location with all the relevant row format information. Once this is done, the user can transform the data and insert them into

any other Hive table. For example, if the file $/tmp/pv_2008-06-08.txt$ contains comma separated page views served on 2008-06-08, and this needs to be loaded into the *page_view* table in the appropriate partition, the following sequence of commands can achieve this:

```
CREATE EXTERNAL TABLE page_view_stg(viewTime INT, userid BIGINT,
                page_url STRING, referrer_url STRING,
                ip STRING COMMENT 'IP Address of the User',
                country STRING COMMENT 'country of origination')
COMMENT 'This is the staging page view table'
ROW FORMAT DELIMITED FIELDS TERMINATED BY '44' LINES TERMINATED BY '12'
STORED AS TEXTFILE
LOCATION '/user/data/staging/page_view';
hadoop dfs -put /tmp/pv_2008-06-08.txt /user/data/staging/page_view
FROM page_view_stg pvs
INSERT OVERWRITE TABLE page_view PARTITION(dt='2008-06-08', country='US')
SELECT pvs.viewTime, pvs.userid, pvs.page_url, pvs.referrer_url,
null, null, pvs.ip
WHERE pvs.country = 'US';
```

In the example above, nulls are inserted for the array and map types in the destination tables but potentially these can also come from the external table if the proper row formats are specified.

This method is useful if there is already legacy data in HDFS on which the user wants to put some metadata so that the data can be queried and manipulated using Hive.

Additionally, the system also supports syntax that can load the data from a file in the local files system directly into a Hive table where the input data format is the same as the table format. If $/tmp/pv_2008-06-08_us.txt$ already contains the data for US, then we do not need any additional filtering as shown in the previous example. The load in this case can be done using the following syntax:

```
LOAD DATA LOCAL INPATH /tmp/pv_2008-06-08_us.txt INTO
TABLE page_view PARTITION(date='2008-06-08', country='US')
```

The path argument can take a directory (in which case all the files in the directory are loaded), a single file name, or a wildcard (in which case all the matching files are uploaded). If the argument is a directory, it cannot contain subdirectories. Similarly, the wildcard must match file names only.

In the case that the input file $/tmp/pv_2008-06-08_us.txt$ is very large, the user may decide to do a parallel load of the data (using tools that

are external to Hive). Once the file is in HDFS - the following syntax can be used to load the data into a Hive table:

```
LOAD DATA INPATH '/user/data/pv_2008-06-08_us.txt' INTO
TABLE page_view PARTITION(date='2008-06-08', country='US')
```

It is assumed that the array and map fields in the input.txt files are null fields for these examples.

9.7.6 Querying and Inserting Data

Simple Query

For all the active users, one can use the query of the following form:

```
INSERT OVERWRITE TABLE user_active
SELECT user.*
FROM user
WHERE user.active = 1;
```

Note that unlike SQL, we always insert the results into a table. We will illustrate later how the user can inspect these results and even dump them to a local file. You can also run the following query in Beeline or the Hive CLI:

```
SELECT user.*
FROM user
WHERE user.active = 1;
```

This will be internally rewritten to some temporary file and displayed to the Hive client side.

Partition Based Query

What partitions to use in a query is determined automatically by the system on the basis of where clause conditions on partition columns. For example, in order to get all the *page_views* in the month of *03/2008* referred from domain *xyz.com*, one could write the following query:

```
INSERT OVERWRITE TABLE xyz_com_page_views
SELECT page_views.*
FROM page_views
WHERE page_views.date >= '2008-03-01' AND page_views.date <= '2008-03-31'
AND
```

```
    page_views.referrer_url like '%xyz.com';
```

Note that *page_views.date* is used here because the table (above) was defined with $PARTITIONEDBY(dateDATETIME, countrySTRING)$; if you name your partition something different, don't expect *.date* to do what you think!

Joins

In order to get a demographic breakdown (by gender) of *page_view* of 2008-03-03 one would need to join the *page_view* table and the user table on the userid column. This can be accomplished with a join as shown in the following query:

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM user u JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2008-03-03';
```

In order to do outer joins the user can qualify the join with LEFT OUTER, RIGHT OUTER or FULL OUTER keywords in order to indicate the kind of outer join (left preserved, right preserved or both sides preserved). For example, in order to do a full outer join in the query above, the corresponding syntax would look like the following query:

```
INSERT OVERWRITE TABLE pv_users
SELECT pv.*, u.gender, u.age
FROM user u FULL OUTER JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2008-03-03';
```

In order check the existence of a key in another table, the user can use LEFT SEMI JOIN as illustrated by the following example.

```
INSERT OVERWRITE TABLE pv_users
SELECT u.*
FROM user u LEFT SEMI JOIN page_view pv ON (pv.userid = u.id)
WHERE pv.date = '2008-03-03';
```

In order to join more than one tables, the user can use the following syntax:

```
INSERT OVERWRITE TABLE pv_friends
SELECT pv.*, u.gender, u.age, f.friends
FROM page_view pv JOIN user u ON (pv.userid = u.id)
JOIN friend_list f ON (u.id = f.uid)
WHERE pv.date = '2008-03-03';
```

Aggregations

In order to count the number of distinct users by gender one could write the following query:

```
INSERT OVERWRITE TABLE pv_gender_sum
SELECT pv_users.gender, count (DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

Multiple aggregations can be done at the same time, however, no two aggregations can have different DISTINCT columns .e.g while the following is possible

```
INSERT OVERWRITE TABLE pv_gender_agg
SELECT pv_users.gender, count(DISTINCT pv_users.userid),
count(*), sum(DISTINCT pv_users.userid)
FROM pv_users
GROUP BY pv_users.gender;
```

however, the following query is not allowed

```
INSERT OVERWRITE TABLE pv_gender_agg
SELECT pv_users.gender, count(DISTINCT pv_users.userid),
count(DISTINCT pv_users.ip)
FROM pv_users
GROUP BY pv_users.gender;
```

Multi Table/File Inserts

The output of the aggregations or simple selects can be further sent into multiple tables or even to hadoop dfs files (which can then be manipulated using hdfs utilities). For example, if along with the gender breakdown, one needed to find the breakdown of unique page views by age, one could accomplish that with the following query:

```
FROM pv_users
INSERT OVERWRITE TABLE pv_gender_sum
    SELECT pv_users.gender, count_distinct(pv_users.userid)
    GROUP BY pv_users.gender
INSERT OVERWRITE DIRECTORY '/user/data/tmp/pv_age_sum'
    SELECT pv_users.age, count_distinct(pv_users.userid)
    GROUP BY pv_users.age;
```

The first insert clause sends the results of the first group by to a Hive table while the second one sends the results to a hadoop dfs files.

Inserting into Local Files

In certain situations you would want to write the output into a local file so that you could load it into an excel spreadsheet. This can be accomplished with the following command:

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'
SELECT pv_gender_sum.*
FROM pv_gender_sum;
```

Notes

[24] http://dl.acm.org/citation.cfm?doid=368996.369025

9.8 Questions

9.8.1 Short answer questions

1. What is Apache Hive?
2. What is Derby?
3. What is InitSchema?
4. What is metastore?
5. What is hql?

9.8.2 Essay questions

1. Explain installation and configuration of Apache Hive.
2. What are the various ways to run Hive? Explain in detail.
3. How do you use Hive with Hadoop mapreduce? Explan.
4. How do you use Hive in local mode? Explain.
5. What are the various data types in Hive? Write in detail with examples.
6. Write about various built in operators and functions.
7. Write about the following using examples.
   (a) DDL commands
   (b) DML commands
   (c) SQL commands

Chapter 10

Spark

10.1 Introduction

Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since.

10.1.1 Overview

Apache Spark has its architectural foundation in the resilient distributed dataset (RDD), a read-only multiset of data items distributed over a cluster of machines, that is maintained in a fault-tolerant way.[25] The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API. [26]

Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a partic-

ular linear dataflow structure on distributed programs: MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory. [27]

Inside Apache Spark the workflow is managed as a directed acyclic graph (DAG). Nodes represent RDDs while edges represent the operations on the RDDs. Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications may be reduced by several orders of magnitude compared to Apache Hadoop MapReduce implementation. Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.[28]

Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster, where you can launch a cluster either manually or use the launch scripts provided by the install package. It is also possible to run these daemons on a single machine for testing), Hadoop YARN, Apache Mesos or Kubernetes. For distributed storage, Spark can interface with a wide variety, including Alluxio, Hadoop Distributed File System (HDFS), MapR File System (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.[29]

10.1.2 Spark shell - basics

Execute the following command to know your present working directory in *Spark Shell*.

```
System.getProperty("user.dir")
```

The above command prints "`String = /home/hadoop`" It is home directory in your OS. [1] There are two other ways to perform the same action.

- `new java.io.File(".").getAbsolutePath`
- `import scala.reflect.io.File; File(".").toAbsolute`

Following is the procedure to list directory

```
scala> import java.nio.file.{FileSystems, Files}

scala> import scala.collection.JavaConverters._

scala> val dir = FileSystems.getDefault.getPath("/tmp/baeldung")
val dir: java.nio.file.Path = /tmp/baeldung

scala> Files.list(dir).iterator().asScala.foreach(println)
/tmp/mk/another_dir
/tmp/mk/b.json
/tmp/mk/a.txt
```

Following statement also works.

```
scala> Files.walk(dir).iterator().asScala.foreach(println)
```

Following code reads data and prints in Scala REPL. [30]

```
import scala.io.Source._
val lines = fromFile("file.txt").getLines
```

or with a single statement

```
val lines = scala.io.Source.fromFile("file.txt", "utf-8"). \
     > getLines.mkString
```

10.2 Resilient Distributed Dataset (RDD)

At a high level, every Spark application consists of a driver program that runs the user's main function and executes various parallel operations on a cluster. The main abstraction Spark provides is a re-

[1] `/home/hadoop` is the home directory in my OS.

silient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel. RDDs are created by starting with a file in the Hadoop file system (or any other Hadoop-supported file system), or an existing Scala collection in the driver program, and transforming it. Users may also ask Spark to persist an RDD in memory, allowing it to be reused efficiently across parallel operations. Finally, RDDs automatically recover from node failures.

A second abstraction in Spark is shared variables that can be used in parallel operations. By default, when Spark runs a function in parallel as a set of tasks on different nodes, it ships a copy of each variable used in the function to each task. Sometimes, a variable needs to be shared across tasks, or between tasks and the driver program. Spark supports two types of shared variables: broadcast variables, which can be used to cache a value in memory on all nodes, and accumulators, which are variables that are only "added" to, such as counters and sums.

10.2.1 Parallelized Collections

Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. There are two ways to create RDDs: parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

Parallelized collections are created by calling SparkContext's parallelize method on an existing collection in your driver program (a Scala Seq). The elements of the collection are copied to form a distributed dataset that can be operated on in parallel. For example, here is how to create a parallelized collection holding the numbers 1 to 5:

```
val data = Array(1, 2, 3, 4, 5)
val distData = sc.parallelize(data)
```

10.2.2 External Datasets

Spark can create distributed datasets from any storage source supported by Hadoop, including your local file system, HDFS, Cassandra, HBase, Amazon S3, etc. Spark supports *text* files, *SequenceFiles,* and any other Hadoop *InputFormat.*

Text file RDDs can be created using SparkContext's textFile method. This method takes a URI for the file (either a local path on the machine, or a `hdfs://,s3a://`, etc URI) and reads it as a collection of lines. Here is an example invocation:

```
scala> val distFile = sc.textFile("data.txt")
distFile: org.apache.spark.rdd.RDD[String] = data.txt
     MapPartitionsRDD[10] at textFile at <console>:26
```

10.2.3 RDD Operations

31

RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset. For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset).

All transformations in Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base dataset (e.g. a file). The transformations are only computed when an action requires a result to be returned to the driver program. This design enables Spark to run more efficiently. For example, we can realize that a dataset created through map will be used in a reduce and return only the result of the reduce to the driver, rather than the larger mapped dataset.

By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory

using the persist (or cache) method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

To illustrate RDD basics, consider the simple program below:

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength = lineLengths.reduce((a, b) => a + b)
```

The first line defines a base RDD from an external file. This dataset is not loaded in memory or otherwise acted on: lines is merely a pointer to the file. The second line defines lineLengths as the result of a map transformation. Again, lineLengths is not immediately computed, due to laziness. Finally, we run reduce, which is an action. At this point Spark breaks the computation into tasks to run on separate machines, and each machine runs both its part of the map and a local reduction, returning only its answer to the driver program.

10.3 Datasets & Dataframes

A DataFrame is a distributed collection of data, which is organized into named columns. Conceptually, it is equivalent to relational tables with good optimization techniques. A DataFrame can be constructed from an array of different sources such as Hive tables, Structured Data files, external databases, or existing RDDs. This API was designed for modern Big Data and data science applications taking inspiration from DataFrame in R Programming and Pandas in Python.

10.3.1 Features of DataFrame

Here is a set of few characteristic features of DataFrame

- Ability to process the data in the size of Kilobytes to Petabytes on a single node cluster to large cluster.
- Supports different data formats (Avro, csv, elastic search, and Cassandra) and storage systems (HDFS, HIVE tables, mysql, etc).

- State of art optimization and code generation through the Spark SQL Catalyst optimizer (tree transformation framework).
- Can be easily integrated with all Big Data tools and frameworks via Spark-Core. Provides API for Python, Java, Scala, and R Programming.

10.3.2 SQLContext

SQLContext is a class and is used for initializing the functionalities of Spark SQL. *SparkContext* class object (sc) is required for initializing SQLContext class object. The following command is used for initializing the SparkContext through spark-shell.

```
\$ spark-shell
```

By default, the SparkContext object is initialized with the name sc when the spark-shell starts. Use the following command to create SQLContext.

```
scala> val sqlcontext = new org.apache.spark.sql.SQLContext(sc)
```

10.3.3 Example

Let us consider an example of employee records in a JSON file named `employee.json`. Use the following commands to create a DataFrame (df) and read a JSON document named `employee.json` with the following content. `employee.json` – Place this file in the directory where the current scala> pointer is located.

```
{
   {"id" : "1201", "name" : "satish", "age" : "25"}
   {"id" : "1202", "name" : "krishna", "age" : "28"}
   {"id" : "1203", "name" : "amith", "age" : "39"}
   {"id" : "1204", "name" : "javed", "age" : "23"}
   {"id" : "1205", "name" : "prudvi", "age" : "23"}
}
```

10.3.4 DataFrame Operations

DataFrame provides a domain-specific language for structured data manipulation. Here, we include some basic examples of structured data processing using DataFrames. Follow the steps given below to perform DataFrame operations

1. Read the JSON Document

   First, we have to read the JSON document. Based on this, generate a DataFrame named (dfs). Use the following command to read the JSON document named employee.json. The data is shown as a table with the fields − id, name, and age.

   ```
   scala> val dfs = sqlContext.read.json("employee.json")
   ```

2. Show the Data

   If you want to see the data in the DataFrame, then use the following command.

   ```
   scala> dfs.show()
   ```

   Output − You can see the employee data in a tabular format.

   ```
   <console>:22, took 0.052610 s
   +----+------+--------+
   |age | id   |  name  |
   +----+------+--------+
   | 25 | 1201 | satish |
   | 28 | 1202 | krishna|
   | 39 | 1203 | amith  |
   | 23 | 1204 | javed  |
   | 23 | 1205 | prudvi |
   +----+------+--------+
   ```

3. Use printSchema Method

   If you want to see the Structure (Schema) of the DataFrame, then use the following command.

   ```
   scala> dfs.printSchema()
   ```

   Output

```
root
   |-- age: string (nullable = true)
   |-- id: string (nullable = true)
   |-- name: string (nullable = true)
```

4. Use Select Method

   Use the following command to fetch name-column among three columns from the DataFrame.

   ```
   scala> dfs.select("name").show()
   ```

   Output − You can see the values of the name column.

   ```
   <console>:22, took 0.044023 s
   +--------+
   |  name  |
   +--------+
   | satish |
   | krishna|
   | amith  |
   | javed  |
   | prudvi |
   +--------+
   ```

5. Use Age Filter

   Use the following command for finding the employees whose age is greater than 23 (age $>$ 23).

   ```
   scala> dfs.filter(dfs("age") > 23).show()
   ```

   Output

   ```
   <console>:22, took 0.078670 s
   +----+------+--------+
   |age | id   | name   |
   +----+------+--------+
   | 25 | 1201 | satish |
   | 28 | 1202 | krishna|
   | 39 | 1203 | amith  |
   +----+------+--------+
   ```

6. Use groupBy Method

   Use the following command for counting the number of employees who are of the same age.

   ```
   scala> dfs.groupBy("age").count().show()
   ```

   Output − two employees are having age 23.

   ```
   <console>:22, took 5.196091 s
   +----+-----+
   |age |count|
   +----+-----+
   | 23 |  2  |
   | 25 |  1  |
   | 28 |  1  |
   | 39 |  1  |
   +----+-----+
   ```

10.4 Spark SQL

Spark SQL is one of the most used Spark modules which is used for processing structured columnar data format. Once you have a DataFrame created, you can interact with the data by using SQL syntax. In other words, Spark SQL brings native RAW SQL queries on Spark meaning you can run traditional ANSI SQL on Spark Dataframe, in the SQL tutorial, you will learn in detail using SQL select, where, group by, join, union etc. For instance, simple calculations can be done as

```
spark.sql("select 1+1").show()
```

The above code produce a result, i.e., 2, and displays in sparks usual way of output. In order to use SQL, first, create a temporary table on DataFrame using the `createOrReplaceTempView()` function. Once created, this table can be accessed throughout the *SparkSession* using *sql()* and it will be dropped along with your *SparkContex*t termination. Use `sql()` method of the SparkSession object to run the query and this method returns a new DataFrame.

10.4.1 Create SQL View

Create a DataFrame from a CSV file. You can find this CSV file at Github project.

```
// Read CSV file into table
val df = spark.read.option("header",true)
          .csv("/Users/admin/simple-zipcodes.csv")
df.printSchema()
df.show()
```

To use ANSI SQL query similar to RDBMS, you need to create a temporary table by reading the data from a CSV file. You can find this CSV file at Github project.

```
// Read CSV file into table
spark.read.option("header",true)
          .csv("/Users/admin/simple-zipcodes.csv")
          .createOrReplaceTempView("Zipcodes")
```

10.4.2 Spark SQL to Select Columns

The `select()` function of DataFrame API is used to select the specific columns from the DataFrame.

```
// DataFrame API Select query
df.select("country","city","zipcode","state")
     .show(5)
```

In SQL, you can achieve the same using `SELECT FROM` clause as shown below.

```
// SQL Select query
spark.sql("SELECT country, city, zipcode, state FROM ZIPCODES")
     .show(5)
```

10.4.3 Filter Rows

To filter the rows from the data, you can use `where()` function from the DataFrame API.

```
// DataFrame API where()
```

```
df.select("country","city","zipcode","state")
  .where("state == 'AZ'")
  .show(5)
```

Similarly, in SQL you can use `WHERE` clause as follows.

```
// SQL where
spark.sql(""" SELECT  country, city, zipcode, state FROM ZIPCODES
          WHERE state = 'AZ' """)
     .show(5)
```

10.4.4 Sorting

To sort rows on a specific column use `orderBy()` function on DataFrame API.

```
// sorting
df.select("country","city","zipcode","state")
  .where("state in ('PR','AZ','FL')")
  .orderBy("state")
  .show(10)
```

In SQL, you can achieve sorting by using ORDER BY clause.

```
// SQL ORDER BY
spark.sql(""" SELECT  country, city, zipcode, state FROM ZIPCODES
          WHERE state in ('PR','AZ','FL') order by state """)
     .show(10)
```

10.4.5 Grouping

The `groupBy().count()` is used to perform the group by on DataFrame.

```
// grouping
df.groupBy("state").count()
  .show()
```

You can achieve group by in Spark SQL by using GROUP BY clause.

```
// SQL GROUP BY clause
spark.sql(""" SELECT state, count(*) as count FROM ZIPCODES
```

```
          GROUP BY state""")
     .show()
```

10.4.6 SQL Join Operations

Similarly, if you have two tables, you can perform the Join operations in Spark. Here is an example

```
// Join DataFrames
empDF.join(deptDF,empDF("emp_dept_id") ===  deptDF("dept_id"),"inner")
    .show(false)
```

In Spark SQL you can do it as below

```
// SQL Join
spark.sql("select * from EMP e, DEPT d where e.emp_dept_id == d.dept_id")
     .show(false)
```

10.4.7 Union

`union()` method of the DataFrame is used to combine two DataFrame's of the same structure/schema. If schemas are not the same it returns an error. First, let's create two DataFrame with the same schema.

```
import spark.implicits._

  val simpleData = Seq(("James","Sales","NY",90000,34,10000),
    ("Michael","Sales","NY",86000,56,20000),
    ("Robert","Sales","CA",81000,30,23000),
    ("Maria","Finance","CA",90000,24,23000)
  )
  val df = simpleData.toDF("employee_name","department","state","salary",
  df.printSchema()
  df.show()
df.printSchema prints the schema and df.show() display DataFrame to consc
```

Output:

```
root
 |-- employee_name: string (nullable = true)
 |-- department: string (nullable = true)
```

```
 |-- state: string (nullable = true)
 |-- salary: integer (nullable = false)
 |-- age: integer (nullable = false)
 |-- bonus: integer (nullable = false)

+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|        James|     Sales|   NY| 90000| 34|10000|
|      Michael|     Sales|   NY| 86000| 56|20000|
|       Robert|     Sales|   CA| 81000| 30|23000|
|        Maria|   Finance|   CA| 90000| 24|23000|
+-------------+----------+-----+------+---+-----+
```

Now, let's create a second Dataframe with the new records and some records from the above Dataframe but with the same schema.

```
val simpleData2 = Seq(("James","Sales","NY",90000,34,10000),
    ("Maria","Finance","CA",90000,24,23000),
    ("Jen","Finance","NY",79000,53,15000),
    ("Jeff","Marketing","CA",80000,25,18000),
    ("Kumar","Marketing","NY",91000,50,21000)
  )
  val df2 = simpleData2.toDF("employee_name","department","state","s
```

This yields below output

```
// Output:
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|Jen          |Finance   |NY   |79000 |53 |15000|
|Jeff         |Marketing |CA   |80000 |25 |18000|
|Kumar        |Marketing |NY   |91000 |50 |21000|
+-------------+----------+-----+------+---+-----+
```

Combine two or more DataFrames using union

DataFrame `union()` method combines two DataFrames and returns the new DataFrame with all rows from two Dataframes regardless of duplicate data.

```
// Combine two or more DataFrames using union
  val df3 = df.union(df2)
  df3.show(false)
```

As you see below it returns all records.

```
// Output:
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Michael      |Sales     |NY   |86000 |56 |20000|
|Robert       |Sales     |CA   |81000 |30 |23000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|James        |Sales     |NY   |90000 |34 |10000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|Jen          |Finance   |NY   |79000 |53 |15000|
|Jeff         |Marketing |CA   |80000 |25 |18000|
|Kumar        |Marketing |NY   |91000 |50 |21000|
+-------------+----------+-----+------+---+-----+
```

Combine DataFrames using unionAll

DataFrame `unionAll()` method is deprecated since Spark "2.0.0" version and recommends using the `union()` method.

```
// Combine DataFrames using unionAll
  val df4 = df.unionAll(df2)
  df4.show(false)
```

Returns the same output as above.

Combine without Duplicates

Since the `union()` method returns all rows without distinct records, we will use the `distinct()` function to return just one record when duplicate exists.

```
// Combine without Duplicates
  val df5 = df.union(df2).distinct()
  df5.show(false)
```

Yields below output. As you see, this returns only distinct rows.

```
// Output:
+-------------+----------+-----+------+---+-----+
|employee_name|department|state|salary|age|bonus|
+-------------+----------+-----+------+---+-----+
|James        |Sales     |NY   |90000 |34 |10000|
|Maria        |Finance   |CA   |90000 |24 |23000|
|Jeff         |Marketing |CA   |80000 |25 |18000|
|Jen          |Finance   |NY   |79000 |53 |15000|
|Kumar        |Marketing |NY   |91000 |50 |21000|
|Michael      |Sales     |NY   |86000 |56 |20000|
|Robert       |Sales     |CA   |81000 |30 |23000|
+-------------+----------+-----+------+---+-----+
```

`unionAll()` is deprecated since Spark "2.0.0" version and replaced with `union()`.

Notes

[25]Zaharia, Matei; Chowdhury, Mosharaf; Franklin, Michael J.; Shenker, Scott; Stoica, Ion. Spark: Cluster Computing with Working Sets (PDF). USENIX Workshop on Hot Topics in Cloud Computing (HotCloud).

[26]Chambers, Bill (2017-08-10). "12". Spark: The Definitive Guide. O'Reilly Media. virtually all Spark code you run, where DataFrames or Datasets, compiles down to an RDD

[27]Zaharia, Matei; Chowdhury, Mosharaf; Das, Tathagata; Dave, Ankur; Ma, Justin; McCauley, Murphy; J., Michael; Shenker, Scott; Stoica, Ion (2010).

[28]Harris, Derrick (28 June 2014). "4 reasons why Spark could jolt Hadoop into hyperdrive". Gigaom. Archived from the original on 24 October 2017. Retrieved 25 February 2016.

[29]Wang, Yandong; Goldstone, Robin; Yu, Weikuan; Wang, Teng (May 2014). "Characterization and Optimization of Memory-Resident MapReduce on HPC Systems". 2014 IEEE 28th International Parallel and Distributed Processing Symposium. IEEE. pp. 799–808.

[30]In scala it is always possible to do a single task in multiple ways. Read about various ways to read the content from a given file at https://stackoverflow.com/questions/1284423/read-entire-file-in-scala.

[31]For more information refer to material available at https://spark.apache.org/docs/latest/rdd-programming-guide.html.

10.5 Questions

10.5.1 Short answer questions

1. What is Apache Spark?
2. What is RDD?
3. What is Spark dataframe?
4. What is Spark sql?
5. What is Spark context?
6. What is Scala?
7. What is pyspark?

10.5.2 Essay questions

1. Write about Sparks shell basics.
2. Explain Spark's RDD using code.
3. Explain Spark's dataframes using code.
4. Explain Spark's sql commands using code.