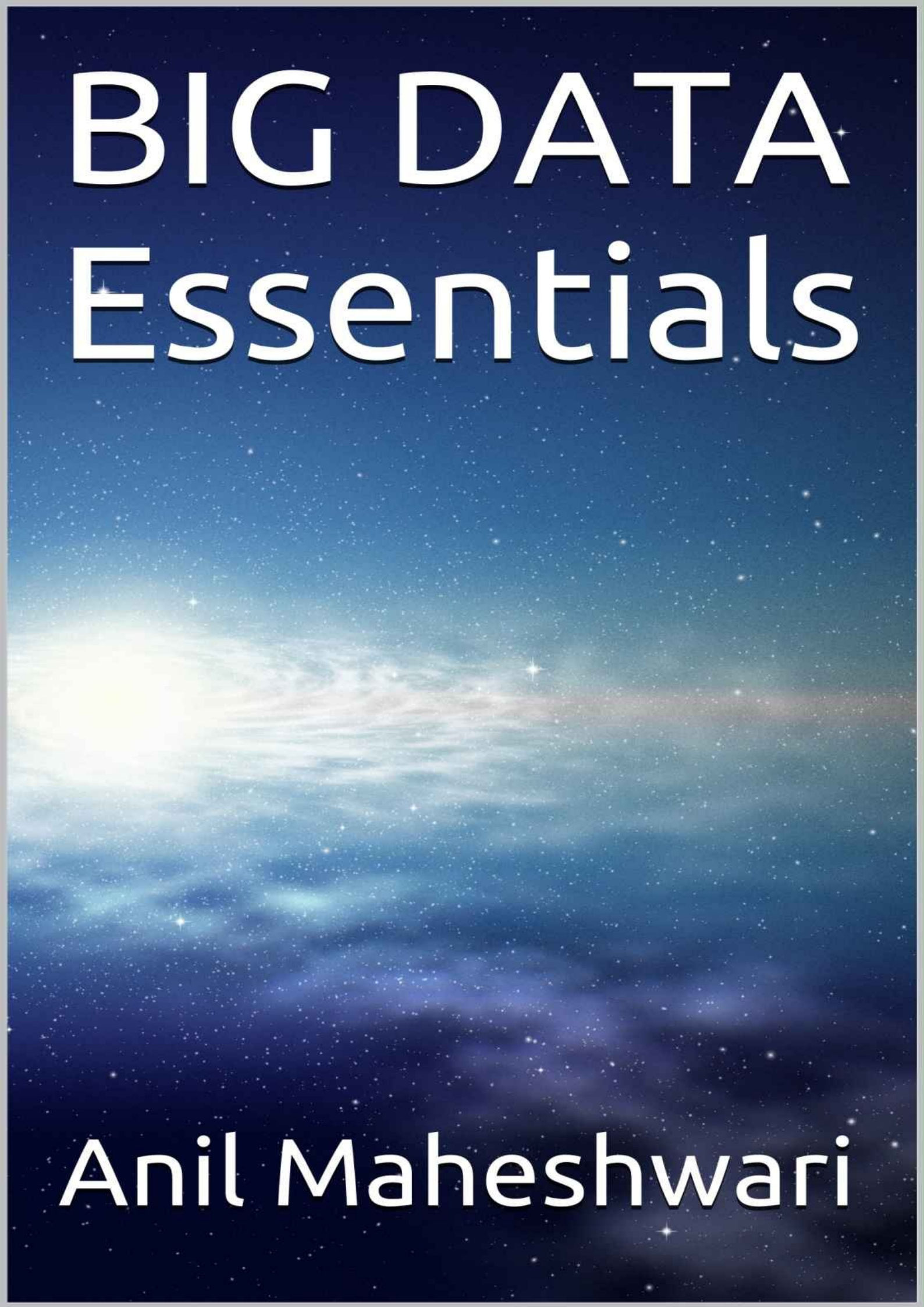

BIG DATA
Essentials
Anil Maheshwari

Big Data Essentials

Other books by the same author:

Data Analytics Made Accessible the #1 Bestseller in Data Mining

Moksha: Liberation Through Transcendence

Preface

Big Data is a new, and inclusive, natural phenomenon. It is as messy as nature itself. It requires a new kind of Consciousness to fathom its scale and scope, and its many opportunities and challenges. Understanding the essentials of Big Data requires suspending many conventional expectations and assumptions about data … such as completeness, clarity, consistency, and conciseness. Fathoming and taming the multi-layered Big Data is a dream that is slowly becoming a reality. It is a rapidly evolving field that is growing exponentially in value and capabilities.

There is a growing number of books being written on Big Data. They fall mostly in two categories. The first kind focus on business aspects, and discuss the strategic internal shifts required for reaping the business benefits from the many opportunities offered by Big Data. The second kind focus on particular technology platforms, such as Hadoop or Spark. This book aims to bring together the business context *and* the technologies in a seamless way.

This book was written to meet the needs for an introductory Big Data course. It is meant for students, as well as executives, who wish to take advantage of emerging opportunities in Big Data. It provides an intuition of the wholeness of the field in a simple language, free from jargon and code. All the essential Big Data technology tools and platforms such as Hadoop, MapReduce, Spark, and NoSql are discussed. Most of the relevant programming details have been moved to Appendices to ensure readability. The short chapters make it easy to quickly understand the key concepts. A complete case study of developing a Big Data application is included.

Thanks to Maharishi Mahesh Yogi for creating a wonderful university whose consciousness-based environment made writing this evolutionary book possible. Thanks to many current and former students for contributing to this book. Dheeraj Pandey assisted with the Weblog analyzer application and its details. Suraj Thapalia assisted with the Hadoop installation guide. Enkhbileg Tseeleesuren helped write the Spark tutorial. Thanks to my family for supporting me in this process. My daughters Ankita and Nupur reviewed the book and made helpful comments. My father Mr. RL Maheshwari and brother Dr. Sunil Maheshwari also read the book and enthusiastically approved it. My colleague Dr. Edi Shivaji too reviewed the book.

May the Big Data Force be with you!

Dr. Anil Maheshwari

August 2016, Fairfield, IA

Contents

Chapter 1 – Wholeness of Big Data

Introduction

Big Data is an all-inclusive term that refers to extremely large, very fast, diverse, and complex data that cannot be managed with traditional data management tools. Ideally, Big Data would harness all kinds of data, and deliver the right information, to the right person, in the right quantity, at the right time, to help make the right decision. Big Data can be managed by developing infinitely scalable, totally flexible, and evolutionary data architectures, coupled with the use of extremely cost-effective computing components. The infinite potential knowledge embedded within this cosmic computer would help connect everything to the Unified Field of all the laws of nature.

This book will provide a complete overview of Big Data for the executive and the data specialist. This chapter will cover the key challenges and benefits of Big Data, and the essential tools and technologies now available for organizing and manipulating Big Data.

Understanding Big Data

Big Data can be examined on two levels. On a fundamental level, it is data that can be analyzed and utilized for the benefit of the business. On another level, it is a special kind of data that poses unique challenges. This is the level that this book will focus on.

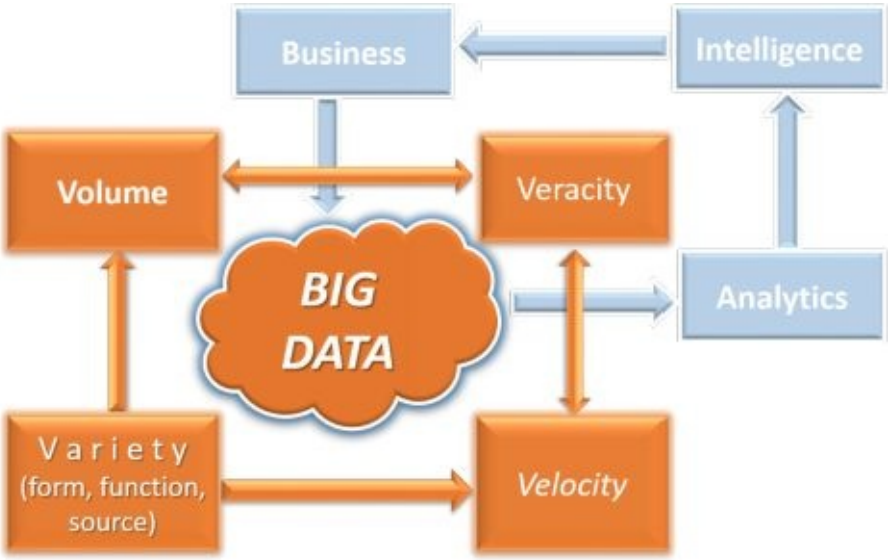


Figure 1-1: Big Data Context

At the level of business, data generated by business operations, can be analyzed to generate insights that can help the business make better decisions. This makes the business grow bigger, and generate even more data, and the cycle continues. This is represented by the blue cycle on the top-right of Figure 1.1. This aspect is discussed in Chapter 10, a primer on Data Analytics.

On another level, Big Data is different from traditional data in every way: space, time, and function. The quantity of Big Data is 1,000 times more than that of traditional data. The speed of data generation and transmission is 1,000 times faster. The forms and functions of Big Data are much more diverse: from numbers to text, pictures, audio, videos, activity logs, machine data, and more. There are also many more sources of data, from individuals to organizations to governments, using a range of devices from mobile phones to computers to industrial machines. Not all data will be of equal quality and value. This is represented by the red cycle on the bottom left of Figure 1.1. This aspect of Big Data, and its new technologies, is the main focus of this book.

Big Data is mostly unstructured data. Every type of data is structured differently, and will have to be dealt with differently. There are huge opportunities for technology providers to innovate and manage the entire life cycle of Big Data … to generate, gather, store, organize, analyze, and visualize this data.

CASELET: IBM Watson: A Big Data system

IBM created the Watson system as a way of pushing the boundaries of Artificial Intelligence and natural language understanding technologies. Watson beat the world champion human players of Jeopardy (quiz style TV show) in Feb 2011. Watson reads up on data about everything on the web including the entire Wikipedia. It digests and absorbs the data based on simple generic rules such as: books have authors; stories have heroes; and drugs treat ailments. A jeopardy clue, received in the form of a cryptic phrase, is broken down into many possible potential sub-clues of the correct answer. Each sub-clue is examined to see the likeliness of its answer being the correct answer for the main problem. Watson calculates the confidence level of each possible answer. If the confidence level reaches more than a threshold level, it decides to offer the answer to the clue. It manages to do all this in a mere 3 seconds.

Watson is now being applied to diagnosing diseases, especially cancer. Watson can read all the new research published in the medical journals to update its knowledge base. It is being used to diagnose the probability of various diseases, by applying factors such as patient's current symptoms, health history, genetic history, medication records, and other factors to recommend a particular diagnosis. (Source: Smartest machines on Earth: youtube.com/watch?v=TCOhyaw5bwg)

Figure 1.2: IBM Watson playing Jeopardy

Q1: What kinds of Big Data knowledge, technologies and skills are required to build a system like Watson? What kind of resources are needed?

Q2: Will doctors be able to compete with Watson in diagnosing diseases

and prescribing medications? Who else could benefit from a system like Watson?

Capturing Big Data

If data were simply growing too large, OR only moving too fast, OR only becoming too diverse, it would be relatively easy. However, when the four Vs (Volume, Velocity, Variety, and Veracity) arrive together in an interactive manner, it creates a perfect storm. While the Volume and Velocity of data drive the major technological concerns and the costs of managing Big Data, these two Vs are themselves being driven by the 3rd V, the Variety of forms and functions and sources of data.

Volume of Data

The quantity of data has been relentlessly doubling every 12-18 months. Traditional data is measured in Gigabytes (GB) and Terabytes (TB), but Big Data is measured in Petabytes (PB) and Exabytes (1 Exabyte = 1 Million TB).

This data is so huge that it is almost a miracle that one can find any specific thing in it, in a reasonable period of time. Searching the world-wide web was the first true Big Data application. Google perfected the art of this application, and developed many of the path-breaking technologies we see today to manage Big Data.

The primary reason for the growth of data is the dramatic reduction in the cost of storing data. The costs of storing data have decreased by 30-40% every year. Therefore, there is an incentive to record everything that can be observed. It is called 'datafication' of the world. The costs of computation and communication have also been coming down, similarly. Another reason for the growth of data is the increase in the number of forms and functions of data. More about this in the Variety section.

Velocity of Data

If traditional data is like a lake, Big Data is like a fast-flowing river. Big Data is being generated by billions of devices, and communicated at the speed of the internet. Ingesting all this data is like drinking from a fire hose. One does not have control over how fast the data will come. A huge unpredictable data-stream is the new metaphor for thinking about Big Data.

The primary reason for the increased velocity of data is the increase in internet speed. Internet speeds available to homes and offices are now increasing from 10MB/sec to 1 GB/sec (100 times faster). More people are getting access to high-speed internet around the world. Another important reason is the increased variety of sources that can generate and communicate data from anywhere, at any time. More on that in the Variety section.

Variety of Data

Big data is inclusive of all forms of data, for all kinds of functions, from all sources and devices. If traditional data, such as invoices and ledgers were like a small store, Big Data is the biggest imaginable shopping mall that offers unlimited variety. There are three major kinds of variety.

1. The first aspect of variety is the **form of data**. Data types range in order of simplicity and size from numbers to text, graph, map, audio, video, and others. There could be a composite of data that includes many elements in a single file. For example, text documents have text and graphs and pictures embedded in them. Video can have charts and songs embedded in them. Audio and video have different and more complex storage formats than numbers and text. Numbers and text can be more easily analyzed than an audio or video file. How should composite entities be stored and analyzed?
2. The second aspect is the variety of **function of data**. There are human chats and conversation data, songs and movies for entertainment, business transaction records, machine operations performance data, new product design data, old data for backup, etc. Human communication data would be processed very differently from operational performance data, with totally different objectives. A variety of applications are needed to compare pictures in order to recognize people's faces; compare voices to identify the speaker; and compare handwritings to identify the writer.
3. The third aspect of variety is the **source of data**. Mobile phones and tablet devices enable a wide series of applications or apps to access data and generate data from anytime anywhere. Web access logs are another new and huge source of diagnostic data. ERP systems generate massive amounts of structured business transactional information. Sensors on machines, and RFID tags on assets, generate incessant and repetitive data. Broadly speaking, there are three broad types of sources of data: Human-human communications; human-machine communications; and machine-to-machine communications. The sources of data, and their respective applications arising from that data, will be discussed in the next chapter.

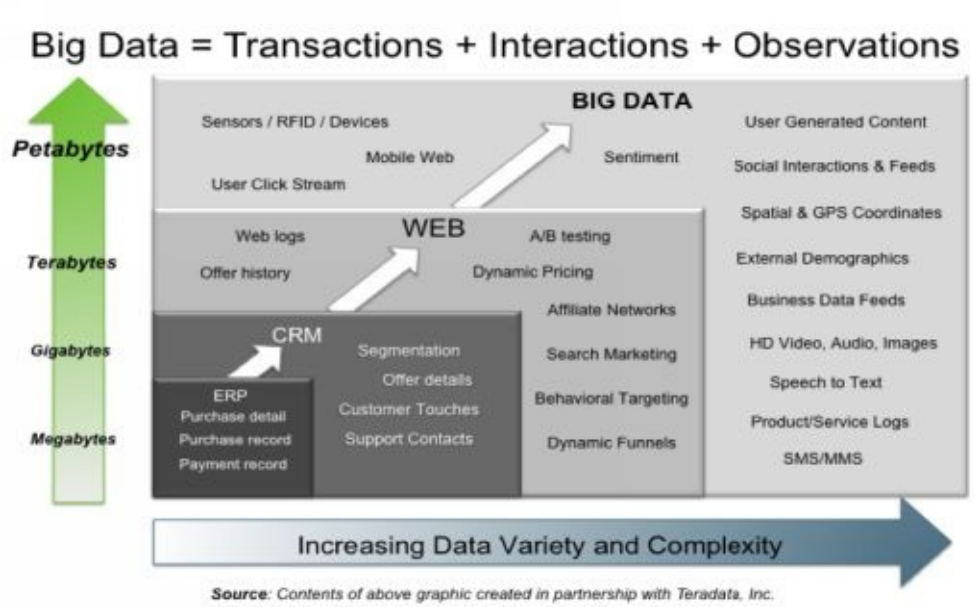


Figure 1.3 Sources of Big Data (Source: Hortonworks.com)

Veracity of Data

Veracity relates to the believability and quality of data. Big Data is messy. There is a lot of misinformation and disinformation. The reasons for poor quality of data can range from human and technical error, to malicious intent.

1. The source of information may not be authoritative. For example, all websites are not equally trustworthy. Any information from whitehouse.gov or from nytimes.com is more likely to be authentic and complete. Wikipedia is useful, but not all pages are equally reliable. The communicator may have an agenda or a point of view.
2. The data may not be received correctly because of human or technical failure. Sensors and machines for gathering and communicating data may malfunction and may record and transmit incorrect data. Urgency may require the transmission of the best data available at a point in time. Such data makes reconciliation with later, accurate, records more problematic.
3. The data provided and received, may however, also be intentionally wrong, for competitive or security reasons.

Data needs to be sifted and organized by quality factors, for it to be put to any great use.

Benefitting from Big Data

Data usually belongs to the organization that generates it. There is other data, such as social media data, that is freely accessible under an open general license. Organizations can use this data to learn about their consumers, improve their service delivery, and design new products to delight their customers and to gain a competitive advantage. Data is also like a new natural resource. It is being used to design new digital products, such as on-demand entertainment and learning.

Organizations may choose to gather and store this data for later analysis, or to sell it to other organizations, who might benefit from it. They may also legitimately choose to discard parts of their data for privacy or legal reasons. However, organizations cannot afford to ignore Big Data. Organizations that do not learn to engage with Big Data, could find themselves left far behind their competition, landing in the dustbin of history. Innovative small and new organizations can use Big Data to quickly scale up and beat larger and more mature organizations.

Big Data applications exist in all industries and aspects of life. There are three major types of Big Data applications: Monitoring and Tracking, Analysis and Insight, and new digital product development.

Monitoring and Tracking Applications: Consumer goods producers use monitoring and tracking applications to understand the sentiments and needs of their customers. Industrial organizations use Big Data to track inventory in massive interlinked global supply chains. Factory owners use it to monitor machine performance and do preventive maintenance. Utility companies use it to predict energy consumption, and manage demand and supply. Information Technology companies use it to track website performance and improve its usefulness. Financial organizations use it to project trends better and make more effective and profitable bets, etc.

Analysis and Insight: Political organizations use Big Data to micro-target voters and win elections. Police use Big Data to predict and prevent crime. Hospitals use it to better diagnose diseases and make medicine prescriptions. Ad agencies use it to design more targeted marketing campaigns quickly. Fashion designers use it to track trends and create more innovative products.

Figure1.4: The first Big Data President

New Product Development: Incoming data could be used to design new products such as reality TV entertainment. Stock market feeds could be a digital product. This area needs much more development.

Management of Big Data

Many organizations have started initiatives around the use of Big Data. However, most organizations do not necessarily have a grip on it. Here are some emerging insights into making better use of Big Data.

1. Across all industries, the business case for Big Data is strongly focused on addressing customer-centric objectives. The first focus on deploying Big Data initiatives is to protect and enhance customer relationships and customer experience.
2. Solve a real pain-point. Big Data should be deployed for specific business objectives in order to have management avoid being overwhelmed by the sheer size of it all.
3. Organizations are beginning their pilot implementations by using existing and newly accessible internal sources of data. It is better to begin with data under one's control and where one has a superior understanding of the data.
4. Put humans and data together to get the most insight. Combining data-based analysis with human intuition and perspectives is better than going just one way.
5. Advanced analytical capabilities are required, but lacking, for organizations to get the most value from Big Data. There is a growing awareness of building or hiring those skills and capabilities.
6. Use more diverse data, not just more data. This would provide a broader perspective into reality and better quality insights.
7. The faster you analyze the data, the more its predictive value. The value of data depreciates with time. If the data is not processed in five minutes, then the immediate advantage is lost.
8. Don't throw away data if no immediate use can be seen for it. Data has value beyond what you initially anticipate. Data can add perspective to other data later on in a multiplicative manner.
9. Maintain one copy of your data, not multiple. This would help avoid confusion and increase efficiency.
10. Plan for exponential growth. Data is expected to continue to grow at exponential rates. Storage costs continue to fall, data generation continues to grow, data-based applications continue to grow in capability and functionality.
11. A scalable and extensible information management foundation is a prerequisite for big data advancement. Big Data builds upon a resilient, secure, efficient, flexible, and real-time information processing environment.

12. Big Data is transforming business, just like IT did. Big Data is a new phase representing a digital world. Business and society are not immune to its strong impacts.

Organizing Big Data

Good organization depends upon the purpose of the organization.

Given huge quantities, it would be desirable to organize the data to speed up the search process for finding a specific, a desired thing in the entire data. The cost of storing and processing the data, too, would be a major driver for the choice of an organizing pattern.

Given the fast speed of data, it would be desirable to create a scalable number of ingest points. It will also be desirable to create at least a thin veneer of control over the data by maintaining count and averages over time, unique values received, etc.

Given the variety in form factors, data needs to be stored and analyzed differently. Videos need to be stored separately and used for serving in a streaming mode. Text data may be combined, cleaned, and visualized for themes and sentiments.

Given different quality levels of data, various data sources may need to be ranked and prioritized before serving them to the audience. For example, the quality of a webpage may be computed through a PageRank mechanism.

Analyzing Big Data

Big Data can be analyzed in two ways. These are called analyzing Big Data in motion or Big Data at rest. First way is to process the incoming stream of data in real time for quick and effective statistics about the data. The other way is to store and structure the data and apply standard analytical techniques on batches of data for generating insights. This could then be visualized using real-time dashboards. Big Data can be utilized to visualize a flowing or a static situation. The nature of processing this huge, diverse, and largely unstructured data, can be limited only by one's imagination.

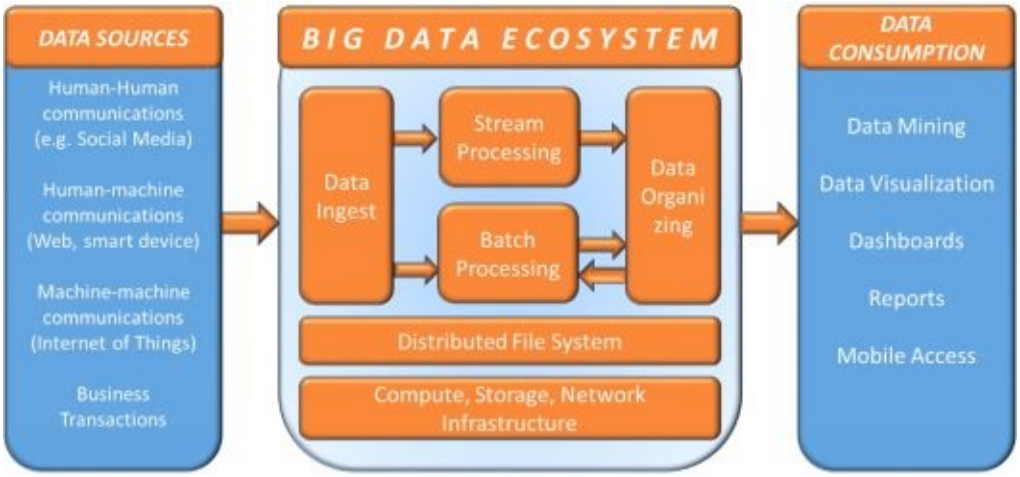


Figure 1.5: Big Data Architecture

A million points of data can be plotted in a graph and offer a view of the density of data. However, plotting a million points on the graph may produce a blurred image which may hide, rather than highlight the distinctions. In such a case, binning the data would help, or selecting the top few frequent categories may deliver greater insights. Streaming data can also be visualized by simple counts and averages over time. For example, below is a dynamically updated chart that shows up-to-date statistics of visitor traffic to my blogsite, anilmah.com. The bar shows the number of page views, and the inner darker bar shows the number of unique visitors. The dashboard could show the view by days, weeks or years also.

Figure 1.6: Real-time Dashboard for website performance for the author's blog

Text Data could be combined, filtered, cleaned, thematically analyzed, and visualized in a wordcloud. Here is wordcloud from a recent stream of tweets (ie Twitter messages) from US Presidential candidates Hillary Clinton and Donald Trump. The larger words implies greater frequency of occurrence in the tweets. This can help understand the major topics of discussion between the two.

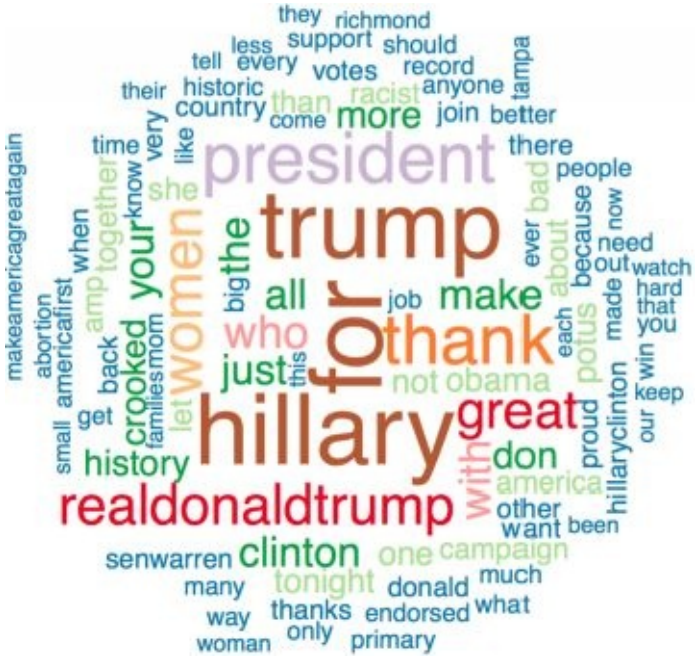


Figure 1.7: A wordcloud of Hillary Clinton's and Donald Trump's tweets

Technology Challenges for Big Data

There are four major technological challenges, and matching layers of technologies to manage Big Data.

Storing Huge Volumes

The first challenge relates to storing huge quantities of data. No machine can be big enough to store the relentlessly growing quantity of data. Therefore, data needs to be stored in a large number of smaller inexpensive machines. However, with a large number of machines, there is the inevitable challenge of machine failure. Each of these commodity machines will fail at some point or another. Failure of a machine could entail a loss of data stored on it.

The first layer of Big Data technology helps store huge volumes of data, while avoiding the risk of data loss. It distributes data across the large cluster of inexpensive commodity machines, and ensures that every piece of data is stored on multiple machines to guarantee that at least one copy is always available. Hadoop is the most well-known clustering technology for Big Data. Its data storage pattern is called Hadoop Distributed File System (HDFS). This system is built on the patterns of Google's File systems, designed to store billions of pages and sort them to answer user search queries.

Ingesting streams at an extremely fast pace

The second challenge relates to the Velocity of data, i.e. handling torrential streams of data. Some of them may be too large to store, but must still be ingested and monitored. The solution lies in creating special ingesting systems that can open an unlimited number of channels for receiving data. These queuing systems can hold data, from which consumer applications can request and process data at their own pace.

Big Data technology manages this velocity problem, using a special stream-processing engine, where all incoming data is fed into a central queueing system. From there, a fork-shaped system sends data to batch processing as well as to stream processing directions. The stream processing engine can do its work while the batch processing does its work. Apache Spark is the most popular system for streaming applications.

Handling a variety of forms and functions of data

The third challenge relates to the structuring and access of all varieties of data that comprise Big Data. Storing them in traditional flat or relational file structures would be too wasteful and slow. The third layer of Big Data technology solves this problem by

storing the data in non-relational systems that relax many of the stringent conditions of the relational model. These are called NoSQL (Not Only SQL) databases.

HBase and Cassandra are two of the better known NoSQL databases systems. HBase, for example, stores each data element separately along with its key identifying information. This is called a key-value pair format. Cassandra stores data in a document format. There are many other variants of NoSQL databases. NoSQL languages, such as Pig and Hive, are used to access this data.

Processing data at huge speeds

The fourth challenge relates to moving large amounts of data from storage to the processor, as this would consume enormous network capacity and choke the network. The alternative and innovative mode would be to move the processor to the data.

The second layer of Big Data technology avoids the choking of the network. It distributes the task logic throughout the cluster of machines where the data is stored. Those machines work, in parallel, on the data assigned to them, respectively. A follow-up process consolidates the outputs of all the small tasks and delivers the final results. MapReduce, also invented by Google, is the best-known technology for parallel processing of distributed Big Data.

Table 1.1: Technological challenges and solutions for Big Data

Challenge	Description	Solution	Technology
Volume	Avoid risk of data loss from machine failure in clusters of commodity machines	Replicate segments of data in multiple machines; master node keeps track of segment location	HDFS
Volume & Velocity	Avoid choking of network bandwidth by moving large volumes of data	Move processing logic to where the data is stored; manage using parallel processing algorithms	Map-Reduce
Variety	Efficient storage of large and small data objects	Columnar databases using key-pair values format	HBase, Cassandra
Velocity	Monitoring streams too large to store	Fork-shaped architecture to process data as stream and as batch	Spark

Once these major technological challenges are met, all traditional analytical and presentation tools can be applied to Big Data. There are many additional supportive technologies to make the task of managing Big Data easier. For example, a resource manager (such as YARN) can help monitor the resource usage and load balancing of the machines in the cluster.

Conclusion and Summary

Big Data is a major phenomenon that impacts everyone, and is an opportunity to create new ways of working. Big Data is extremely large, complex, fast, and not always clean, it is data that comes from many sources such as people, web, and machine communications. It needs to be gathered, organized and processed in a cost-effective way that manages the volume, velocity, variety and veracity of Big Data. Hadoop and Spark systems are popular technological platforms for this purpose. Here is a list of the many differences between traditional and Big Data.

Table 1.2: Comparing Big Data with Traditional Data

Feature	Traditional Data	Big Data
Representative Structure	Lake / Pool	Flowing Stream / river
Primary Purpose	Manage business activities	Communicate, Monitor
Source of data	Business transactions, documents	Social media, Web access logs, machine generated
Volume of data	Gigabytes, Terabytes	Petabytes, Exabytes
Velocity of data	Ingest level is controlled	Real-time unpredictable ingest
Variety of data	Alphanumeric	Audio, Video, Graphs, Text
Veracity of data	Clean, more trustworthy	Varies depending on source
Structure of data	Well-Structured	Semi- or Un-structured
Physical Storage of Data	In a Storage Area Network	Distributed clusters of commodity computers
Database organization	Relational databases	NoSQL databases
Data Access	SQL	NoSQL such as Pig
Data Manipulation	Conventional data processing	Parallel processing
		Dynamic dashboards with simple

Data Visualization	Variety of tools	measures
Database Tools	Commercial systems	Open-source - Hadoop, Spark
Total Cost of System	Medium to High	high

Organization of the rest of the book

This book will cover applications, architectures, and the essential Big Data technologies. The rest of the book is organized as follows.

Section 1 will discuss sources, applications, and architectural topics. Chapter 2 will discuss a few compelling business applications of Big Data, based on the understanding of the different sources and formats of data. Chapter 3 will cover some examples of architectures used by many Big Data applications.

Section 2 will discuss the six major technology elements identified in the Big Data Ecosystem (Figure 1.5). Chapter 4 will discuss Hadoop and how its Distributed File system (HDFS) works. Chapter 5 will discuss MapReduce and how this parallel processing algorithm works. Chapter 6 will discuss NoSQL databases to learn how to structure the data into databases for fast access. Pig and Hive languages, for data access, will be included. Chapter 7 will cover streaming data, and the systems for ingesting and processing this data. This chapter will cover Spark, an integrated, in-memory processing toolset to manage Big Data. Chapter 8 will cover Data ingest system, with Apache Kafka. Chapter 9 will be a primer on Cloud Computing technologies used for renting storage and computers at third party locations.

Section 3 will include Primers and tutorials. Chapter 10 will present a case study on the web log analyzer, an application that ingests a log of a large number of web request entries every day and can create summary and exception reports. Chapter 11 will be a primer on data analytics technologies for analyzing data. A full treatment can be found in my book, Data Analytics Made Accessible. Appendix 1 will be a tutorial on installing Hadoop cluster on Amazon EC2 cloud. Appendix 2 will be a tutorial on installing and using Spark.

Review Questions

Q1. What is Big Data? Why should anyone care?

Q2. Describe the 4V model of Big Data.

Q3. What are the major technological challenges in managing Big Data?

Q4: What are the technologies available to manage Big Data?

Q5. What kind of analyses can be done on Big Data?

Q6: Watch Cloudera CEO present the evolution of Hadoop at https://www.youtube.com/watch?v=S9xnYBVqLws . Why did people not pay attention to Hadoop and MapReduce when it was introduced? What implications does it have to emerging technologies?

Liberty Stores Case Exercise: Step B1

Liberty Stores Inc. is a specialized global retail chain that sells organic food, organic clothing, wellness products, and education products to enlightened LOHAS (Lifestyles of the Healthy and Sustainable) citizens worldwide. The company is 20 years old, and is growing rapidly. It now operates in 5 continents, 50 countries, 150 cities, and has 500 stores. It sells 20000 products and has 10000 employees. The company has revenues of over $5 billion and has a profit of about 5% of its revenue. The company pays special attention to the conditions under which the products are grown and produced. It donates about one-fifth (20%) from its pre-tax profits from global local charitable causes.

Q1: Create a comprehensive Big Data strategy for the CEO of the company.

Q2: How can Big Data systems such as IBM Watson help this company?

Section 1

This section covers three important high-level topics.

Chapter 2 will cover big data sources, and many applications in many industries.

Chapter 3 will architectures for managing big data

Chapter 2 - Big Data Applications

Introduction

If a traditional software application is a lovely cat, then a Big Data application is a powerful tiger. An ideal Big Data application will take advantage of all the richness of data and produce relevant information to make the organization responsive and successful. Big Data applications can align the organization with the totality of natural laws, the source of all success.

Companies like the consumer goods giant, Proctor & Gamble, have inserted Big Data into all aspects of its planning and operations. The industrial giant, Volkswagen, asks all its business units to identify some realistic initiative using Big Data to grow their unit's sales. The entertainment giant, Netflix, processes 400 billion user actions every day, and these are some of the biggest users of Big Data.

Figure 2-0-1: Big Data application is a powerful tiger (Source: Flickr.com)

CASELET: Big Data Gets the Flu

Google Flu Trends was an enormously successful influenza forecasting service, pioneered by Google. It employed Big Data, such as the stream of search terms used in its ubiquitous Internet search service. The program aimed to better predict flu outbreaks using data and information from the U.S. Centers for Disease Control and Prevention (CDC). What was most amazing was that this application was able to predict the onset of flu, almost two weeks before CDC saw it coming. From 2004 till about 2012 it was able to successfully predict the timing and geographical location of the arrival of the flu season around the world.

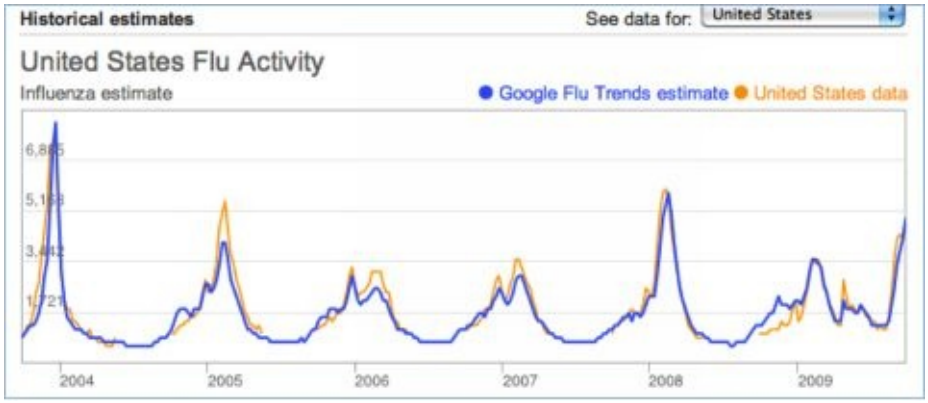


Figure 2-0-2: Google Flu trends

However, it failed spectacularly to predict the 2013 flu outbreak. Data used to predict Ebola's spread in 2014-15 yielded wildly inaccurate results, and created a major panic. Newspapers across the globe spread this application's worst-case scenarios for the Ebola outbreak of 2014.

Google Flu Trends failed for two reasons: Big Data hubris, and algorithmic dynamics, (a) The quantity of data does not mean that one can ignore foundational issues of measurement and construct validity and reliability and dependencies among data and (b) Google Flu Trends predictions were based on a commercial search algorithm that frequently changes, based on Google's business goals. This uncertainty skewed the data in ways even Google engineers did not understand, even skewing the accuracy of predictions. Perhaps the biggest lesson is that there is far less information in the data, typically available in the early stages of an outbreak, than is needed to parameterize the test models.

Q1: What lessons would you learn from the death of a prominent and highly successful Big Data application?

Q2: What other Big Data applications can be inspired from the success of this application?

Big Data Sources

Big Data is inclusive of all data about all activities everywhere. It can, thus, potentially transform our perspective on life and the universe. It brings new insights in real-time and can make life happier and make the world more productive. Big Data can, however, also bring perils—in terms of violation of privacy, and social and economic disruption.

There are three major categories of data sources: human communications, human-machine communications, and machine-machine communications.

People to People Communications

People and corporations increasingly communicate over electronic networks. Distance and time have been annihilated. Everyone communicates through phone and email. News travels instantly. Influential networks have expanded. The content of communication has become richer and multimedia. High-resolution cameras in mobile phones enable people to take pictures and videos, and instantly share them with friends and family. All these communications are stored in the facilities of many intermediaries, such as telecom and internet service providers. Social media is a new, but particularly transformative type of human-human communications.

Social Media

Social media platforms such as Facebook, Twitter, LinkedIn, YouTube, Flickr, Tumblr, Skye, Snapchat, and others have become an increasingly intimate part of modern life. These are among the hundreds of social media that people use and they generate huge streams of text, pictures, videos, logs, and other multimedia data.

People share messages and pictures through social media such as Facebook and YouTube. They share photo albums through Flickr. They communicate in short asynchronous messages with each other on Twitter. They make friends on Facebook, and follow others on Twitter. They do video conferencing, using Skype and leaders deliver messages that sometimes go viral through social media. All these data streams are part of Big Data, and can be monitored and analyzed to understand many phenomena, such as patterns of communication, as well as the gist of the conversations. These media have been used for a wide variety of purposes with stunning effects.

Figure 2-0-3: Sampling of major social media

People to Machine Communications

Sensors and web are two of the kinds of machines that people communicate with. Personal assistants such as Siri and Cortana are the latest in man-machine communications as they try to understand human requests in natural language, and fulfil them. Wearable devices such as FitBit and smart watch are smart devices that read, store and analyze people's personal data such as blood pressure and weight, food and exercise data, and sleep patterns. The world-wide web is like a knowledge machine that people interact with to get answers for their queries.

Web access

The world-wide-web has integrated itself into all parts of human and machine activity. The usage of the tens of billions of pages by billions of web users generates huge amount of enormously valuable clickstream data. Every time a web page is requested, a log entry is generated at the provider end. The webpage provider tracks the identity of the requesting device and user, and time and spatial location of each request. On the requester side, there are certain small pieces of computer code and data called cookies which track the webpages received, date/time of access, and some identifying information about the user. All the web access logs, and cookie records, can provide web usage records that can be analyzed for discovering opportunities for marketing purposes.

A web log analyzer is an application required to monitor streaming web access logs in real-time to check on website health and to flag errors. A detailed case study of a practical development of this application is shown in chapter 8.

Machine to Machine (M2M) Communications

M2M communications is also sometimes called the Internet of Things (IoT). A trillion devices are connected to the internet and they communicate with each other or some master machines. All this data can be accessed and harnessed by makers and owners of those machines.

Machines and equipment have many kinds of sensors to measure certain environmental parameters, which can be broadcast to communicate their status. RFID tags and sensors embedded in machines help generate the data. Containers on ships are tagged with RFID tags that convey their location to all those who can listen. Similarly, when pallets of goods are moved in warehouses or large retain stores, those pallets contain electromagnetic (RFID) tags that convey their location. Cars carry an RFID transponder to identify themselves to automated tollbooths and pay the tolls. Robots in a factory, and internet-connected refrigerators in a house, continually broadcast a 'heartbeat' that they are functionally normally. Surveillance videos using commodity cameras are another major source of machine-generated data.

Automobiles contain sensors that record and communicate operational data. A modern car can generate many megabytes of data every day, and there are more than 1 billion motor vehicles on the road. Thus the automotive industry itself generate huge amounts of data. Self-driving cars would only add to the quantity of data generated.

RFID tags

An RFID tag is a radio transmitter with a little antenna that can respond to and communicate essential information to special readers through Radio Frequency (RF) channel. A few years ago, major retailers such as Walmart decided to invest in RFID technology to take the retail industry to a new level. It forced their suppliers to invest in RFID tags on the supplied products. Today, almost all retailers and manufacturers have implemented RFID-tags based solutions.

Figure 2-0-4: A small passive RFID tag

Here is how an RFID tag works. When a passive RFID tag comes in the vicinity of an RF reader and is 'tickled', the tag responds by broadcasting a fixed identifying code. An

active RFID tag has its own battery and storage, and can store and communicate a lot more information. Every reading of message from an RFID tag by an RF reader creates a log entry. Thus there is a steady stream of data from every reader as it records information about all the RFID tags in its area of influence. The records may be logged regularly, and thus there will be many more records than are necessary to track the location and movement of an item. All the duplicate and redundant records is removed, to produce clean, consolidated data about the location and status of items.

Sensors

A sensor is a small device that can observe and record physical or chemical parameters. Sensors are everywhere. A photo sensor in the elevator or train door can sense if someone is moving and to thus keep the door from closing. A CCTV camera can record a video for surveillance purposes. A GPS device can record its geographical location every moment.

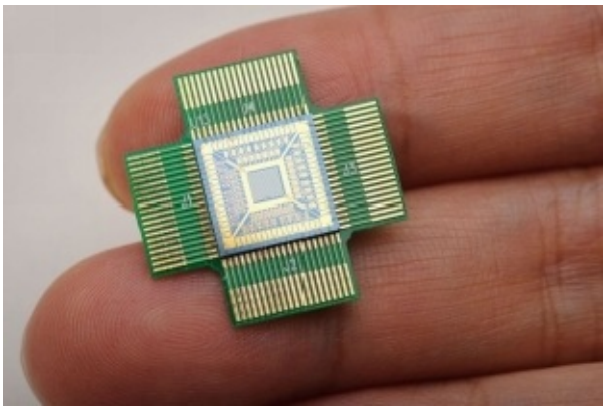


Figure 2-0-5: An embedded sensor

Temperature sensors in a car can measure the temperature of the engine and the tires and more. The thermostat in a building or a refrigerator too have temperature sensors. A pressure sensor can measure the pressure inside an industrial boiler.

Big Data Applications

Monitoring and Tracking Applications

Public Health Monitoring

The US government is encouraging all healthcare stakeholders to establish a national platform for interoperability and data sharing standards. This would enable secondary use of health data, which would advance Big Data analytics and personalized holistic precision medicine. This would be a broad-based platform like the Google Flu Trends case.

Consumer Sentiment Monitoring

Social Media has become more powerful than advertising. Many consumer goods companies have moved a bulk of their marketing budgets from traditional advertising media into social media. They have set up Big Data listening platforms, where Social Media data streams (including tweets and Facebook posts and blog posts) are filtered and analyzed for certain keywords or sentiments, by certain demographics and regions. Actionable information from this analysis is delivered to marketing professionals for appropriate action, especially when the product is new to the market.

Figure 2-0-6: Architecture for a Listening Platform (source: Intelligenthq.com)

Asset tracking

The US Department of Defense is encouraging the industry to devise a tiny RFID chip that could prevent the counterfeiting of electronic parts that end up in avionics or circuit boards for other devices. Airplanes are one of the heaviest users of sensors which track every aspect of the performance of every part of the plane. The data can be displayed on

the dashboard, as well as stored for later detailed analysis. Working with communicating devices, these sensors can produce a torrent of data.

Theft by visitors, shoppers and even employees, is a major source of loss of revenue for retailers. All valuable items in the store can be assigned RFID tags, and the gates of the store are equipped with RF readers. This helps secure the products, and reduce leakage (theft), from the store.

Supply chain monitoring

All containers on ships communicate their status and location using RFID tags. Thus, retailers and their suppliers can gain real-time visibility to the inventory throughout the global supply chain. Retailers can know exactly where the items are in the warehouse, and so can bring them into the store at the right time. This is particularly relevant for seasonal items that need to be sold on time, or else they will be sold at a discount. With item-level RFID tacks, retailers also gain full visibility of each item and can serve their customers better.

Electricity Consumption Tracking

Electric utilities can track the status of generating and transmission systems, and also measure and predict the consumption of electricity. Sophisticated sensors can help monitor voltage, current, frequency, temperature, and other vital operating characteristics of huge and expensive electric distribution infrastructure. Smart meters can measure the consumption of electricity at regular intervals of one hour or less. This data is analyzed to make real-time decisions to maximize power capacity utilization and the total revenue generation.

Preventive Machine Maintenance

All machines, including cars and computers, will fail sometime, because one or more or their components will fail. Any precious equipment could be equipped with sensors. The continuous stream of data from the sensors data could be monitored and analyzed to forecast the status of key components, and thus, monitor the overall machine’s health. Preventive maintenance can be scheduled to reduce the cost of downtime.

Analysis and Insight Applications

Big Data can be structured and analyzed using data mining techniques to produce insights and patterns that can be used to make business better.

Predictive Policing

The Los Angeles Police Department (LAPD) invented the concept of Predictive Policing. The LAPD worked with UC Berkeley researchers to analyze its large database of 13 million crimes recorded over 80 years, and predicted the likeliness of crimes of certain types, at certain times, and in certain locations. They identified hotspots of crime where crimes had occurred, and where crime was likely to happen in the future. Crime patterns were mathematically modeled after a simple insight borrowed from a metaphor of earthquakes and its aftershocks. In essence, it said that once a crime occurred in a location, it represented a certain disturbance in harmony, and would thus, lead to a greater likelihood of a similar crime occurring in the local vicinity in the near future. The model showed for each police beat, the specific neighborhood blocks and specific time slots, where crime was likely to occur.

Figure 2-0-7: LAPD officer on predicting policing (Source: nbclosangeles.com)

By including the police cars' patrol schedules in accordance with the model's predictions, the LAPD was able to reduce crime by 12% to 26% for different categories of crime. Recently, the San Francisco Police Department released its own crime data for over 2 years, so data analysts could model that data and prevent future crimes.

Winning Political Elections

The US President, Barack Obama, was the first major political candidate to use Big Data in a significant way, in the 2008 elections. He is the first Big Data president. His campaign gathered data about millions of people, including supporters. They invented the "Donate Now" button for use in emails to obtain campaign contributions from millions of supporters. They created personal profiles of millions of supporters and what they had done and could do for the campaign. Data was used to determine undecided voters who could be converted to their side. They provided phone numbers of these undecided voters to the supporters to call, and then recorded the outcome of those calls all over the web,

using interactive applications. Obama himself used his twitter account to communicate his messages directly with his millions of followers.

After the elections, Obama converted the list of supporters to an advocacy machine that would provide the grassroots support for the President's initiatives. Since then, almost all campaigns use Big Data. Senator Bernie Sanders used the same Big Data playbook to build an effective national political machine powered entirely by small donors. Analyst, Nate Silver, created sophistical predictive models using inputs from many political polls and surveys to win pundits to successfully predict winners of the US elections. Nate was however, unsuccessful in predicting Donald Trump's rise, and that shows the limits of Big Data.

Personal Health

Correct diagnosis is the sine qua non of effective treatment. Medical knowledge and technology is growing by leaps and bounds. IBM Watson is a Big Data Analytics engine that ingests and metabolizes all the medical information in the world, and then applies it intelligently to an individual situation. Watson can provide a detailed and accurate medical diagnosis using current symptoms, patient history, medication history, and environmental trends, and other parameters. Similar products might be offered as an App to licensed doctors, and even individuals, to improve productivity and accuracy in health care.

New Product Development

These applications are totally new concepts that did not exist earlier.

Flexible auto insurance

An auto insurance company can use the GPS data from cars to calculate the risk of accidents based on travel patterns. The automobile companies can use the car sensor data to track the performance of a car. Safer drivers can be rewarded and the errant drivers can be penalized.

Figure 2-0-8: GPS based tracking of vehicles

Location-based retail promotion

A retailer, or a third-party advertiser, can target customers with specific promotions and coupons based on location data obtained through GPS, the time of day, the presence of stores nearby, and mapping it to the consumer preference data available from social media databases. Ads and offers can be delivered through mobile apps, SMS, and email. These are examples of mobile apps.

Recommendation service

Ecommerce is a fast growing industry in the last couple of decades. A variety of products are sold and shared over the internet. Web users' browsing and purchase history on ecommerce sites is utilized to learn about their preferences and needs, and to advertise relevant product and pricing offers in real-time. Amazon uses a personalized recommendation engine system to suggest new additional products to consumers based on affinities of various products. Netflix also uses a recommendation engine to suggest entertainment options to its users.

Conclusion

Big Data has applicability across all industries. There are three major types of data sources of Big Data. They are people-people communications, people-machine communications, and machine-machine communications. Each type has many sources of data. There are three types of applications. They are the monitoring type, the analysis type, and new product development. This chapter presents a few business applications of each of those three types.

Review Questions

Q1: What are the major sources of Big Data? Describe a source of each type.

Q2: What are the three major types of Big Data applications? Describe two applications of each type.

Q3: Would it be ethical to arrest someone based on a Big Data Model's prediction of that person likely to commit a crime?

Q4: An auto insurance company learned about the movements of a person based on the GPS installed in the vehicle. Would it be ethical to use that as a surveillance tool?

Q5: Research can describe a Big Data application that has a proven return on investment for an organization.

Liberty Stores Case Exercise: Step B2

The Board of Directors asked the company to take concrete and effective steps to become a data-driven company. The company wants to understand its customers better. It wants to improve the happiness levels of its customers and employees. It wants to innovate on new products that its customers would like. It wants to relate its charitable activities to the interests of its customers.

Q1: What kind of data sources should the company capture for this?

Q2: What kind of Big Data applications would you suggest for this company?

Chapter 3 - Big Data Architecture

Introduction

Big Data Application Architecture is the configuration of tools and modules to accomplish the whole task. An ideal architecture would be resilient, secure, cost-effective, and adaptive to new needs and environments. This is achieved through beginning with proven architectures, and creatively and progressively restructuring it with new elements as additional needs and problems arise. Big Data architectures ultimately align with the architecture of the Universe, the source of all invincibility.

CASELET: Google Query Architecture

Google invented the first Big Data architecture. Their goal was to gather all the information on the web, organize it, and search it for specific queries from millions of users. An additional goal was to find a way to monetize this service by serving relevant and prioritized online advertisements on behalf of clients.

Google developed web crawling agents which would follow all the links in the web and make a copy of all the content on all the webpages it visited.

Google invented cost-effective, resilient, and fast ways to store and process all that exponentially growing data. It developed a scale-out architecture in which it could linearly increase its storage capacity by inserting additional computers into its computing network. The data files were distributed over the large number of machines in the cluster. This distributed files system was called the Google File system, and was the precursor to HDFS.

Google would sort or index the data thus gathered so it can be searched efficiently. They invented the key-pair NoSQL database architecture to store variety of data objects. They developed the storage system to avoid updates in the same place. Thus the data was written once, and read multiple times.

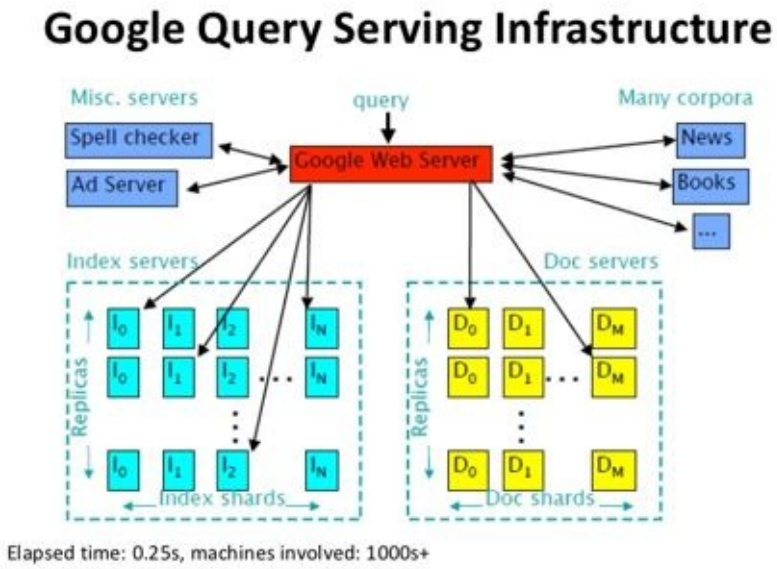


Figure 3-0-1: Google Query Architecture

Google developed the MapReduce parallel processing architecture whereby large datasets could be processed by thousands of computers in parallel, with each computer processing a chunk of data, to produce quick results for the overall job.

The Hadoop ecosystem of data management tools like Hadoop distributed file system (HDFS), columnar database system like HBase, a querying tool such as Hive, and more, emerged from Google's inventions. Storm is a streaming data technologies to produce instant results. Lambda Architecture is a Y-shaped architecture that branches out the incoming data stream for batch as well as stream processing.

Q1: Why should Google publish its File System and the MapReduce parallel programming system and send it into open-source system?

Q2: What else can be done with Google's repository of all the web's data?

Standard Big data architecture

Here is the generic Big Data Architecture introduced in Chapter 1. There are many sources of data. All data is funneled in through an ingest system. The data is forked into two sides: a stream processing system and a batch processing system. The outcome of these processing can be sent into NoSQL databases for later retrieval, or sent directly for consumption by many applications and devices.

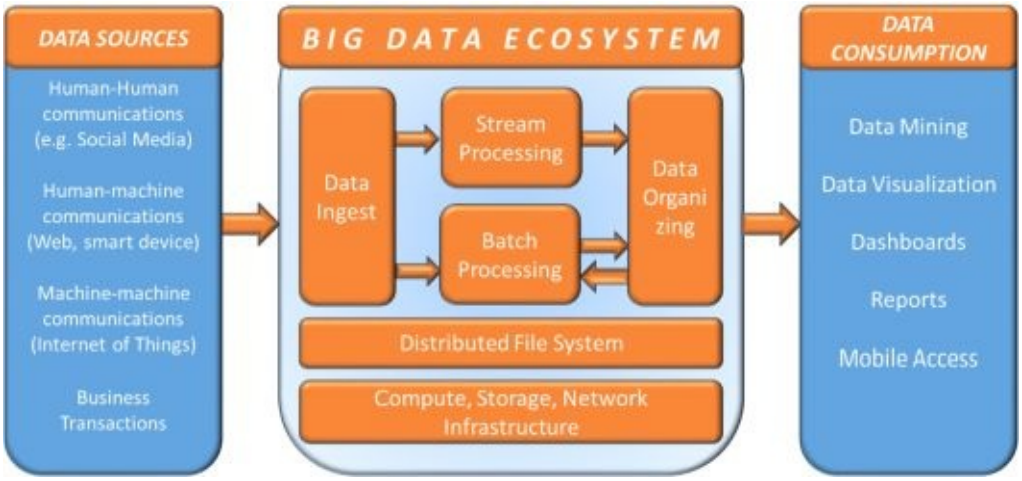


Figure 3-0-2: Big Data Application Architecture

A big data solution typically comprises these as logical layers. Each layer can be represented by one or more available technologies.

Big data sources: The sources of data for an application depends upon what data is required to perform the kind of analyses you need. The various sources of Big data were described in chapter 2. The data will vary in origin, size, speed, form, and function, as described by the 4 Vs in chapter 1. Data sources can be internal or external to the organization. The scope of access to data available could be limited. The level of structure could be high or low. The speed of data and its quantity will also by high or low depending upon the data source.

Data ingest layer: This layer is responsible for acquiring data from the data sources. The data is through a scalable set of input points that can acquire at various speeds and in various quantities. The data is sent to a batch processing system, a stream processing system, or directly to a storage file system (such as HDFS). Compliance regulations and governance policies impact what data can be stored and for how long.

Batch Processing layer: The analysis layer receives data from the ingest point or from the file system or from the NoSQL databases. Data is processed using parallel programming techniques (such as MapReduce) to process it and produce the desired results. This batch processing layer thus needs to understand the data sources and data types, the algorithms

that would work on that data, and the format of the desired outcomes. The output of this layer could be sent for instant reporting, or stored in a NoSQL databases for an on-demand report, for the client.

Streaming Processing layer: This layer receives data directly from the ingest point. Data is processed using parallel programming techniques (such as MapReduce) to process it in real time, and produce the desired results. This layer thus needs to understand the data sources and data types extremely well, and the super-light algorithms that would work on that data to produce the desired results. The outcome of this layer too could be stored in the NoSQL Databases.

Data Organizing Layer: This layer receives data from both the batch and stream processing layers. Its objective is to organize the data for easy access. It is represented by NoSQL databases. SQL-like languages like Hive and Pig can be used to easily access data and generate reports.

Data Consumption layer: This layer consumes the output provided by the analysis layers, directly or through the organizing layer. The outcome could be standard reports, data analytics, dashboards and other visualization applications, recommendation engine, on mobile and other devices.

Infrastructure Layer: At bottom there is a layer that manages the raw resources of storage, compute, and communication. This is increasingly provided through a cloud computing paradigm.

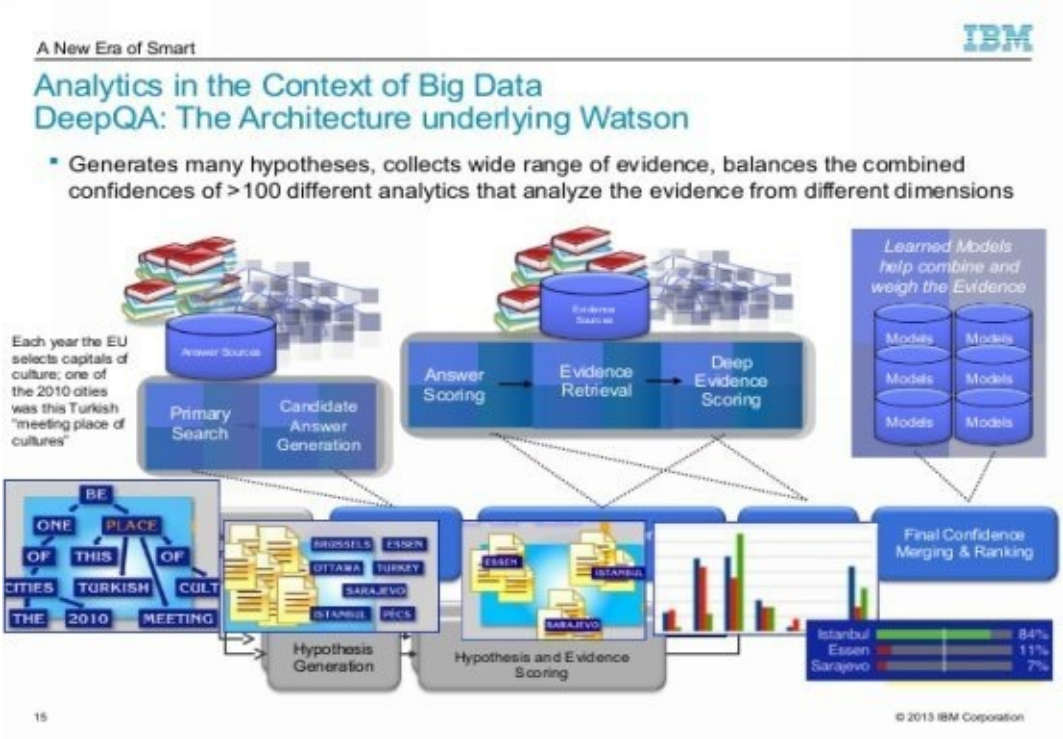
Distributed File System Layer: It would also include the Hadoop Distributed File System (HDFS). It would also include supporting applications, such as YARN (Yet Another Resource Manager), that enable the efficient access to data storage and its transfer.

Big Data Architecture examples

Every major organization and applications has a unique optimized infrastructure to suit its specific needs. Here below are some architecture examples from some very prominent users and designers of Big Data applications.

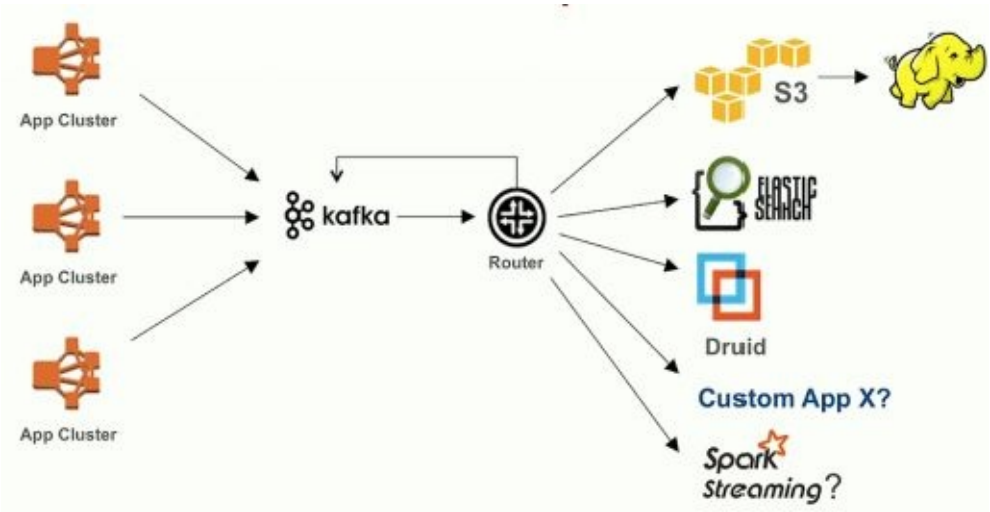
IBM Watson

IBM Watson uses Spark to manage incoming data streams. It also uses Spark's Machine Learning library (MLLib) to analyze data and predict diseases.



Netflix

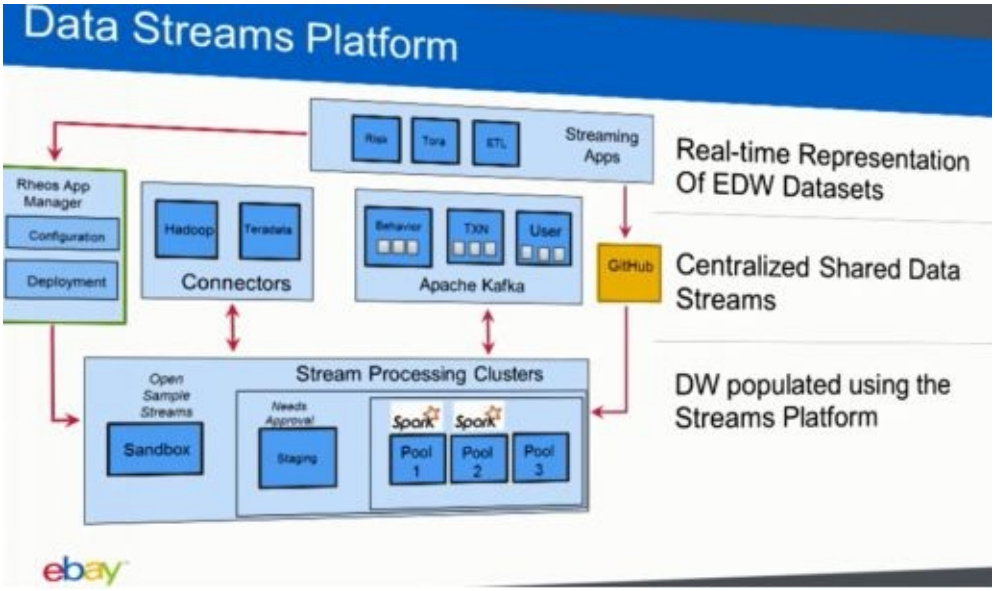
This is one of the largest providers of online video entertainment. They handle 400 Billion online events per day. As a cutting-edge user of big data technologies, they are constantly innovating their mix of technologies to deliver the best performance. Kafka is the common messaging system for all incoming requests. They host the entire infrastructure on Amazon Web Services (AWS). The database is AWS' S3 as well as Cassandra and Hbase to store data. Spark is used for stream processing.



(Source: Netflix)

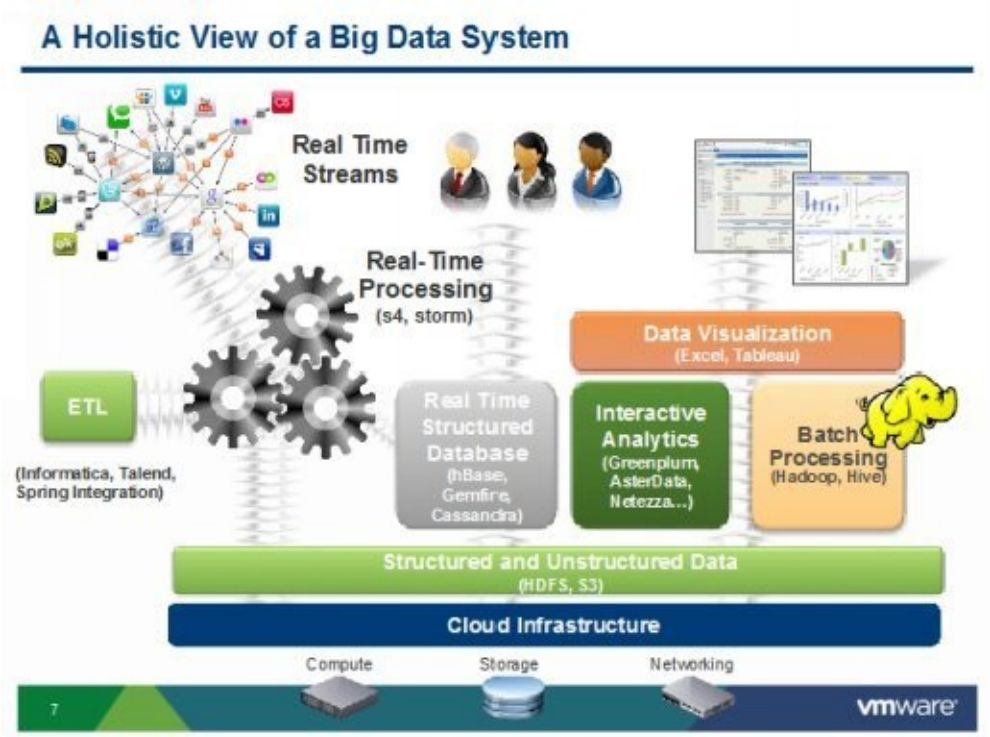
Ebay

Ebay is the second-largest Ecommerce company in the world. It delivers 800 million listings from 25 million sellers to 160 million buyers. To manage this huge stream of activity, EBay uses a stack of Hadoop, Spark, Kafka, and other elements. They think that Kafka is the best new thing for processing data streams.



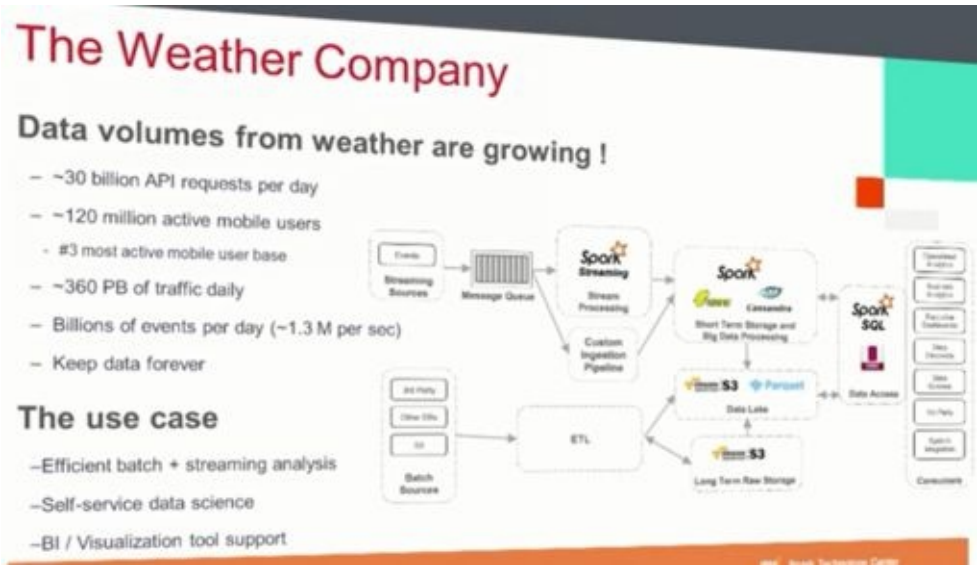
VMWare

Here is VMware's view of a Big Data architecture. It is similar to, but more detailed than, our main big architecture diagram.



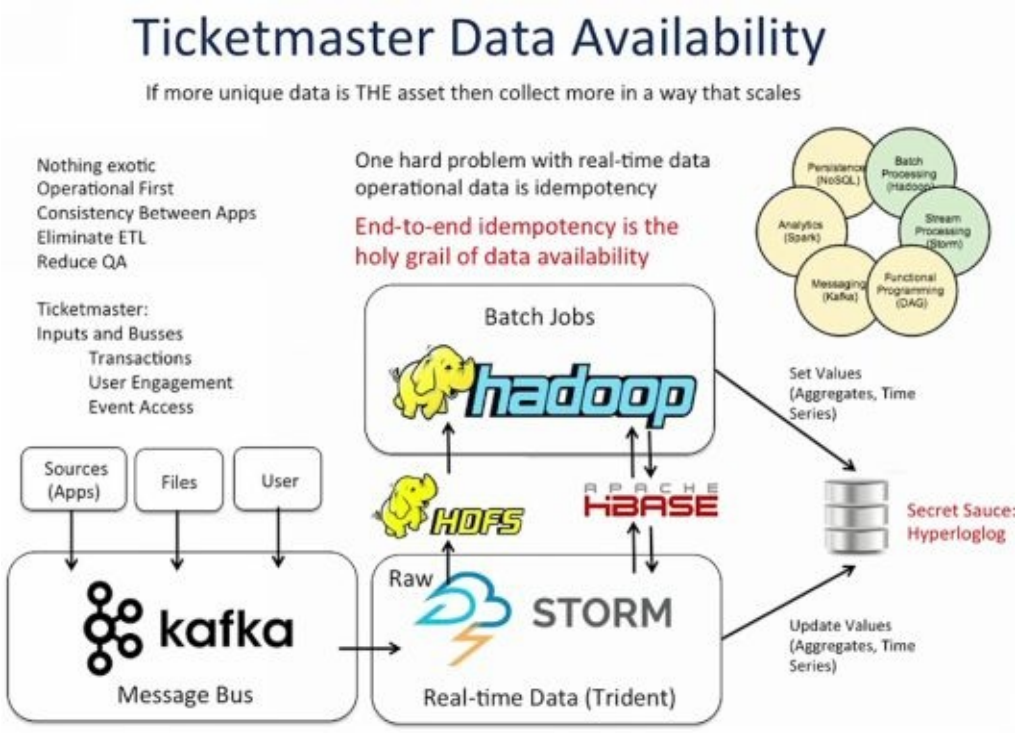
The Weather Company

The Weather company serves weather data globally through websites and mobile apps. It uses streaming architecture using Apache Spark.



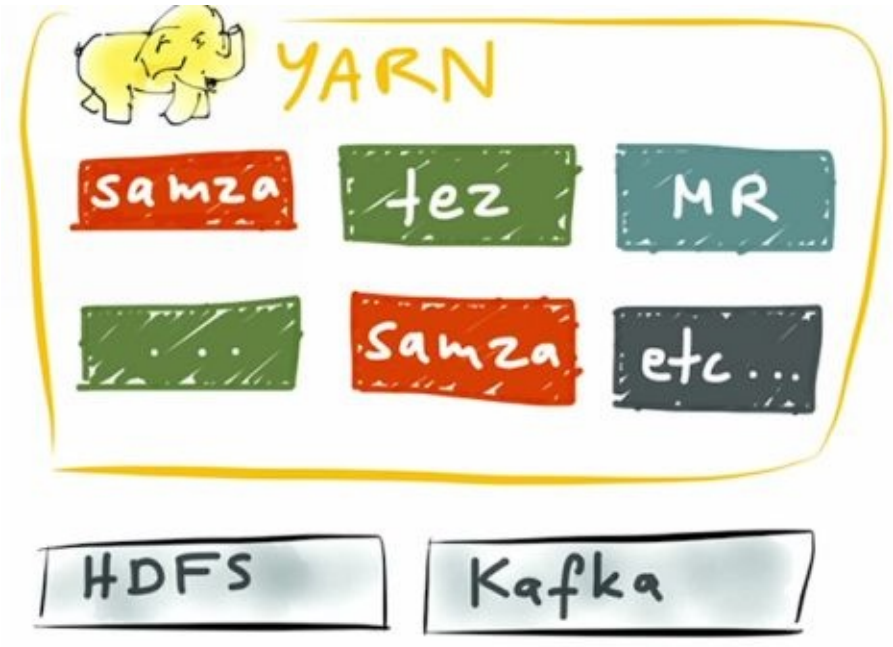
TicketMaster

This is the world's largest company that sells event tickets. Their goal is to make tickets available to purchase for real fans, and prevent bad actors from manipulating the system to increase the price of the tickets in the secondary markets.



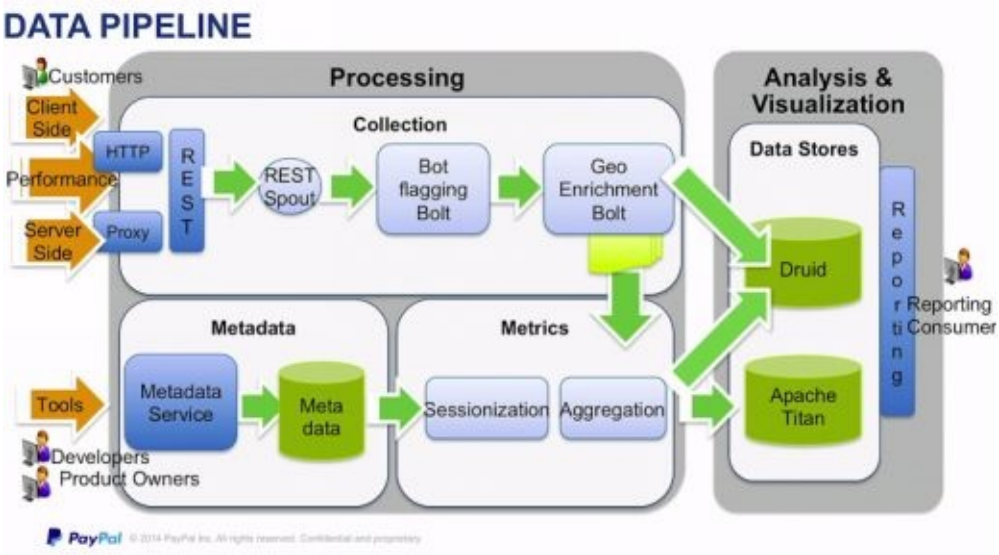
LinkedIn

The goal of this professional networking company is to maintain an efficient system for processing the streaming data and make the link options available in real-time.



Paypal

This payments-facilitation company needs to understand and acquire customers, and process a large number of payment transactions.



CERN

This premier high-energy physics research lab compute petabytes of data using in-memory stream processing to process data from millions of sensors and devices.

Conclusion

Big Data applications are architected to do stream as well as batch processing. Data is ingested and fed into streaming and batch processing. Most tools used for big data processing are open source tools served through the Apache community, and some key distributors of those technologies.

Review Questions

Q1: Describe the Big Data processing architecture.

Q2: What are Google’s contributions to Big data processing?

Q3: What are some of the hottest technologies visible in Big Data processing?

Liberty Stores Case Exercise: Step B3

The wants to build a scalable and futuristic platform for its Big Data.

Q1: What kind of Big Data Processing architecture would you suggest for this company

Section 2

This section covers the important Big Data technologies defined in the Big Data architecture specified in chapter 3.

Chapter 4 will cover Hadoop and its Distributed File System (HDFS)

Chapter 5 will cover the parallel processing algorithm, MapReduce.

Chapter 6 will NoSQL databases such as HBase and Cassandra. It will also cover Pig and Hive languages used for accessing those databases.

Chapter 7 will cover Spark, a fast and integrated streaming data management platform.

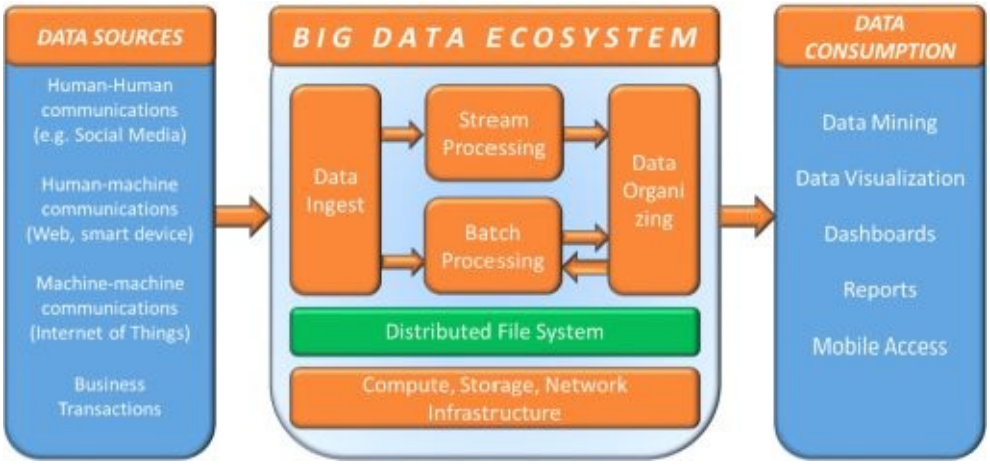
Chapter 8 will cover Data Ingest systems, using Apache Kafka

Chapter 9 will cover Cloud Computing model.

Chapter 4: Distributed Computing using Hadoop

Introduction

A distributed system is a clever way of storing huge quantities of data, securely and cost-effectively, for speed and ease, for retrieval and processing, using a networked collection of commodity machines. The ideal distributed file system would store infinite amounts of data while making the complexity completely transparent to the user, and enable easy access to the right data instantly. This would be achieved by storing fragments of data at different locations, and internally managing the lower-level tasks of storing and replicating data across the network. The distributed system ultimately leads to the creation of the unbounded cosmic computer that is aligned with the Unified Field of all the laws of nature.



Hadoop Framework

The Apache Hadoop distributed computing framework is composed of the following modules:

1. Hadoop Common – contains libraries and utilities needed by other Hadoop modules
2. Hadoop Distributed File System (HDFS) – a distributed file-system that stores data on commodity machines, providing very high aggregate bandwidth across the cluster
3. YARN – a resource-management platform responsible for managing computing resources in clusters and using them for scheduling of users' applications, and
4. MapReduce – an implementation of the MapReduce programming model for large scale data processing.

This chapter will cover Hadoop Common, HDFS, and YARN. The next chapter will cover MapReduce.

HDFS Design Goals

The Hadoop distributed file system (HDFS) is a distributed and scalable file-system. It is designed for applications that deal with large data sizes. It is also designed to deal with mostly immutable files, i.e. write data once, but read it many times.

HDFS has the following major design goals:

1. Hardware failure management – it will happen, and one must plan for it.
2. Huge volume – create capacity for large number of huge file sizes, with fast read/write throughput
3. High speed – create a mechanism to provide low latency access to streaming applications
4. High variety – Maintain simple data coherence, by writing data once but reading many times.
5. Open-source – Maintain easy accessibility of data using any hardware, software, and database platform
6. Network efficiency – Minimize network bandwidth requirement, by minimizing data movement

Master-Slave Architecture

Hadoop is an architecture for organizing computers in a master-slave relationship that helps achieve great scalability in processing. An HDFS cluster has two types of nodes operating in a master−worker pattern: a single master node (called NameNode), and a large number of slave worker nodes (called DataNodes).A small Hadoop cluster includes a single master and multiple worker nodes. A large Hadoop cluster would consist of a master and thousands of small ordinary machines as worker nodes.

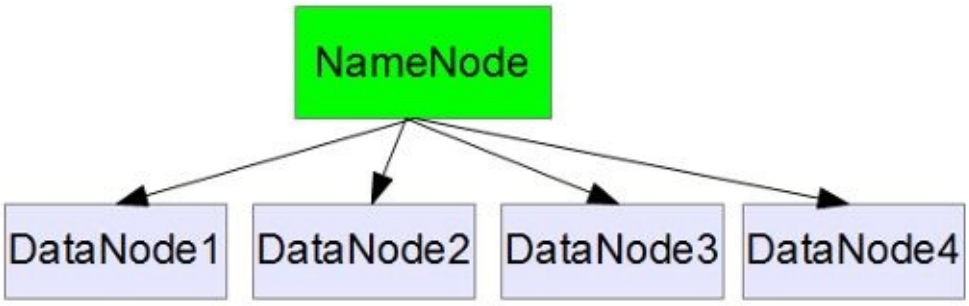


Figure 4-0-1: Master-Slave Architecture

The master node manages the overall file system, its namespace, and controls the access to files by clients. The master node is aware of the data-nodes: i.e. what blocks of which file are stored on which data node. It also controls the processing plan for all applications running on the data on the cluster. There is only one master node. Unfortunately, that makes it a single point of failure. Therefore, whenever possible, the master node has a hot backup just in case the master node dies unexpectedly. The master node uses a transaction log to persistently record every change that occurs to file system metadata.

The worker nodes store the data blocks in their storage space, as directed by the master node. Each worker node typical contains many disks to maximize storage capacity and access speed. Each worker node has its own local file system. A worker node has no awareness of the distributed file structure. It simply stores each block of data as directed, as if each block were a separate file. The DataNodes store and serve up blocks of data over the network using a block protocol, under the direction of the NameNode.

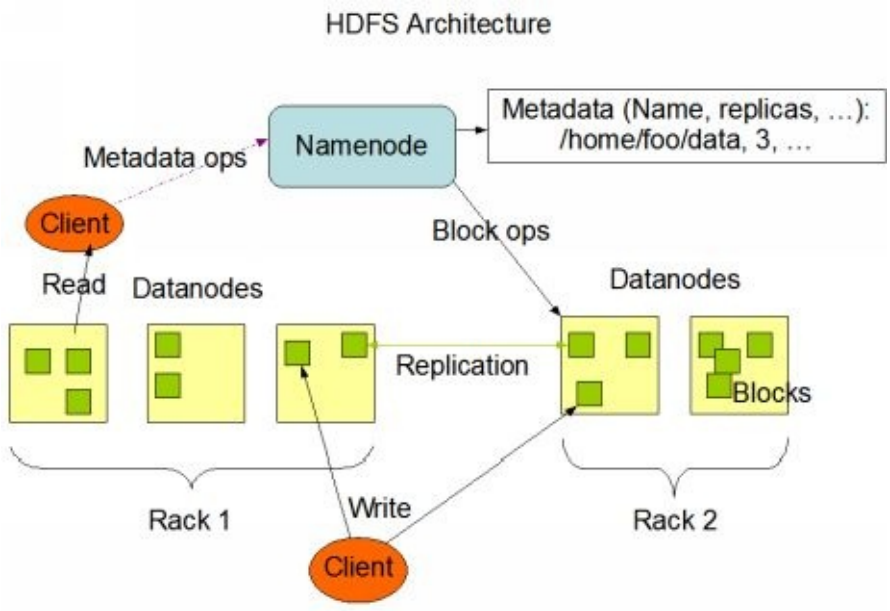


Figure 4-0-2: Hadoop Architecture (Source: Hadoop.apache.org)

The Namenode stores all relevant information about all the DataNodes, and the files stored in those DataNodes. The NameNode will contain:

- For every DataNode, its name, Rack, Capacity, and Health
- For every File, its Name, replicas, Type, Size, TimeStamp, Location, Health, etc.

It a DataNode fails, there is no serious problem. The data on the failed dataNode will be accessed from its replicas on other DataNodes. The failed DataNode can be automatically recreated on another machine, by writing all those file blocks of from the other healthy replicas. Each data-node sends a heartbeat message to the name-node periodically. Without this message, the DataNode is assumed to be dead. The DataNode replication effort would automatically kick-in to replace the dead data-node.

The file system has a set of features and capabilities to completely hide the splintering and scattering of data, and enable the user to deal with the data at a high, logical level.

The NameNode tries to ensure that files are evenly spread across the data-nodes in the cluster. That balances the storage and computing load, and also limits the extent of loss from the failure of a node. The NameNode also tries to optimize the networking load. When retrieving data or ordering the processing, the NameNode tries to pick Fragments from multiple nodes to balance the processing load and speed up the totally processing effort. The NameNode also tries to store fragments of files on the same node for speed of read and writing. Processing is done on the node where the file fragment is stored.

Any piece of data is stored typically on three nodes: two on the same rack, and one on a

different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high.

Block system

HDFS stores large files (typically gigabytes to terabytes) by storing segments (called blocks) of the file across multiple machines. A block of data is the fundamental storage unit in HDFS. Data files are described, read and written in block-sized granularity. All storage capacity and file sizes are measured in blocks. A block ranges from 16-128MB in size, with a default block size of 64MB. Thus, an HDFS file is chopped up into 64 MB chunks, and if possible, each chunk will reside on a different DataNode.

Every data file takes up a number of blocks depending upon its size. Thus a 100 MB file will occupy two blocks (100MB divided by 64MB), with some room to spare. Every storage disk can accommodate a number of blocks depending upon the size of the disk. Thus a 1 Terabyte storage will have 16000 blocks (1TB divided by 64MB).

Every file is organized as a consecutively numbered sequence of blocks. A file's blocks are stored physically close to each other for ease of access, as far as possible. The file's block size and replication factor are configurable by the application that writes the file on HDFS.

Ensuring Data Integrity

Hadoop ensures that no data will be lost or corrupted, during storage or processing. The files are written only once, and never updated in place. They can be read many times. Only one client can write or append to a file, at a time. No concurrent updates are allowed.

If a data is indeed lost or corrupted, or if a part of the disk gets corrupted, a new healthy replica for that lost block will be automatically recreated by copying from the replicas on other data-nodes. At least one of the replicas is stored on a data-node on a different rack. This guards against the failure of the rack of nodes, or the networking router, on it.

A checksum algorithm is applied on all data written to HDFS. A process of serialization is used to turn files into a byte stream for transmission over a network or for writing to persistent storage. Hadoop has additional security built in, using Kerberos verifier.

Installing HDFS

It is possible to run Hadoop on an in-house cluster of machines, or on the cloud inexpensively. As an example, The New York Times used 100 Amazon Elastic Compute Cloud (EC2) instances (DataNodes) and a Hadoop application to process 4 TB of raw image TIFF data stored in Amazon Simple Storage Service (S3) into 11 million finished PDFs in the space of 24 hours at a computation cost of about $240 (not including bandwidth). See Chapter 9 for a primer on Cloud Computing. See Appendix 1 for a step-by-step tutorial on installing Hadoop on Amazon EC2.

Hadoop is written in Java. Hadoop also requires a working Java installation. Installing Hadoop takes a lot of resources. For example, all information about fragments of files needs to be in Name-node memory. A thumb rule is that Hadoop needs approximately 1GB memory to manage 1M file fragments. Many easy mechanisms exist to install the entire Hadoop stack. Using a GUI such as Cloudera Resources Manager to install a Cloudera Hadoop stack is easy. This stack includes, HDFS, and many other related components, such as HBase, Pig, YARN, and more. Installing it on a cluster on a cloud services provider like AWS is easier than installing Java Virtual Machines (JVMs) on HDFS can be installed by using Cloudera GUI Resources Manager. If doing from command line, download Hadoop from one of the Apache mirror sites

Hadoop is written in Java. And most access to files is provided through Java abstract class org.apache.hadoop.fs.FileSystem. HDFS can be mounted directly with a Filesystem in Userspace (FUSE) virtual file system on Linux and some other Unix systems. File access can be achieved through the native Java application programming interface (API). Another API, called Thrift, helps to generate a client in the language of the users' choosing (such as C++, Java, Python). When the Hadoop command is invoked with a *classname* as the first argument, it launches a Java virtual machine (JVM) to run the class, along with the relevant Hadoop libraries (and their dependencies) on the *classpath*.

HDFS has a UNIX-like command like interface (CLI). Use *sh* shell to communicate with Hadoop. HDFS has UNIX-like permissions model for files and directories. There are three progressively increasing levels of permissions: read (r), write (w), and execute (x). Create a *hduser*, and communicate using *ssh* shell on the local machine.

```
% hadoop fs -help    ## get detailed help on every command.
```

Reading and Writing Local Files into HDFS

There are two different ways to transfer data: from the local file system, or form an

input/output stream. Copying a file from the local filesystem to HDFS can be done by:

```
% hadoop fs -copyFromLocal path/filename
```

Reading and Writing Data Streams into HDFS

Read a file from HDFS by using a java.net.URL object to open a stream to read the data requires a short script, as below.

```
InputStream in = null;
Start {
instream = new URL("hdfs://host/path").openStream();  // details of process in }
Finish  {  IOUtils.closeStream(instream); }
```

A simple method to create a new file is as follows:

```
public FSDataOutputStream create(Path p) throws IOException
```

Data can be appended to an existing file using the append() method:

```
public FSDataOutputStream append(Path p) throws IOException
```

A directory can be created by a simple method:

```
public boolean mkdirs(Path p) throws IOException
```

List the contents of a directory using:

```
public FileStatus[] listStatus(Path p) throws IOException
public FileStatus[] listStatus(Path p, PathFilter filter) throws IOException
```

Sequence Files

The incoming data files can range from very small to extremely large, and with different structures. Big Data files are therefore organized quite differently to handle the diversity of file sizes and type. Large files are stored as HDFS files, with FileFragments distributed across the cluster. However, smaller files should be bunched together into single segment for efficient storage.

Sequence Files are a specialized data structure within Hadoop to handle smaller files with smaller record sizes. Sequence File uses a persistent data structure for data available in key-value pair format. These help efficiently store smaller objects. HDFS and MapReduce are designed to work with large files, so packing small files into a Sequence File container, makes storing and processing the smaller files more efficient for HDFS and MapReduce.

Sequence files are row-oriented file formats, which means that the values for each row are stored contiguously in the file. This formats are appropriate when a large number of columns of a single row are needed for processing at the same time. There are easy commands to create, read and write Sequence File structures. Sorting and merging Sequence Files is native to MapReduce system. A MapFile is essentially a sorted Sequence File with an index to permit lookups by key.

YARN

YARN (Yet Another Resource Negotiator) is the architectural center of Hadoop, It is often characterized as a large-scale, distributed operating system for big data applications. YARN manages resources and monitors workloads, in a secure multi-tenant environment, while ensuring high availability across multiple Hadoop clusters. YARN also brings great flexibility as a common platform to run multiple tools and applications such as interactive SQL (e.g. Hive), real-time streaming (e.g. Spark), and batch processing (MapReduce), to work on data stored in a single HDFS storage platform. It brings clusters more scalability to expand beyond 1000 nodes, it also improves cluster utilization through dynamic allocation of cluster resources to various applications.

Figure 4-0-3: Hadoop Distributed Architecture including YARN

The Resource Manager in YARN has two main components: Scheduler and Applications Manager.

YARN Scheduler allocates resources to the various requesting applications. It does so based on an abstract notion of a resource **Container** which incorporates elements such as Memory, CPU, Disk storage, Network, etc. Each machine also has a NodeManager that manages all the Containers on that machine, and reports status on resources and Containers to the YARN Scheduler.

YARN Applications Manager accepts new job submissions from the client. It then requests a first resource Container for the application-specific ApplicationMaster program, and monitors the health and execution of the application. Once running, the ApplicationMaster directly negotiates additional resource containers from the Scheduler as needed.

Conclusion

Hadoop is the major technology for managing big data. HDFS securely stores data on large clusters of commodity machines. A master machine controls the storage and processing activities of the worker machines. A NameNode controls the namespace and storage information for the file system on the DataNodes. A master JobTracker controls the processing of tasks at the DataNodes. YARN is the resources manager that manages all resources dynamically and efficiently across all applications on the cluster. Hadoop File system and other parts of the Hadoop stack are distributed by many vendors, and can be easily installed on cloud computing infrastructure. Hadoop installation tutorial is in Appendix A.

Review Questions

Q1: How does Hadoop differ from a traditional file system?

Q2: What are the design goals for HDFS?

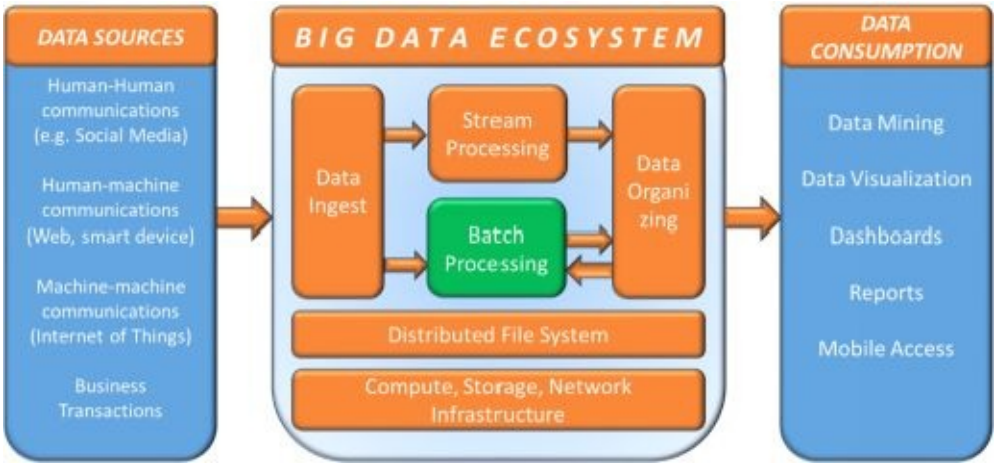
Q3: How does HDFS ensure security and integrity of data?

Q4: How does a master node differ from the worker node?

Chapter 5 – Parallel Processing with MapReduce

Introduction

A parallel processing system is a clever way to process huge amounts of data in a short period of time by enlisting the services of many computing devices to work on parts of the job, simultaneously. The ideal parallel processing system will work across any computational problem, using any number of computing devices, across any size of data sets, with ease and high programmer productivity. This is achieved by framing the problem in a way that it can be broken down into many parts, such that that each part can be partially processed independently of the other parts; and then the intermediate results from processing the parts can be combined to produce a final solution. Infinite parallel processing is the essence of infinite dynamism of the laws of nature.



MapReduce Overview

MapReduce is a parallel programming framework for speeding up large scale data processing for certain types of tasks. It achieves so with minimal movement of data on distributed file systems such as HDFS clusters, to achieve near-realtime results. There are two major pre-requisites for MapReduce programming. (a) The application must lend itself to parallel programming. (b) The data can be expressed in key-value pairs.

MapReduce processing is similar to UNIX sequence (also called pipe) structure

e.g. the UNIX command:

grep | sort | count myfile.txt

will produce a wordcount in the text document called myfile.txt.

There are three commands in this sequence, and they work as follows: (a) grep is command to read the text file and create an intermediate file with one word on a line; (b) sort command will sort that intermediate file, and produce an alphabetically sorted list of words in that set; (c) the count command will work on that sorted list, to produce the number of occurrences of each word, and display the results to the user in a "word, frequency" pair format.

For example: Suppose myfile.txt contains the following text:

Myfile: *We are going to a picnic near our house. Many of our friends are coming. You are welcome to join us. We will have fun.*

The outputs of Grep, Sort and Wordcount will as shown below.

Grep		**Sort**		**WordCount**	
We		a		a	1
are		are		are	3
going		are		coming	1
to		are		friends	1
a		coming		fun	1
picnic		friends		going	1

near		fun		have	1
our		going		house	1
house		have		join	1
Many		house		many	1
of		join		near	1
our		many		of	1
friends		near		our	2
are		of		picnic	1
coming		our		to	2
You		our		us	1
are		picnic		we	2
welcome		to		welcome	1
to		to		will	1
join		us		you	1
us		We			
we		we			
will		welcome			
have		will			
fun		you			

If the file is very large, then it will be take the computer a long time to process it. Parallel processing can help here.

MapReduce speeds up the computation by reading and processing small chunks of file, by different computers in parallel. Thus if a file can be broken down into 100 small chunks,

each chunk can be processed at a separate computer in parallel. The total time taken to process the file could be 1/100 of the time taken otherwise. However, now the results of the computation on small chunks are residing in a 100 different places. These large number of partial results need to be combined to produce a composite result. The results of the outputs from various chunks will be combined by another program called the Reduce program.

The Map step will distribute the full job into smaller tasks that can be done on separate computers each using only a part of the data set. The result of the Map step will be considered as intermediate results. The Reduce step will read the intermediate results, and will combine all of them and produce the final result. The programmer needs to specifies the functional logic for both the map and reduce steps. The sorting, between the Map and Reduce steps, does not need to be specified and is automatically taken care of the MapReduce system as a standard service provided to every job. The sorting of the data requires a field to sort on. Thus the intermediate results need to have some kind of a key field, and a set of associated non-key attribute(s) for that key.

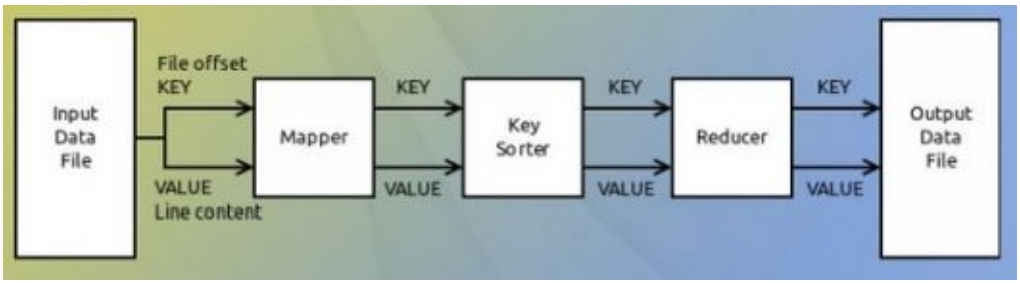


Figure 5-0-1: MapReduce Architecture

In practice, to manage the variety of data structures stored in the file system, data is stored as one key and one non-key attribute. Thus the data is represented as a key-value pair. The intermediate results, and the final results all will also be in key-pair format. Thus a key requirement for the use of MapReduce parallel processing system is that the input data and output data must both be represented in key-values formats.

Map step reads data in key-value pair format. The programmer decide what should be the characteristics of the key and value fields. The Map step produces results in key-value pair format. However, the characteristics of the keys produced by the Map step, i.e. the intermediate results, need not be same keys at the input data. So, those can be called key2-value2 pairs.

The Reduce step reads the key2-value2 pairs, the intermediate results produced by the Map step. Reduce step will produce an output using the same keys that it read. Only the values associated with those keys will change though as a result of processing. Thus it can

be labeled as key2-value3 format.

Suppose the text in the myfile.txt can be split into 4 approximately equal segments. It could be done with each sentence as a separate piece of text. The four segments will look as following:

Segment1: We are going to a picnic near our house.

Segment2: Many of our friends are coming.

Segment3: You are welcome to join us.

Segment4: We will have fun.

Thus the input to the 4 processors in the Map Step will be in key-value pair format. The first column is the key, which is the entire sentence in this case. The second column is the value, which in this application is the frequency of the sentence.

We are going to a picnic near our house.	1
Many of our friends are coming.	1
You are welcome to join us.	1
We will have fun.	1

This task can be done in parallel by four processors. Each of this segment will be task for a different processor. Thus each task will produce a file of words, with a count of 1. There will be four intermediate files, in <key,value> pair format, shown below.

Key2	**Value2**		**Key2**	**Value2**		**Key2**	**Value2**		**Key2**	**Value2**
we	1		many	1		you	1		we	1
are	1		of	1		are	1		will	1
going	1		our	1		welcome	1		have	1

to	1		friends	1		to	1		fun	1
a	1		are	1		join	1			
picnic	1		coming	1		us	1			
near	1									
our	1									
house	1									

The sort process inherent within MapReduce will sort each of the intermediate files, and produce the following sorted key-pair values:

Key2	**Value2**		**Key**	**Value2**		**Key**	**Value2**		**Key**	**Value2**
a	1		are	1		are	1		fun	1
are	1		coming	1		join	1		have	1
going	1		friends	1		to	1		we	1
house	1		many	1		us	1		will	1
near	1		of	1		welcome	1			
our	1		our	1		you	1			
picnic	1									
to	1									
we	1									

The Reduce function will read the sorted intermediate files, and combine the counts for all the unique words, to produce the following output. The keys remain the same as in the intermediate results. However, the values change as counts from each of the intermediate files are added up for each key. For example, the count for the word ‘are’ goes up to 3.

Key2	Value3

a	1
are	3
coming	1
friends	1
fun	1
going	1
have	1
house	1
join	1
many	1
near	1
of	1
our	2
picnic	1
to	2
us	1
we	2
welcome	1
will	1
you	1

This output will be identical to that produced by the UNIX sequence earlier.

MapReduce programming

A data processing problem needs to be transformed into the MapReduce model. The first step is to visualize the processing plan into a map and a reduce step. When the processing gets more complex, this complexity can be generally manifested in having more MapReduce jobs, or more complex map and reduce jobs. Having more but simpler MapReduce jobs leads to more easily maintainable mapper and reducer programs.

MapReduce Data Types and Formats

MapReduce has a simple model of data processing: inputs and outputs for the map and reduce functions are key-value pairs. The map and reduce functions in Hadoop MapReduce have the following general form:

map: (K1, V1) → list (K2, V2)

reduce: (K2, list(V2)) → list (K3, V3)

In general, the map input key and value types (K1 and V1) are different from the map output types (K2 and V2). However, the reduce input must have the same types as the map output, although the reduce output types may be different again (K3 and V3). Since Mapper and Reducer are separate classes, the type parameters have different scopes,

Hadoop can process many different types of data formats, from flat text files to databases. An input split is a chunk of the input that is processed by a single map. Each map processes a single split. Each split is divided into records, and the map processes each record—a key-value pair—in turn. Splits and records are logical: and may map to a full file, a part of a file, or a collection of files. In a database context, a split might correspond to a range of rows from a table and a record to a row in that range

Writing MapReduce Programming

Start by writing pseudocode for the map and reduce functions. The program code for both the map and the reduce function can then be written in Java or other languages. In Java, the map function is represented by the generic Mapper class. It uses four parameters: input key, input value, output key, output value. This class uses an abstract map() method. This method received the input key and input value. It would normally produce and output key and output value. For more complex problems, it is better to use a higher-level language than MapReduce, such as Pig, Hive, Cascading, Crunch, or Spark.

A mapper commonly performs input format parsing, projection (selecting the relevant fields), and filtering (selecting the records of interest). The reducer typically combines

(adds or averages) those values.

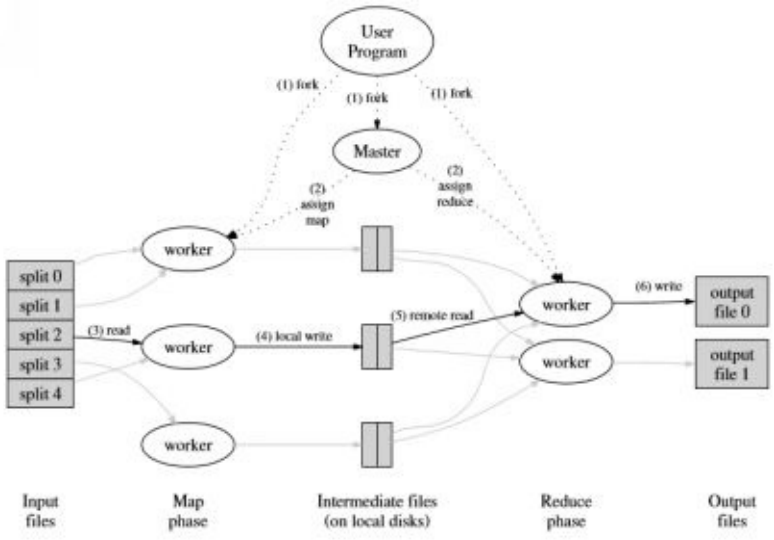


Figure 5-0-2: MapReduce program Flow

Here below is the step-by-step logic Imagine that we want to do a word count of all unique words in a text.

1. The big document is split into many segments. The map step is run on each segment of data. The output will be a set of key,value pairs. In this case, the key will be a word in the document.
2. The system will gather the key,value pair outputs from all the mappers, and will sort them by key. The sorted list itself may then be split into a few segments.
3. A Reducer task will read the sorted list and produce a combined list of word counts.

Here is the Java code for wordcount:.

```
map(String key, String value):
        for each word w in value:
        EmitIntermediate(w, “1”);
reduce(String key, Iterator values):
        int result = 0;
        for each v in values:
                result += ParseInt(v);
        Emit(AsString(result));
```

Testing MapReduce Programs

Mapper programs running on a cluster can be complicated to debug. The time-honored

way of debugging programs is via print statements. However, with the programs eventually running on tens or thousands of nodes, it is best to debug the programs in stages. Therefore, run the program using small sample datasets to ensure that the program is working correctly. Expand the unit tests to cover larger dataset and run it on a cluster. Ensure that the mapper or reducer can handle the inputs correctly. Running against the full dataset is likely to expose some more issues, which should be fixed, by altering your mapper or reducer to handle the new cases. After the program is working, the program may be tuned to make the entire MapReduce job run faster.

It may be desirable to split the logic into many simple mappers and chaining them into a single mapper using a facility (the ChainMapper library class) built into Hadoop. It can run a chain of mappers, followed by a reducer and another chain of mappers, in a single MapReduce job.

MapReduce Jobs Execution

A MapReduce job is specified by the Map program and the Reduce program, along with the data sets associated with that job. There is another master program that resides and runs endlessly on the NameNode. It is called the Job tracker, and it tracks the progress of the MapReduce jobs from beginning to the completion. Hadoop divides the job into two tasks: map tasks and reduce tasks. Hadoop moves the Map and Reduce computation logic to each DataNode that is hosting a part of the data. The communication between the nodes is accomplished using YARN, Hadoop's native resource manager.

The master machine (NameNode) is completely aware of the data stored on each of the worker machines (DataNodes). It schedules the map or reduce jobs to task trackers with full awareness of the data location. For example: if node A contains data (x,y,z) and node B contains data (a,b,c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the data traffic and prevents choking of the network.

Each DataNode has a master program called the Job tracker. This program monitors the execution of every task assigned to it by the NameNode. When the task is completed, the Tasktracker sends a completion message to the JobTracker program on the

The jobs and tasks work in a master-slave mode.

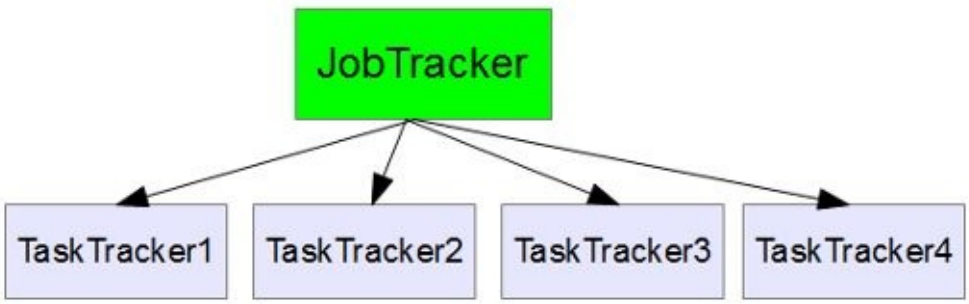


Figure 5-0-3: Hierarchical Monitoring Architecture

When there is more than one job in a MapReduce workflow, it is necessary they be executed in the right order. For a linear chain of jobs it might be easy. For a more complex directed acyclic graph (DAG) of jobs, there are libraries that can help orchestrate your workflow. Or one can use Apache Oozie, a system for running workflows of dependent jobs.

Oozie consists of two main parts: a workflow engine that stores and runs workflows composed of different types of Hadoop jobs (MapReduce, Pig, Hive, and so on), and a coordinator engine that runs workflow jobs based on predefined schedules and data

availability. Oozie has been designed to scale, and it can manage the timely execution of thousands of workflows in a Hadoop cluster.

The dataset for the MapReduce job is divided into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split. The tasks are scheduled using YARN and run on nodes in the cluster. YARN ensures that if a task fails or inordinately delayed, it will be automatically scheduled to run on a different node. The outputs of the map jobs are fed as input to the reduce job. That logic is also propagated to the node(s) that will do the reduce jobs. To save on bandwidth, Hadoop allows the use of a combiner function on the map output. Then the combiner function's output forms the input to the reduce function.

How MapReduce Works

A MapReduce job can be executed with a single method call: submit() on a Job object. When the resource manager receives a call to its submitApplication() method, it hands off the request to the YARN scheduler. The scheduler allocates a container, and the resource manager then launches the application master's process. The application master for MapReduce jobs is a Java application whose main class is MRAppMaster. It initializes the job by creating a number of bookkeeping objects to keep track of the job's progress. It retrieves the input splits computed in the client from the shared filesystem. It then creates a map task object for each split, as well as a number of reduce task objects determined by the mapreduce.job.reduces property (set by the setNumReduceTasks() method on Job). Tasks are given IDs at this point. The application master must decide how to run the tasks that make up the MapReduce job. The application master requests containers for all the map and reduce tasks in the job from the resource manager. Once a task has been assigned resources for a container on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager. The task is executed by a Java application whose main class is YarnChild.

Managing Failures

There can be failures at the level of the entire job or particular tasks. The entire application master itself could fail.

Task failure usually happens when the user code in the map or reduce task throws a runtime exception. If this happens, the task JVM reports the error to its parent application master, where it is logged into error logs. The application master will then reschedule execution of the task on another data node.

The entire job, i.e. MapReduce application master application running on YARN, too can fail. In that case, it is started again, subject to a maximum number which is a user-set configuration parameter.

If a datanode manager fails by crashing or running very slowly, it will stop sending heartbeats to the resource manager (or send them very infrequently). The resource manager will then remove it from its pool of nodes to schedule containers on. Any task or application master running on the failed node manager will be recovered using error logs, and started on other nodes.

Resource Manager YARN can also fail, and it has more severe consequences for the entire cluster. Therefore, typically, there will be a hot-standby for YARN. If the active resource manager fails, then the standby can take over without a significant interruption to the client. The new resource manager can read the application information from the state store, and then restart the application that were running on the cluster.

Shuffle and Sort

MapReduce guarantees that the input to every reducer is sorted by key. The process by which the system performs the sort—and transfers the map outputs to the reducers as inputs—is known as the shuffle.

When the map function starts producing output, it is not directly written to disk. The takes advantage of buffering writes in memory and doing some presorting for efficiency reasons. Each map task has a circular memory buffer that it writes the output to. Before it writes to disk, the thread first divides the data into partitions corresponding to the reducers that they will ultimately be sent to. Within each partition, the background thread performs an in-memory sort by key. If there is a combiner function, it is run on the output of the sort so that there is less data to transfer to the reducer.

The reduce task needs the map output for its particular partition from several map tasks across the cluster. The map tasks may finish at different times, so the reduce task starts reading their outputs as soon as each completes. When all the map outputs have been read, the reduce task merges the map outputs, maintaining their sort ordering. The reduce function is invoked for each key in the sorted output. The output of this phase is written directly to the output filesystem such as HDFS.

Progress and Status Updates

MapReduce jobs are long-running batch jobs, taking a long time to run. It is important for

the user to get feedback on how the job's progress. A job and each of its tasks have a status value (e.g., running, successfully completed, failed), the progress of maps and reduces, the values of the job's counters. These values are constantly communicated back to the client. When the application master receives a notification that the last task for a job is complete, it changes the status for the job to "successful." Job statistics and counters are communicated to the user.

Hadoop comes with a native web-based GUI for tracking the MapReduce jobs. It displays useful information about a job's progress such as how many tasks have been completed, and which ones are still being executed. Once the job is completed, one can view the job statistics and logs.

Hadoop Streaming

Hadoop Streaming uses standard Unix streams as the interface between Hadoop and user program. Streaming is an ideal application for text processing. Map input data is passed over standard input to your map function, which processes it line by line and writes lines to standard output. A map output key-value pair is written as a single tab-delimited line. Input to the reduce function is in the same format—a tab-separated key-value pair—passed over standard input. The reduce function reads lines from standard input, which the framework guarantees are sorted by key, and writes its results to standard output.

Conclusion

MapReduce is the first popular parallel programming framework for Big Data. It works well for applications where the data can be large, and divisible into separate sets, and represented in <key,value> pair format. The application logic is divided into two parts: a Map program and a Reduce Program. Each of these programs can be run in parallel by several machines.

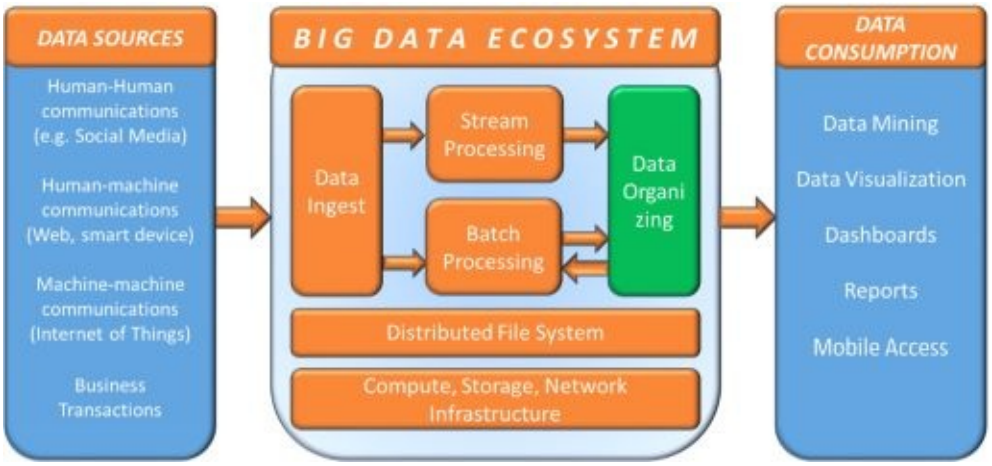
Review Questions

Q1: What is MapReduce? What are its benefits?

Q2: What is the key-value pair format? How is it different from other data structures? What are its benefits? And limitations.

Chapter 6 – NoSQL databases

A NoSQL database is a clever way to cost-effectively organize large amounts of heterogeneous data for efficient access and updates. The ideal NoSQL database is completely aligned with the nature of the problems being solved, and is superfast in that task. This is achieved by releasing and relaxing many of the integrity and redundancy constraints of storing data in relational databases, and storing data in many innovative formats as aligned with business need. The diverse NoSQL databases will ultimately collective evolve into a holistic set of efficient and elegant data structures at the heart of a cosmic computer of infinite organization capacity.



Introduction

Relational data management systems (RDBMS) are a powerful and universally used database technology by almost all enterprises. Relational databases are structured and optimized to ensure accuracy and consistency of data, while also eliminating any redundancy of data. These databases are stored on the largest and most reliable of computers to ensure that the data is always available at a granular level and at a high speed.

Big data is however a much larger and unpredictable stream of data. Relational databases are inadequate for this task, and will also be very expensive for such large data volumes. Managing the costs and speed of managing such large and heterogeneous data streams requires relaxing many of the strict rules and requirements of relational data. Depending upon which constraint(s) are relaxed, a different kind of database structure will emerge. These are called NoSQL databases, to differentiate them from relational databases that use Structured Query Language (SQL) as the primary means to manipulate data.

NoSQL databases are next-generation databases that are non-relational in their design. The name NoSQL is meant to differentiate it from antiquated, 'PRE-relational' databases. Today, almost every organization that needs to gather customer feedback and sentiments to improve their business, will use a NoSQL database. NoSQL is useful when an enterprise needs to access, analyze and utilize massive amounts of either structured or unstructured data or data that's stored remotely in any virtual server across the globe.

The constraints of a relational database are relaxed in many ways. For example, relational databases require that any data element could be randomly accessed and its value could be updated in that same physical location. However, the simple physics of storage says that it is simpler and faster to read or write sequential blocks of data on a disk. Therefore, NoSQL database files are written once and almost never updated in place. If a new version of a part of the data become available, it would be stored elsewhere by the system. The system would have the intelligence to link the updated data to the old data.

Pig and Hive are two key and popular languages in the Hadoop ecosystem that works well on NoSQL databases. Pig originated at Yahoo while Hive originated at Facebook. Both Pig and Hive can use the same data as an input, and can achieve similar results with queries. Both Pig Latin and Hive commands eventually compile to Map and Reduce jobs. They have a similar goal - to ease the complexity of writing complex java MapReduce programs. Most MapReduce jobs can be implemented easily in Hive or Pig.

For analytical needs, Hive is preferable over Pig. For controlled processing, Pig's scripting design is preferable Hive leads to ease and productivity using its SQL like design and user interface. Pig offers greater control over data flows. JavaMR can be used for more advanced APIs to accomplish things when there is something special needed, such as interacting with a third-party tool, or some special data characteristics.

RDBMS Vs NoSQL

They are different in many ways. First, NoSQL databases, do not support relational schema or the SQL language. The term NoSQL stands mostly for “Not only SQL”. Second, their transaction processing capabilities are fast but weak, and they do not support the ACID (Atomicity, Consistency, Isolation, Durability) properties associated with transaction processing using relational databases. Instead, they are approximately accurate at any point in time, and will be eventually consistent. Third, these databases are also distributed and horizontally scalable to manage web-scale databases using Hadoop clusters of storage. Thus they work well with the write-once, read-many storage mechanism of Hadoop clusters.

Feature	RDBMS	NoSQL
Applications	Mostly centralized Applications (e.g. ERP)	Mostly designed for the decentralized applications (e.g. Web, mobile, sensors)
Availability	Moderate to high	Continuous availability to receive and serve data
Velocity	Moderate velocity of data	High velocity of data (devices, sensors, social media, etc.). Low latency of access.
Data Volume	Moderate size; archived after for a certain period	Huge volume of data, stored mostly for a long time or forever; Linearly scalable DB.
Data Sources	Data arrives from one or few, mostly predictable sources	Data arrives from multiple locations and are of unpredictable nature
Data type	Data are mostly structured	Structured or unstructured data
Data Access	Primary concern is reading the data	Concern is both read and write
Technology	Standardized relational schemas; SQL language	Many designs with many implementations of data structures and access languages
Cost	Expensive; commercial	Low; open-source software

Types of NoSQL Databases

The variety of big data means that file size and types will vary enormously. There are specialized databases to suit different purposes.

1. **Document Databases**: Storing a 10GB video movie file as a single object could be speeded up by sequentially storing the data in contiguous blocks of physical storage. An index could store the identifying information about the movie, and the address of the starting block. The rest of storage details could be handled by the system. This storage format would be a called document store format. The index would contain the name of the movie, and the value is the entire video file, characterized by the first block of storage. Document databases are generally useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications. We would avoid using document databases for systems that need complex transactions spanning multiple operations or queries against varying aggregate structures.
2. **Key-Value Pair Databases**: There could be a collection of many data elements such as a collection of text messages which could also fit into a single physical block of storage. Each text message is a unique object. This data would need to be queried often. That collection of messages could also be stored in a key-value pair format, by combining the identifier of the message and the content of the message. Key-value databases are useful for storing session information, user profiles, preferences, and shopping cart data. Key-value databases don't work so well when we need to query by non-key fields or on multiple key fields at the same time.
3. **Graph Databases**: Geographic map data that is stored in set of relationships or links between points. Graph databases are very well suited to problem spaces where we have connected data, such as social networks, spatial data, routing information, and recommendation engines.
4. **Columnar Databases**: Some kind of databases are needed to speed up some oft-sought queries from very large data sets. Suppose there is an extremely large data warehouse of web log access data, which is rolled up by the number of web access by the hour. This needs to be queried, or summarized often, involving only some of the data fields from the database. Thus the query could be speeded up by creating a database structure that included only the relevant columns of the dataset, along with the key identifying information. This is called a columnar database format, and is useful for content management systems, blogging platforms, maintaining counters, expiring usage, heavy write volume such as log

aggregation. Column family databases for systems well when the query patterns have stabilized.

The choice of NoSQL database depends on the system requirements. There are at least 200 implementations of NoSQL databases of these four types. Visit nosql-database.org for more.

Despite the name, a NoSQL database does not necessarily prohibit structured query language (like MySQL). While some of the NoSQL systems are entirely non-relational, others just avoid some selected functionality of RDMS such as fixed table schemas and join operations. For NoSQL systems, instead of using tables, the data can be organized the data in key/ value pair format, and then SQL can be used.

The first popular NoSQL database was HBase, which is a part of the Hadoop family. The most popular NoSQL database used today is Apache Cassandra, which was developed and owned by Facebook till it was released as open source in 2008. Other NoSQL database systems are SimpleDB, Google's BigTable, MemcacheDB, Oracle NoSQL, Voldemort, etc.

Architecture of NoSQL

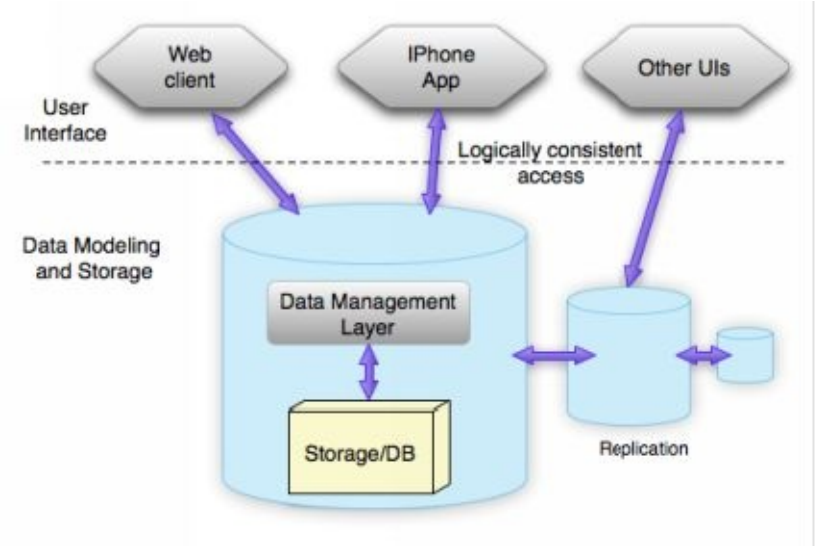


Figure 6-0-1: NoSQL Databases Architecture

One of the key concepts underlying the NoSQL databases is that database management has moved to a two-layer architecture; separating the concerns of data modeling and data storage. The data storage layer focuses on the task of high-performance scalable data storage for the task at hand. The data management layer a variety of database formats, and allows for low-level access to that data through specialized languages that are more appropriate for the job, rather than being constrained by using the standard SQL format.

NoSQL databases maps the data in the key/ value pairs and saves the data in the storage unit. There is no storage of data in a centralized tabular form, so the database is highly scalable. The data could be of different forms, and coming from different sources, and they can all be stored in similar key/value pair formats.

There are a variety of NoSQL architectures. Some popular NoSQL databases like MongoDB are designed in a master/ slave model like many RDBMS. But other popular NoSQL databases like Cassandra are designed in a master-less fashion where all the nodes in the clusters are the same. So, it is the architecture of the NoSQL database system that determines the benefits of distributed and scalable system emerges like continuous availability, distributed access, high speed, and so on.

NoSQL databases provide developers lot of options to choose from and fine tune the system to their specific requirements. Understanding the requirements of how the data is going to be consumed by the system, questions such as is it read heavy vs write heavy, is there a need to query data with random query parameters, will the system be able handle inconsistent data.

CAP theorem

Data is expected to be accurate and available. In a distributed environment, accuracy depends upon the consistency of data. A system is considered Consistent if all replicas of copy contain the same value. The system is considered Available, if the data I is available at all points in time. It is also desirable for the data to be consistent and available even when a network failure renders the database partitioned into two or more islands. A system is considered partition tolerant if processing can continue in both partitions in the case of a network failure. In practice it is hard to achieve all three.

The choice between Consistency and Availability remains the unavoidable reality for distributed data stores. CAP theorem states that in any distributed system one can choose only two out of the three (Consistency, Availability and Partition Tolerance). The third will be determined by those choices.

NoSQL databases can be tuned to suit one's choice of high consistency or availability. For example, for a NoSQL database, there are essentially three parameters:

- N = replication factor, i.e. the number of replicas created for each piece of data
- R = Minimum number of nodes that should respond to a read request for it to be considered successful
- W = Minimum number of nodes that should respond to a write request before its considered successful.

Setting the values of R and W very high (R=N, and W=N) will make the system more consistent. However, it will be slow to report Consistency, and thus Availability will be low. On the other end, setting R and W to be very low (such as R=1 and W=1), would make the cluster highly available, as even a single successful read (or write) would let the cluster to report success. However, consistency of data on the cluster will be low since many of the may not have yet received the latest copy of the data.

If a network gets partitioned because of a network failure, then one has to trade off availability versus consistency. NoSQL database users often choose availability and partition tolerance over strong consistency. They argue that short periods of application misbehavior are less problematic than short periods of unavailability.

Consistency is more expensive in terms of throughput or latency, than is Availability. However, HDFS chooses consistency – as three failed datanodes can potentially render a

file’s blocks completely unavailable.

Popular NoSQL Databases

We cover two of the more popular offerings.

HBase

Apache HBase is a column-oriented, non-relational, distributed database system that runs on top of HDFS. An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key; all access to HBase tables is done using the Primary Key. An HBase column represents an attribute of an object. For example, if the table is storing diagnostic logs from web servers, each row will be a log record. Each column in that table will represent an attribute such as the date/time of the record, or the server name. HBase permits many attributes to be grouped together into a column family, so that all elements of a column family are all stored as essentially a composite attribute.

Columnar databases are different from a relational database in terms of how the data is stored. In the relational database, all the columns / attributes of a given row are stored together. With HBase you must predefine the table schema and specify the column families. All rows of a column family will stored sequentially. However, it's very flexible in that new columns can be added to families at any time, making the schema flexible and therefore able to adapt to changing application requirements.

Architecture Overview

HBase is built on master-slave concept. In HBase a master node manages the cluster, while the worker nodes (called region servers) store portions of the tables and perform the work on the data. HBase is designed after Google Bigtable, and offers similar capabilities on top of Hadoop and HDFS. It does consistent reads and writes. It does automatic and configurable sharding of tables. A shard is a segment of the database.

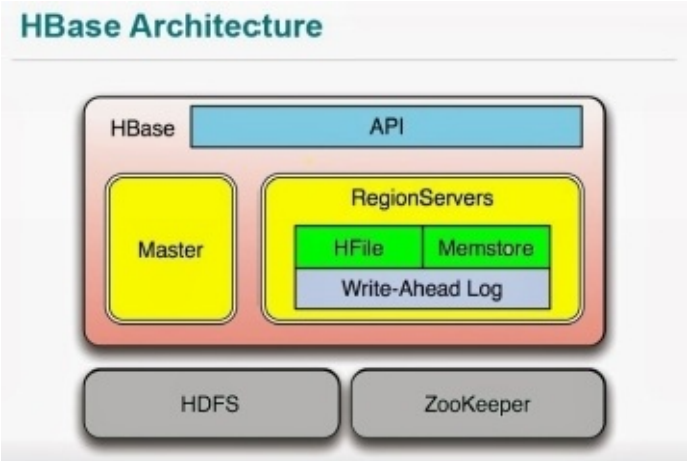


Figure 6-0-2: HBASE Architecture

Physically, HBase is composed of three types of servers in a master slave type of architecture.

(a) The NameNode maintains metadata information for all the physical data blocks that comprise the files.

(b) Region servers serve data for reads and writes.

(c) The Hadoop DataNode stores the data that the Region Server is managing.

HBase Tables are divided horizontally by row key range into “Regions.” A region contains all rows in the table between the region’s start key and end key. Region assignment, DDL (create, delete tables) operations are handled by the HBase Master process. Zookeeper, which is part of HDFS, maintains a live cluster state. There is an automatic failover support between RegionServers. All HBase data is stored in HDFS files. Region Servers are collocated with the HDFS DataNodes, which enable data locality (putting the data close to where it is needed) for the data served by the RegionServers. HBase data is local when it is written, but when a region is moved, it is not local until compaction.

Each Region Server creates an ephemeral node. The HMaster monitors these nodes to discover available region servers, and it also monitors these nodes for server failures.

A master is responsible for coordinating the region servers, including assigning regions on startup, load balancing of recovery among regions, and monitoring their health. It is also the interface for creating, deleting, updating tables

Reading and Writing Data

There is a special HBase Catalog table called the META table, which holds the location of the regions in the cluster. ZooKeeper stores the location of the META table.

This is what happens the first time a client reads or writes to HBase:

The client gets the Region server that hosts the META table from ZooKeeper.

The client will query the .META. server to get the region server corresponding to the row key it wants to access. The client caches this information along with the META table location.

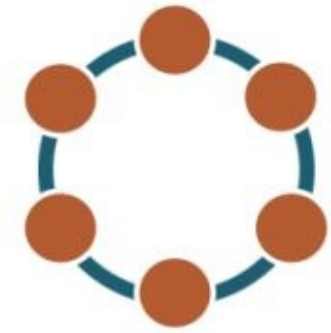
It will get the Row from the corresponding Region Server.

For future reads, the client uses the cache to retrieve the META location and previously read row keys. Over time, it does not need to query the META table, unless there is a miss because a region has moved; then it will re-query and update the cache.

Cassandra

Apache Cassandra is a largely scalable open source non-relational database that offers continuous uptime, simplicity and easy data distribution across multiple data centers and cloud. Cassandra was originally developed at Facebook and was open sourced in 2008. It provides many benefits over the traditional relational databases for modern online applications like scalable architecture, continuous availability, high data protection, multi data replications over data centers, data compression, SQL like language and so on.

Architecture Overview

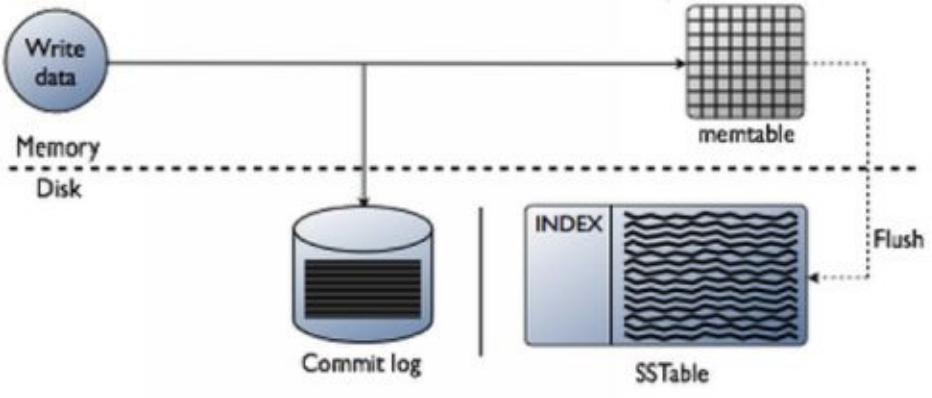


Cassandra architecture provides its ability to scale and provide continuous availability. Rather than using master-slave architecture, it has a master-less “ring” design that is easy to set up and maintain. In Cassandra, all nodes play an equal role, all nodes communicate with one another by a distributed and highly scalable protocol called gossip.

So, the Cassandra scalable architecture provides the capacity of handling large volume of data, and large number of concurrent users or operations occurring at the same time, across multiple data centers, just as easily as a normal operation for the relational databases. To enhance its capacity, one simply needs to add new nodes to an existing cluster without taking down the system and designing from the scratch.

Also the Cassandra architecture means that unlike other master slave systems, it has no single point of failure and thus is capable of offering continuous availability and uptime.

Reading and Writing Data



Data to be written to a Cassandra node is first recorded in an on disk commit log and then it is written to a memory based unit called a “memTable”. When a “memTable” size exceeds a certain set threshold, the data is then written to file on disk called an “SSTable”. Thus, in this way the write operation is fully sequential in nature.with many input output operation occurring at the same time, rather than occurring one at a time over a long period.

For a read operation, Cassandra looks in an in memory data structure called a “Bloom filter” that fetch the probability of a “SSTable” having the required data. The Bloom filter can perform the task very quickly to tell if a file has the needed data or not. If it return true then Cassandra looks for another layer of in memory caches, and then fetches the compressed data on disk. If the answer is false, Cassandra doesn’t bother with reading the “SSTable” and looks for another file to fetch the required data.

Write Syntax:

```
TTransport tr = new TSocket(HOST, PORT);

TFramedTransport tf = new TFramedTransport(tr);
TProtocol protocal = new TBinaryProtocol(tf);
Cassandra.Client client = new Cassandra.Client(protocal);

tf.open();

client.insert(userIDKey, cp,  new Column(“Colume-name”.getBytes(UTF8),  “Colume-data”.getBytes(), clock), CL);
```

Read Syntax:

```
Column col = client.get(userIDKey, colPathName, CL).getColumn();

LOG.debug(“Column name: ” + new String(col.Colume-name, UTF8));

LOG.debug(“Column value: ” + new S tring(col.Colume-data, UTF8));
```

Hive Language

Hive is a declarative SQL-like language for queries. Hive was designed to appeal to a community comfortable with SQL. It is used mainly by data analysts on the server side, for designing reports. It has its own metadata section which can be defined ahead of time, before data is loaded. Hive supports map and reduce transform scripts in the language of the user's choice, which can be embedded within SQL clauses. It is widely used in Facebook by analysts comfortable with SQL, as well as by data miners programming in Python. Hive is best used for traditional data warehousing tasks; it is not designed for online transaction processing.

Hive is best suited for structured data. Hive can be used to query data stored in Hbase, which is a key-value store. Hive's SQL-like structure makes transformation of data to and from RDBMS is easier. Supporting SQL syntax also makes it easy to integrate with existing BI tools. Hive needs the data to be first imported (or loaded) and after that it can be worked upon. In case of streaming data, one would have to keep filling buckets (or files), and then Hive can be used to process each filled bucket, while using other buckets to keep storing the newly arriving data.

Hive data Columns are mapped to tables in HDFS. This mapping is stored in Metadata. All HQL queries are converted to MapReduce jobs. A table can have one more partition keys. There are usual SQL data types, and Arrays and Maps and Structs to represent more complex types of data. There are user defined functions for mapping, aggregating

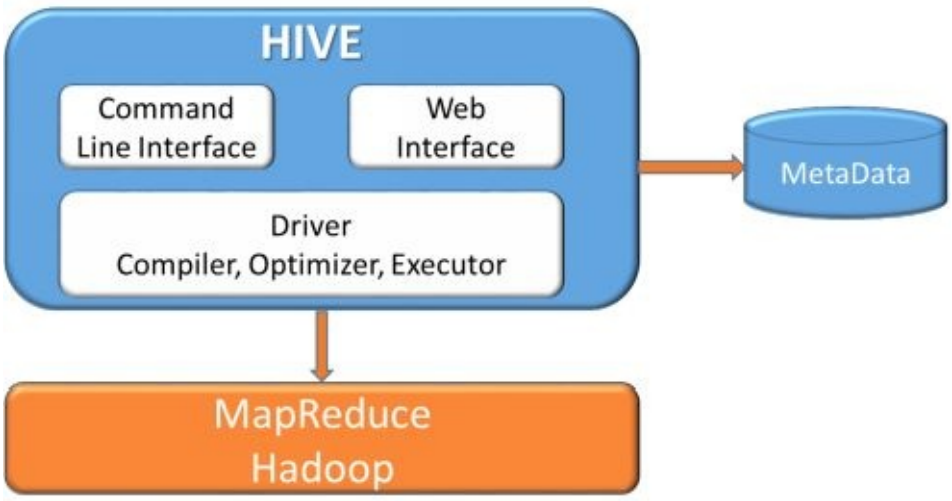


Figure 6-3: Hive Architecture

HIVE Language Capabilities

Hive's SQL provides almost all basic SQL operations. These operations work on tables and or partitions. These operations are: SELECT, FROM, WHERE, JOIN, GROUP BY,

ORDER BY. It also allows the results to be stored in another table, or in a HDFS file.

Hive	Relational Database
SQL	SQL
Analytics	OLTP or analytics
Batch only	Real-time or batch
No transactions	Transactions
No INSERT or UPDATE Adding through partitions	Random INSERT or UPDATE
Distributed processing - 100s of nodes	Depends on the system - If available, < 100
Achieve high performance on commodity hardware	Achieve high performance on proprietary hardware
Low cost for huge amounts of storage	Expensive, limited compared to Hadoop based solutions

The statement to create a page_view table 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;
```

Here is a script for loading data into this file.

```
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';
```

The table created above can be stored in HDFS as a TextFile or as a SequenceFile.

An INSERT query on this table will look like:

```
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';
```

Pig Language

Pig is a high-level procedural language. It is used mainly for programming. It helps to create a step-by-step flow of data to do processing. It operates mostly on the client side of the cluster. Pig Latin follows a procedure programming model and more natural to use to build a data pipeline, such as ETL job. It gives full control over how the data flows through the pipeline, when to checkpoint the data in pipeline, and it support DAGs in pipeline such as split, and gives more control over optimization. Pig works well with unstructured data. For complex operations such as analyzing matrices, or search for patterns in unstructured data, Pig will give greater control and options.

Pig allows one to load data and user code at any point in the pipeline. This can be important for ingesting streaming data from satellites or instruments. Pig also uses lazy evaluation. Pig is faster in the data import but slower in actual execution than an RDBMS friendly language like Hive. Pig is well suited to parallelization and so it is better suited for very large datasets throughput (amount of data processed) is more important than latency (speed of response).

Pig is SQL-like, but differs to a great extent. It does not have a dedicated metadata section; the schema will have to be defined in the program itself. It is. Pig can be easier for someone who had not earlier experience with SQL.

Conclusion

NoSQL databases emerged in response to the limitations of relational databases in handling the sheer volume, nature and growth of data. NoSQL databases have the functionality like MapReduce. NoSQL database is proving to be a viable solution to the enterprise data needs and continue to do so. There are four types of NoSQL databases: columnar, Key-pair, document, and graphical databases. Cassandra and HBase are among the most popular NOSQL databases. Hive is an SQL-type language to access data from NoSQL databases. Pig is a procedural high-language that gives greater control over data flows.

Review Questions

Q1: What is a NoSQL database? What are the different types of it?

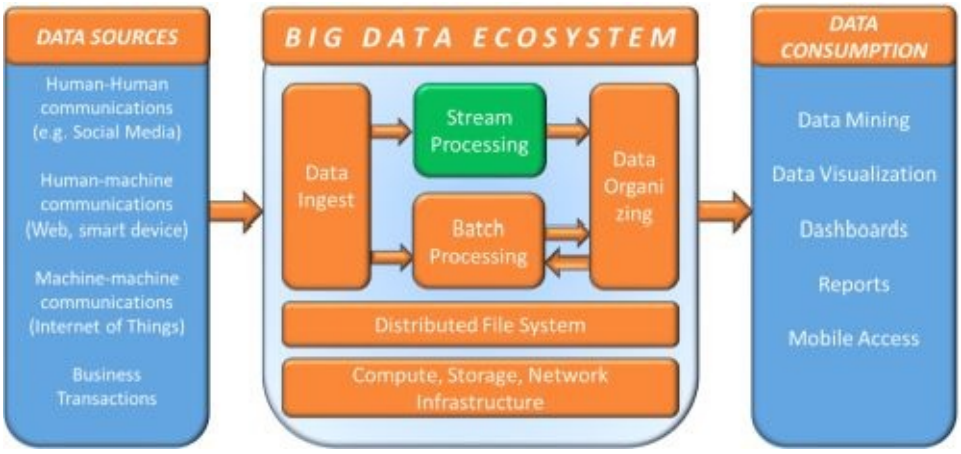
Q2: How does a NoSQL database leverage the power of MapReduce?

Q3: what are the kinds of NoSQL databases? What are the advantages of each?

Q3: What are the similarities and differences between Hive and Pig?

Chapter 7 – Stream Processing with Spark

A stream processing system is a clever way to process large quantities of data from a vast set of extremely fast incoming data streams. The ideal stream processing engine will capture and report in real time the essence of all data streams, no matter the speed or size of number. This is achieved by using innovative algorithms and filters that relax many computational accuracy requirements, to compute simple approximate metrics in real time. Stream processing engine aligns with the infinite dynamism of the flow of nature.

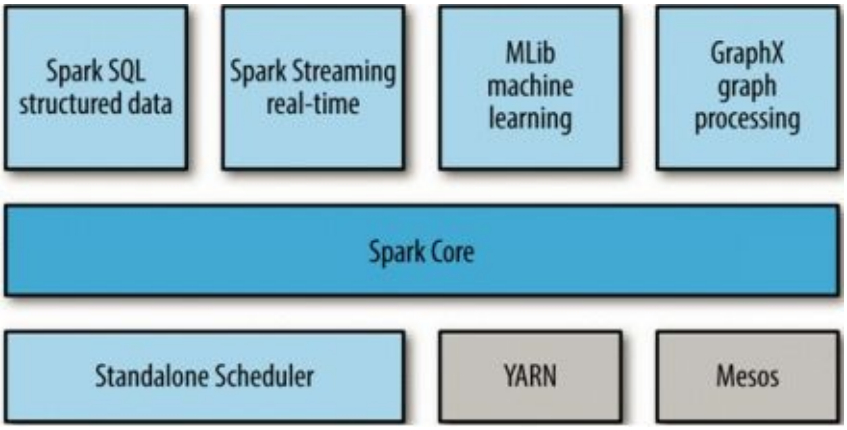


Introduction

Apache Spark is an integrated, fast, in-memory, general-purpose engine for large-scale data processing. Spark is ideal for iterative and interactive processing tasks on large data sets and streams. Spark achieves 10-100x performance over Hadoop by operating with an in-memory construct called ‘Resilient Distributed Datasets’, which help avoid the latencies involved in disk reads and writes. While Spark is compatible with Hadoop file systems and tools, a large scale adoption of Spark and its built-in libraries (for Machine Learning, Graph Processing, Stream processing, SQL) will deliver seamless fast data processing along with high programmer productivity. Spark has become a more efficient and productive alternative for Hadoop ecosystem, and is increasing being used in industry.

Apache Spark was originally developed in 2009 in UC Berkeley’s AMPLab, and open sourced in 2010 as an Apache project. It can process data from a variety of data repositories, including the Hadoop Distributed File System (HDFS), and NoSQL databases such as HBase and Cassandra. Spark supports in-memory processing to boost the performance of big data analytics applications, but it can also do conventional disk-based processing when data sets are too large to fit into the available system memory. Spark gives us a comprehensive, unified framework to manage big data processing requirements with a variety of data sets that are diverse in nature (text data, graph data etc) as well as the source of data (batch v. real-time streaming data). Spark enables applications in Hadoop clusters to run up to 100 times faster in memory and 10 times faster even when running on disk. Spark is an alternative to Hadoop MapReduce rather than a replacement for Hadoop. It provides a comprehensive and unified solution to manage different big data use cases and requirements.

Spark Architecture



The core Spark engine functions partly as an application programming interface (API) layer and underpins a set of related tools for managing and analyzing data, including a SQL query engine, a library of machine learning algorithms, a graph processing system and streaming data processing software. Spark allows programmers to develop complex, multi-step data pipelines using directed acyclic graph (DAG) pattern. It also supports in-memory data sharing across DAGs, so that different jobs can work with the same data. Spark runs on top of existing Hadoop Distributed File System (HDFS) infrastructure to provide enhanced and additional functionality. It provides support for deploying Spark applications in an existing Hadoop v1 cluster (with SIMR – Spark-Inside-MapReduce) or Hadoop v2 YARN cluster or even Apache Mesos.

Next we will introduce the two importance features in spark: RDDs and DAG.

Resilient Distributed Datasets (RDD)

RDD, Resilient Distributed Datasets, is a distributed memory distribution. They are motivated by two types of applications that current computing frameworks handle inefficiently: iterative algorithms and interactive data mining tools. In both cases, keeping data in memory can improve performance by an order of magnitude.

RDDs are Immutable and partitioned collection of records, which can only be created by coarse grained operations such as map, filter, group by etc. By coarse grained operations, it means that the operations are applied on all elements in a dataset. RDDs can only be created by reading data from a stable storage such as HDFS or by transformations on existing RDDs.

Once data is read into an RDD object in Spark, a variety of operations can be performed by calling abstract Spark APIs. The two major types of operation available are transformations and actions. Transformations return a new, modified RDD based on the original. Several transformations are available through the Spark API, including map(),

filter(), sample(), and union(). Actions return a value based on some computation being performed on an RDD. Some examples of actions supported by the Spark API include reduce(), count(), first(), and foreach().

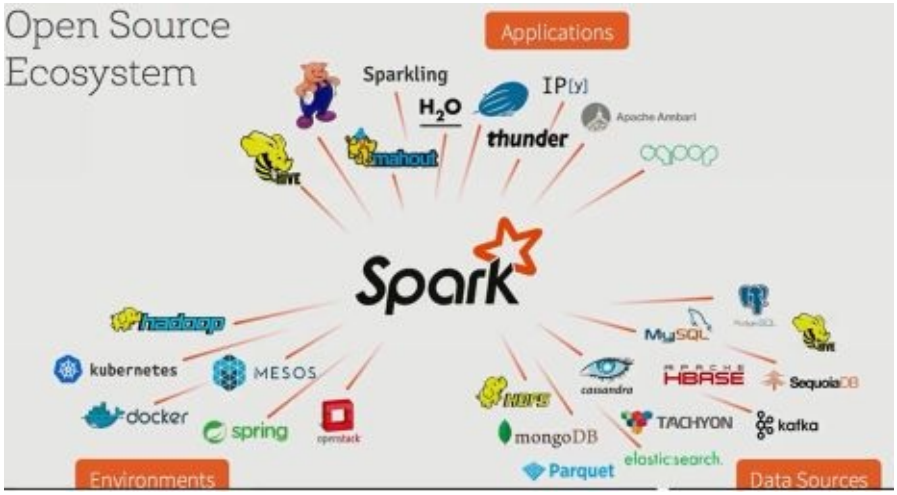
Directed Acyclic Graph (DAG)

DAG refers a directed acyclic graph. This approach is an important feature for real-time nig Data platforms. Those tools, including Storm, Spark, and Tez, offer amazing new capabilities for building highly interactive, real-time computing systems to power your real-time BI, predictive analytics, real-time marketing and other critical systems.

DAG Scheduler is the scheduling layer of Apache Spark that implements stage-oriented scheduling, i.e. after an RDD action has been called it becomes a job that is then transformed into a set of stages that are submitted as TaskSets for execution. In general, DAG Scheduler does three things in Spark: Computes an execution DAG, i.e. DAG of stages, for a job; Determines the preferred locations to run each task on; Handles failures due to shuffle output files being lost.

Spark Ecosystem

Spark is an integrated stack of tools responsible for scheduling, distributing, and monitoring applications consisting of many computational tasks across many worker machines, or a computing cluster. Spark is written primarily in Scala, but includes code from Python, Java, R, and other languages. Spark comes with a set of intergrated tools that reduce learning time and deliver higher user productivity. Spark ecosystem includes Mesos resource manager, and other tools.



Spark has already overtaken Hadoop in general because of benefits it provides in terms of faster execution in iterative processing algorithms.

Spark for big data processing

Spark support big data mining through relevant libraries including MLlib, GraphX and SparkR. And through Spark SQL language and Streaming library.

MLlib

MLlib is Spark's machine learning library. It consists of basic machine learning algorithms such as classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs. At the same time, we care about algorithmic performance. Spark excels at iterative computation, enabling MLlib to run fast. So MLlib also contains high-quality algorithms that leverage iteration, and can yield better results than the one-pass approximations sometimes used on MapReduce. In addition, Spark MLlib is easy to use and it can support scala, Java, Python, and SparkR.

For example, Decision trees is a popular data classification technique, Spark MLlib can support decision trees for binary and multiclass classification, using both continuous and categorical features. The implementation partitions data by rows, allowing distributed training with millions of instances.

Functions in Decision Trees

class : public static DecisionTreeModel trainClassifier(…)

Method to train a decision tree model for binary or multiclass classification.

Parameters:

- input - Training dataset: RDD of LabeledPoint. Labels should take values {0, 1, …, numClasses-1}.
- numClassesForClassification - number of classes for classification.
- categoricalFeaturesInfo - Map storing arity of categorical features.
- impurity - Criterion used for information gain calculation. Supported values: "gini" or "entropy"
- maxDepth - Maximum depth of the tree. (suggested value: 4).
- maxBins - maximum number of bins used for splitting features (suggested value: 100).

Returns: DecisionTreeModel that can be used for prediction

Spark GraphX

Efficient processing of large graphs is another important and challenging issue. Many

practical computing problems concern large graphs. For example, google have to run its PageRank on billions of webpages and maybe trillions of weblinks. GraphX is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new Graph abstraction: a directed multi-graph with properties attached to each vertex and edge.

To support graph computation, GraphX exposes a set of fundamental operators such as subgraph, joinVertices, and aggregateMessages on the baiss of an optimized variant of the Pregel API(Pregel is the system at Google that powers PageRank). In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.

We compute the PageRank of each user as follows:

```
//load the edges as a graph object
val graph = GraphLoader.edgeListFile(sc, “outlink.txt”)
// Run pagerank
val ranks = graph.pagerank(0.00000001).vertices
// join the rank with the webpages
val pages = sc.textFile(“pages.txt”).map{line => val fields = line.split(“,”)
(fields(0).toLong, fields(1)) }
val ranksByPagename = pages.join(ranks).map { case (id, (pagename, rank)) =>
(pagename, rank)}
//print the output
println(rankByPagename.collect().mkString(“\n”))
```

SparkR

R is a popular statistical programming language with a number of extensions that support data processing and machine learning tasks. However, interactive data analysis in R is usually limited as the runtime is single-threaded and can only process data sets that fit in a single machine’s memory. SparkR, an R package initially developed at the AMPLab, can provide an R frontend to Apache Spark and using Spark’s distributed computation engine allows us to run large scale data analysis from the R shell. SparkR exposes the RDD API of Spark as distributed lists in R. For example, one can read an input file from HDFS and process every line using *lapply* on a RDD. There is a caselet as follows:

```
sc<- sparkR.init(“local”)
```

```
lines <- textFile(sc, “hdfs://data.txt”)
wordsPerLine <- lapply(lines, function(line)) { length(unlist(strsplit(line,””)))})
```

In addition to lapply, SparkR also allows closures to be applied on every partition using lapplyWithPartition. Other supported RDD functions include operations like reduce, reduceByKey, groupByKey and collect.

SparkSQL

Spark SQL is a language provided to deal with the structured data. Using this one can run queries on the data and get some meaningful result. It supports the queries through SQL as well as HQL (Hive Query Language) which is Apache’s Hive version of SQL.

Spark Streaming

Spark Streaming gains data streams from input sources, process them in a cluster, push out to databases/ dashboards. Spark further chops up data streams into batches of few seconds. Spark treats each batch of data as RDDs and processes them using RDD operations. The processed results are pushed out as batches.

Spark applications

Some hot data problems that are solved well by a tool like Apache Spark include: 1. Real-time Log Data monitoring. 2. Massive Natural Language Processing 3. Large Scale Online Recommendation Systems.

A simple Wordcount application can be run in Spark shell as below.

val textFile = sc.textFile(“C:\Users\MyName\Documents\ obamaSpeech.txt”)

Comment: saves the text file as textFile

val counts = textFile.flatMap(line => line.split(” “)).map(word => (word, 1))
.reduceByKey(_ + _)

Comment: Calculate the total words by splitting with space

counts.count();

Results the output as below***

Long = 52

counts.saveAsTextFile(“C:\Users\ MyName\Desktop\counts1”)

***Comment: saves the file on my Desktop ***

Spark vs Hadoop

Spark and Hadoop are both popular Apache projects dedicated to big data processing. Hadoop, for many years, was the leading open source big data platform and many companies already use a distributed computing framework like Hadoop based on MapReduce. Table 9.1 provides a summary of the differences between Hadoop and Spark.

Feature	Hadoop	Spark
Purpose	Resilient cost-effective storage and processing of large data sets	Fast general-purpose engine for large-scale data processing
Core component	Hadoop Distributed File system (HDFS)	Spark Core, the in-memory processing engine.
Storage	HDFS manages massive data collections across multiple nodes within a cluster of commodity servers.	Spark doesn't do distributed storage. It operates on distributed data collections.
Fault Tolerance	Hadoop uses replication to achieve fault tolerance.	Spark uses RDD for fault tolerance that minimizes network I/O.
Nature of processing	Accompanied by MapReduce, it includes batch processing of this data in parallel mode	Batch as well as stream processing.
Sweet spot	Batch processing	Iterative and interactive processing jobs, that can fit in the memory
Processing Speed	Map Reduce is slow.	Spark can be up to 10x faster than MapReduce for batch processing and up to 100x faster for stream processing.
Security	More secure	Less secure
Failure recovery	Hadoop can recover from system faults or failures since data are written to disk after every operation	With Spark, data objects are stored in RDD. These can be reconstructed after faults or failures
Analytics tools		Built-in MLLib (Machine

	Separate engine	Learning) and GraphX (Graph Processing) libraries
Compatibility	Primary storage model is HDFS	Compatibility with HDFS and other storage formats
Language support	Java	Scala is native language. APIs for python, java, R, others.
Driving Organization	Yahoo	AMPLabs from UCBerkeley
Technology owners	Apache, Open-source, free	Open-source, free
Key Distributors	Cloudera, Horton, MapR	Databricks, AMPLabs
Cost of System	Medium to High	Medium to High

Conclusion

Spark is a new integrated system for big data processing. Its most important core abstraction is RDDs, along with relevant libraries like MLlib and GraphX. Spark is a really powerful open source processing engine build around speed, ease of use, and sophisticated analytics.

Review Questions

Q1: Describe the spark ecosystem.

Q2: Compare Spark and Hadoop in terms of their ability to do stream computing?

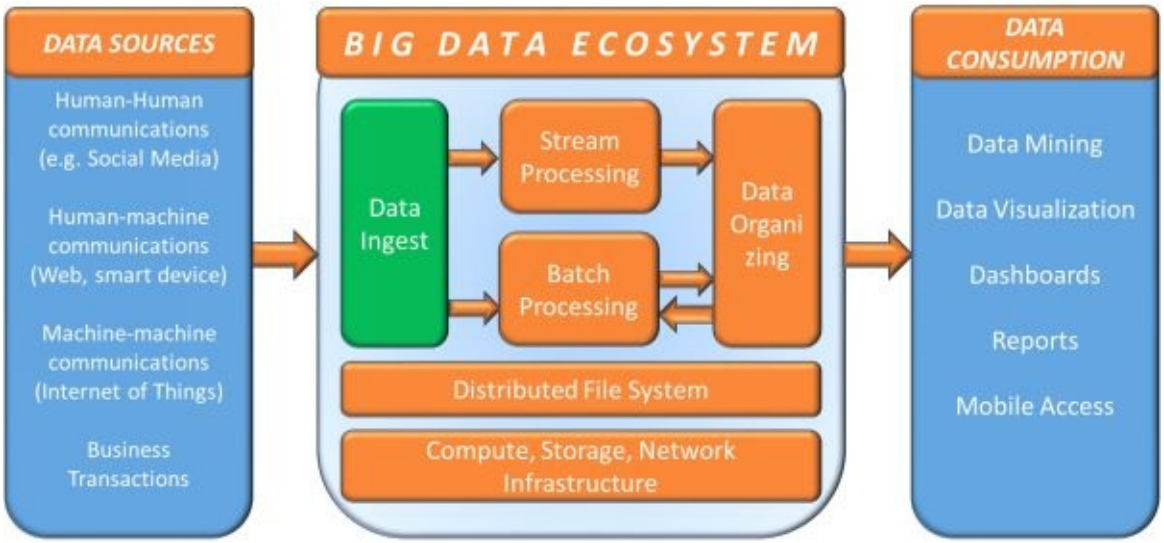
Q3: What is an RDD? How does it make Spark faster?

Q4: Describe three major capabilities in Spark for data analytics.

Chapter 8 – Ingesting Data

Wholeness

A Data ingesting system is a reliable and efficient point of reception for all data coming into a system. This system is designed to be flexible and scalable to receive data from various sources, at various times and speeds and quantities. The ingest system makes the data available for use by the target applications in real time. Ideally, all data would be smoothly received, and made available for downstream applications to securely and reliably access at their own convenience. A dedicating data ingest mechanism is achieved by creating a fast and flexible buffer for receiving and storing all incoming streams of data. The data in the buffer is stored in a sequential manner, and is made available to all consuming applications in a fast and orderly manner.



Big Data arrives into a system at unpredictable speeds and quantities. Business applications thereafter receive and process this data at some planned throughput capacity. An ingest buffer is needed to communicate the data without loss of data or speed. This buffer idea has historically been called a messaging system, not too dissimilar from a mailbox system at the post office. Incoming messages are put into a set of organized locations, from where the target applications would receive them when they are ready.

With huge amounts of data coming in from different sources, and many more consuming applications, a point-to-point system of delivering messages becomes inadequate and slow. Alternatively, incoming data can be categorized into certain topics, and stored in the respective location or locations for those topics. Instead of data being received and held in storage for a specific target application, now the data may be consumed by any application that is interested in data related to a topic. Each consuming application can choose to read data about one or more topics of its interest. This is called the publish-and-subscribe system.

Messaging Systems

A Messaging System is an asynchronous mode of communicating data between application. There are two generic kinds of messaging systems − a point-to-point system, and a publish-subscribe (pub-sub) system. Most of the messaging patterns now follow pub-sub model.

Point to Point Messaging System

In a point-to-point system, every message is directed at a particular receiver. A common queue can receive messages from many producers or messages. Any particular message can be received and consumed by only one receiver. Once that target consumer reads a message in the queue, that message disappears from that queue. The typical example of this system is an Order Processing System, where each order will be processed by one Order Processor.

Publish-Subscribe Messaging System

In a pub-sub messaging system, the applications publish their output to a standard messaging queue. The target recipient will only need to know where to get the message, whenever it is ready to pick up the message. Applications thus can ignore the mechanics of interaction with other applications, and simply care about the message itself. This is especially valuable when there may be many target recipients for a message. In a pub-sub system, messages are entered into the messaging queue asynchronously from client applications.

A message queuing system needs to be fast and secure to serve many applications, both producers and subscribers. Messages are also replicated across multiple locations for reliability of data.

There are two popular Data ingesting systems used in Big Data. An older system, called Flume, is closely tied to the Hadoop distributed file system. The new and more popular system is a general purpose system called Apache Kafka. In this chapter we will discuss the new system, Kafka.

Apache Kafka

Apache Kafka is an open source publish-and-subscribe message broker system. Kafka aims to provide an integrated high-throughput, low- latency messaging platform for handling real-time data feeds. In the abstract, it is a single point of contact between all producers and consumers of data. All producers of data send data to Kafka. All consumers of data read data from Kafka. (Figure 8.1)

Figure 8-1: Kafka core idea

Kafka is a distributed, partitioned, scalable, replicated messaging system, with a simple but unique design. It was initially developed by LinkedIn and was open sourced in early 2011. Apache Software Foundation is now responsible for its development and improvement. Kafka is a valuable for an enterprises level infrastructure because of its simplicity and scalability. Kafka system is written in the high-level Scala programming language.

Use Cases

Following are some popular use cases of Apache Kafka.

Messaging

Kafka is a very good alternative for a traditional message broker because Kafka messaging system has better throughput, built in partitioning, replication and better fault tolerance. Kafka is very good solution for a large scale message processing applications.

Website Activity Tracking

Website Activity Tracking was one of initial use cases for Kafka for LinkedIn. Users' online activity tracking pipeline was rebuilt as a set of real time data feeds. General web activity tracking includes very large volume of data, and Kafka is very good at handling this huge volume of data. User activity types such as page view, searches, clicks, etc can be designated as central topics, and the activity data can be published to those topics. Those events are available for real time or offline processing and reporting.

Stream Processing

Popular frameworks such as Storm and Spark Streaming read data from a topic, process it, and send to other users and consumer applications. They may even write it back to Kafka to a new topic. Kafka's strong durability is also very useful for stream processing.

Log Aggregation

Activity Log aggregation typically gathers physical log files from servers and puts them all in a central place for processing. Kafka can abstract away the details of the files and provide a cleaner abstraction of log data as a stream of messages. Use of Kafka then allows for lower-latency processing and easier support for multiple data sources and distributed data consumption. Unlike dedicated log-centric systems, Kafka offers higher performance and stronger durability guarantees due to replication.

Commit Log

Kafka can be used as external commit log for a distributed database system. This audit log can help to re-sync data between the failed nodes to restore their data. The log compaction in Kafka helps to achieve this feature more efficiently.

Kafka Architecture

In the abstract, Kafka brokers deal with producers and consumers of data. A producer pushes data into the ingest system at its own speed, scale and convenience. A consumer pulls data out of the system at its own speed, scale and convenience. All the received data is organized by categories, called topics. Incoming data is sorted and stored into topic servers. The consumers of data can subscribe to one or more topics (Figure 8.2).

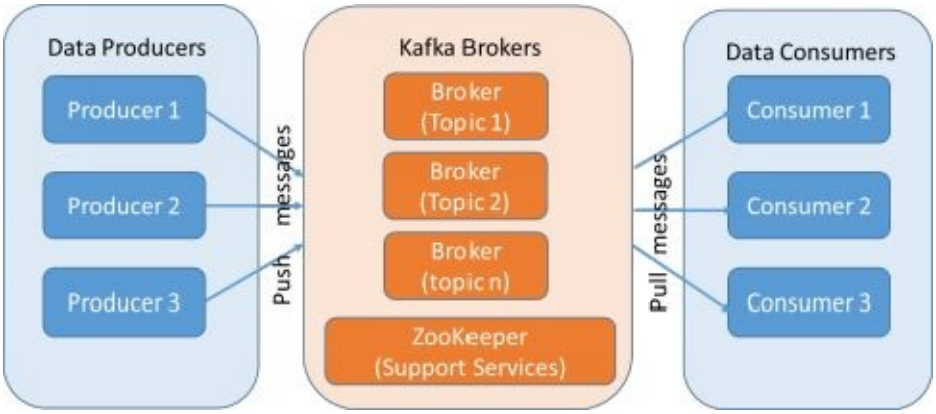


Figure 8-2:Kafka Ecosystem

There are more than one brokers (also called servers, or partitions) for each topic, for reliability of the messaging system. Thus two or more brokers will store data on each topic. Only one broker can be leader at any given time. In the lead broker fails, then a second one can automatically take over and prevent the loss of access to data.

Kafka is designed for distributed high throughput systems. In comparison to other messaging systems, Kafka has better throughput, built-in partitioning, replication and inherent fault-tolerance, which makes it a good fit for large-scale message processing applications. It has the ability to handle a large number of diverse consumers. It integrates very well with Apache Storm, Spark and other real-time streaming data applications. Kafka is very fast and can perform 2 million writes/sec. It also guarantees zero downtime and zero data loss.

There are a lot of contributing organizations helping to improve the Kafka open-source system. It has very well documented online resources. It has been used by many big organizations such as LinkedIn, Cisco System, Spotify, Paypal, HubSpot, Shopify, Uber and more. HubSpot uses Kafka to deliver real time notification of when a recipient opens their email. Paypal uses Kafka to process millions of updates in a minute.

Producers

A producer is responsible for selecting the partition, and the topic for the message that it wants to convey. It can use round-robin algorithm to balance the load among partitions. There can be both synchronous and asynchronous producers for producing message and publishing to the partition.

Consumers

A consumer is responsible for reading the data about the topic that it has subscribed. The consumer is responsible for reading the data within a reasonable period of time, before the

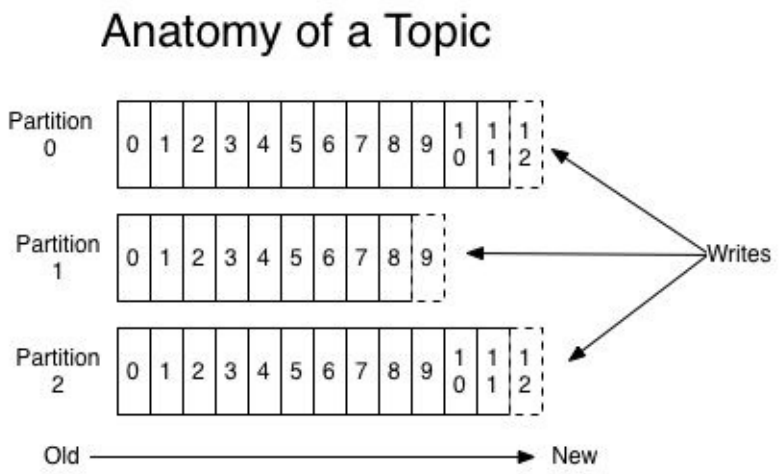
queues are emptied for efficient management of storage. Different consuming applications can read the data at different times. Kafka has stronger ordering guarantees than a traditional messaging system. A consumer needs to know how far it has read in that queue, so as to avoid duplicates or lose some data.

Broker

A broker is a server in a Kafka cluster. The cluster may have many such servers or brokers.

Topic

A topic is a category into which messages are published. For each topic there is a separate partition log for storage of messages. Each partition has an ordered sequence of messages for that topic. Each message in the partition is assigned a unique sequential number, also called the offset. This offset helps to identify each message within the partition.



The consumer reads the data sequentially according to offset numbers. The consumer maintains the offset to remember how far it has read. Generally, the offset increases linearly as messages are consumed. However, a consumer can reset offset to access the data gain and reprocess it as needed.

The Kafka cluster keeps all the published messages whether or not they have been consumed for a configurable period of time or not. For example, if the log retention is set to seven days, then for the seven days after publishing, the message is available for consumption. After seven days, Kafka discards the messages to free up space.

Kafka's performance is not affected by the size of data. Each partition must fit on the servers that host it, but a topic may have multiple partitions. This enables Kafka to manage an arbitrary amount of data. Also, it acts as the unit of parallelism.

Summary of Key Attributes

1. Disk based: Kafka works on a cluster of disks. It does not keep everything in memory, and keeps writing to the disk to make the storage permanent.
2. Fault tolerant: Data in Kafka is replicated across multiple brokers. When any leader broker fails, a follower broker takes over as leader and everything

continues to work normally.

3. Scalable: Kafka can scale up easily by adding more partitions or more brokers. More brokers help to spread the load and this provides greater throughput.
4. Low latency: Kafka does very little processing on the data. Thus it has very low latency rate Messages produced by the consumer are published and available to the consumer within a few milliseconds.
5. Finite Retention: Kafka by default keeps the message in the cluster for a week. After that the storage is refreshed. Thus the data consumers have up to a week to catch up on data, in case they fall behind for any reason.

Distribution

The Kafka cluster maintains multiple servers over the distributed network. The partitions of the log are maintained over this network. Each server handles data and requests for a share of the partitions. Each partition is replicated across a configurable number of servers for fault tolerance. But one of the server for each partition acts as the main server also called “leader” while it may or may not have one or more secondary server also known as “followers”. The leader server is responsible for handling all the read and write operation for the partition while the followers silently replicates the leader. The follower server becomes very helpful when the leader server fails. The follower server automatically becomes the leader and then handles the failure. One server can be a leader for some of the partitions on it, while it may be follower for other partitions. Thus one server can act as both leader and follower. This helps to balance the work load on the servers within the cluster.

Guarantees

Messages sent always maintain the order they were sent. For example, if a message M1 and M2 were sent by the same producer and M1 was sent first then the message M1 will have lower offset than message M2. Therefore, M1 will always appear before the M2 for the consumer.

Each topic has a replication factor N and the system can tolerate up to N-1 server failures without losing any messages committed to the log.

Client Libraries

Kafka supports following client libraries:

1. Python: Pure python implementation with full protocol support and Consumer Producer are also included.
2. C: High performance C library with full protocol support.
3. C++, Ruby, Javascript and more.

Apache ZooKeeper

Kafka is built on top of ZooKeeper. Apache Zookeeper is a distributed configuration and synchronization service in Hadoop clusters. Here it serves as the coordination interface between the Kafka brokers and consumers. The Kafka servers stores basic metadata in Zookeeper and shares information about topics, brokers, and consumer offsets (queue readers) and so on.

Since Zookeeper does it own layers of replication, the failure of a Kafka broker does not affect the state of the Kafka cluster. Even if Zookeeper fails, Kafka will restore the state, once the Zookeeper restarts. This gives zero downtime for Kafka. Zookeeper also manages the alternative leader broker selection, in case of a Kafka leader failure.

Kafka Producer example in Java

```
//Configure

Properties config =  new Properties();

config.setProperty(ProducerConfig.BOOTSTRAP_SERVER_CONFIG,
“localhost:8082”);

KafkaProducer producer = new KafkaProducer(config);

ProducerRecord record= new ProducerRecord(“topic”, “key”.getBytes(),
”value”.getBytes());

Future<RecordMetaData>  response = producer.send(record);
```

Conclusion

Big data is ingested using a dedicated system. These often take the form of messaging systems. Publish-and-subscribe systems are efficient ways of delivering data from many sources to many targets, in a reliable, secure and efficient way. Kafka is an open-source, reliable, secure, and scalable data publish-subscribe messaging system. It deals with producers as well as consumers of data. Messages are published to a set of central topics. Each consumer can subscribe to any number of topics. Kafka uses a leader-follower system of managing replicated partitions for the same set of data, to ensure full reliability and zero downtime.

Review Questions

Q1: What is a data ingest system? Why is it an important topic?

Q2: What are the two ways of delivering data from many sources to many targets?

Q3: What is Kafka? What are its advantages? Describe 3 use cases of Kafka.

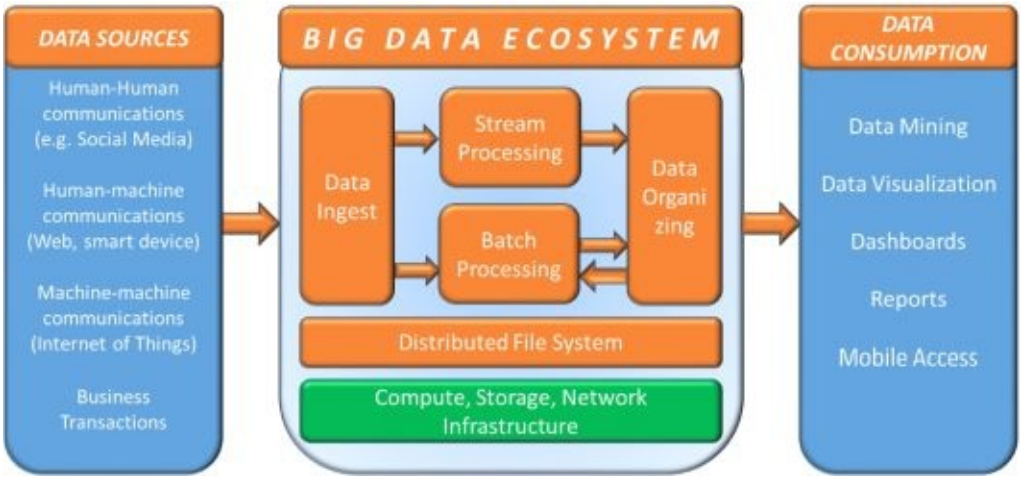
Q4: What is a topic? How does it help with data ingest management?

References

1. http://kafka.apache.org/documentation.html#introduction

Chapter 9 – Cloud Computing Primer

Cloud computing is a cost-effective and flexible mode of delivering IT infrastructure as a service to clients, over internet, on a metered basis. The cloud computing model offers clients enormous flexibility to use as much IT capacity – compute, storage, network – as needed without having to invest in a dedicated IT capacity on one's own. The IT usage can be scaled up or down in minutes. The complex IT infrastructure management skills are all owned by the cloud computing provider, and problems can be resolved much faster. The client can simply access a smoothly running IT infrastructure over a fast internet connection. IT capacity in the cloud can be purchased as a custom package depending upon one's needs in terms of average and peak IT requirements. The computing cloud is the ultimate cosmic computer aligned with all laws of nature.



Introduction

Managing very large and fast data streams is a huge challenge. It requires making critical decisions about its storage, structure, and access. This data would be stored in large clusters of hundreds or thousands of inexpensive computers. Such clusters are often called server farms. The location and size of such clusters impacts costs. The server farms may be located in their own data centers, or they may be rented from specialized third-party organizations called cloud computing service providers.

Cloud computing provides the IT leadership a cost-effective and predictable solution for reliably meeting their large data management needs. There are many vendors offering this service. Prices keep dropping regularly, because IT components keep getting cheaper, there is growing volume of business, and there is effective competition. With cloud computing, the IT expense becomes an operating expense rather than a capital expense. The costs of IT becomes aligned with revenue streams and makes cash flow management easier.

One of the main reasons for enterprises moving to cloud computing is to experiment with new and risky projects. This flexible model makes it much easier to launch new products and services, without being exposed to the risk of a heavy loss in IT infrastructure. For example, a new Hollywood movie's site will have millions of visitors to its website for a month before and for a month after the movie's release date. After that the visits to the website will drop dramatically. The website owner would benefit enormously from using a cloud computing model where they pay for the peak web usage capacity for those few months, and much less as the usage drops down. More importantly, the flexibility ensures that their website will not crash just in case the movie becomes a super-hit and attracts unusually large number of visitors to the website.

Cloud Computing Characteristics

Here are the major characteristics of a cloud computing model.

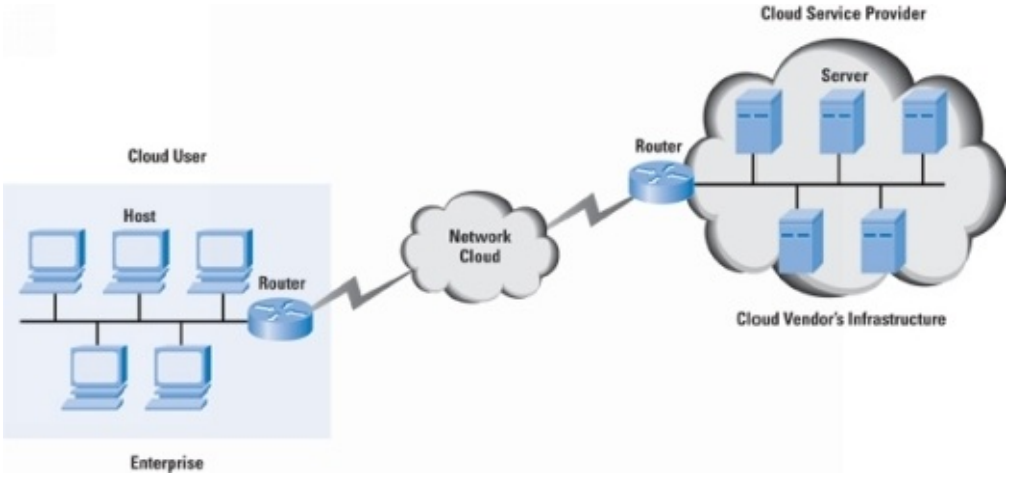
1. **Flexible Capacity**: The capacity can scale up rapidly. One can expand and reduce resources according to one's specific service requirements, as and when needed. The cloud internally does regular workload balancing among the needs of millions of clients, and this helps bring down costs for everyone.
2. **Attractive payment model**: Cloud computing works on a pay-per-use model. i.e. one pays only for what one uses, and for how long one uses it. IT costs become an expense rather than a capital expense for the client. The resource prices may be negotiated at long-term contract rates, and can also be purchased at spot market rates.
3. **Resiliency and Security**: The failure of any individual server and storage resources does not impact the user. The Servers and storage for all clients are isolated to maximize security of data.

In-house storage

Most organizations have data centers for running their regular IT operations. An organization may decide to expand its own data center to store large streams of data. The organization can ensure complete security and privacy of its data if it keeps all the data in-house. However, the costs and complexity of managing this data are increasing, and it is not cost-effective for every organization to manage huge data centers. Hiring and retaining scarce advanced skills to manage such data centers would also be a challenge.

Cloud storage

It is now becoming a trend for organizations to choose to store their data in massive data centers owned by other specialized companies. Their data and processing capacity resides in some sort of a huge cloud out there, which is accessible from anywhere anytime through a simple internet connection.



Companies like Amazon, Google, Microsoft, Apple, and IBM are among the major providers of cloud storage and computing services around the world. They own and operate data centers with millions of computers in them.

Figure 0-1: A cloud computing data center

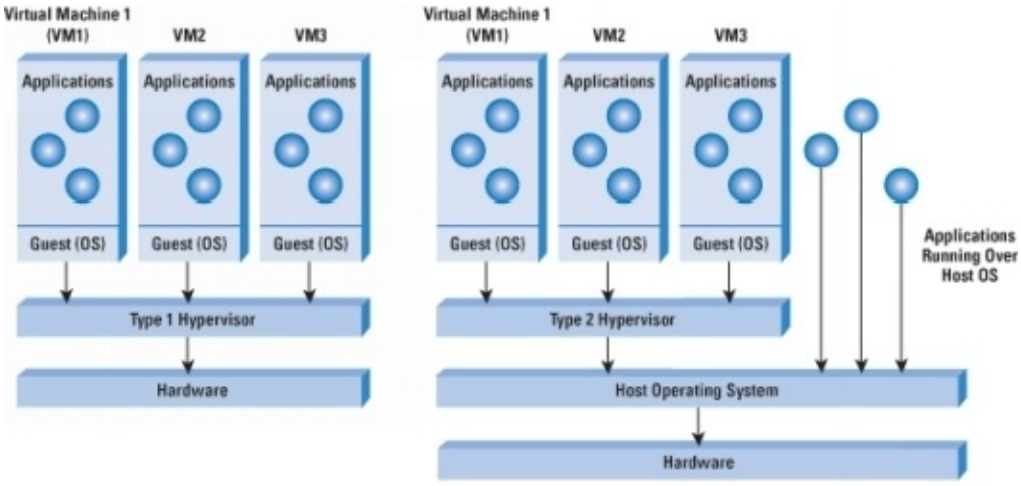
Commercially, cloud service providers are able to consolidate the requirements of thousands or millions of customers, and supply flexible amounts of data storage and computing facility available to clients on a per-usage basis. This pay model is similar to how electric utility companies charge consumers for their usage of electricity in homes and offices. Cloud computing offers much lower costs per use, just like using the electric utility costs much less than owning and operating one’s own electricity generators.

A major disadvantage of cloud storage is that the data is stored away from one’s physical control. Thus security of precious data is left to the hands of the cloud computing provider. While the security protocols are rapidly improving, however, there are no failsafe methods for securing data in the cloud. There is also a risk of being locked into one provider’s infrastructure. The cost-benefit tradeoffs have definitely tilted towards using cloud computing providers. At some future point in time, the cloud services providers might be heavily regulated like the electric utilities.

Cloud Computing: Evolution of Virtualized Architecture

Cloud computing is essentially a commercial model for virtualized server infrastructure. IBM began to offer time-sharing services on its mainframe computers beginning in the 1960s. Now that same technology has been offered on networks of small machines through the virtualization process.

Virtualization assumes that logical machines can be differentiated from physical machines. A physical server could run multiple *Virtual Machines* (VMs); and one virtual machine may span multiple physical servers. The virtualization software is called a hypervisor. It abstracts all machines into Virtual Machines, using easy GUI interface. A virtualization software can typically run on a heterogeneous physical infrastructure, and convert all IT capacity into a single unified capacity. This capacity can then be provisioned in slices and packages. The user applications are not aware that they are running in a virtualized environment; so they run as if running on a dedicated machine. The applications can also run on top of their own native operating systems.

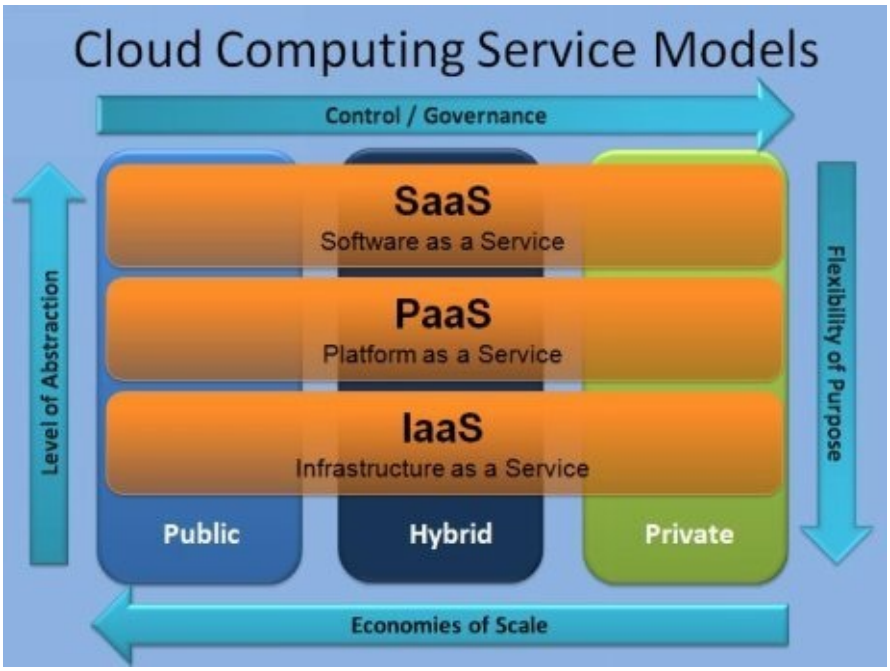


Type 1 Hypervisor-Based Virtualization

Type 2 Hypervisor-Based Virtualization

Cloud Service Models

There are two major dimensions to conceptualize the Cloud computing models: the scope of services received; and the control over and cost of those services.



1. The range of cloud computing services from a cloud computing provider, fall in three broad buckets:
    1. **Infrastructure as a service: This is the lowest level of services, and included only raw capacity of compute, storage, and networking. The price for this services is the lowest.**
    2. **Platform as a service**: This includes IaaS, along with other technologies and services. These are still very general tools such open source Hadoop or Spark or Cassandra implementation, along with certain monitoring tools. The costs are a little higher because of the additional management and monitoring services provided by the provider.
    3. **Software as a service**: This includes the computing platform as well as business applications that get work done. For example, salesforce.com was one of the first CRM application sold only on a SaaS model. Google sells an email service to organizations on a per-user-per-month basis. This is also the most expensive type of cloud service.
2. The other way the cloud services differ is in terms of the ownership and control.
    1. **Public cloud**: This will be a large shared infrastructure made available to one and all, in a low-cost and multi-tenancy model. The client can access it using any device. The downside is that the data also resides on the cloud, and thus could be vulnerable to theft or hacking. The costs to

client are low, and variable depending upon use.

2. **Private cloud**: This is a cloud version of an in-house IT infrastructure. The organization will have exclusive control over the entire infrastructure. The costs would be fixed and higher.
3. **Hybrid cloud**: This is a mix of flexibility of capacity, and much control over some key aspects of it. One could retain complete control over critical applications, while using shared infrastructure for non-critical applications.

All levels of infrastructure and pay models are useful, as they server different levels of needs for client organizations. However, most of the growth in cloud computing is happening because of the attractiveness of the low cost of the public cloud model.

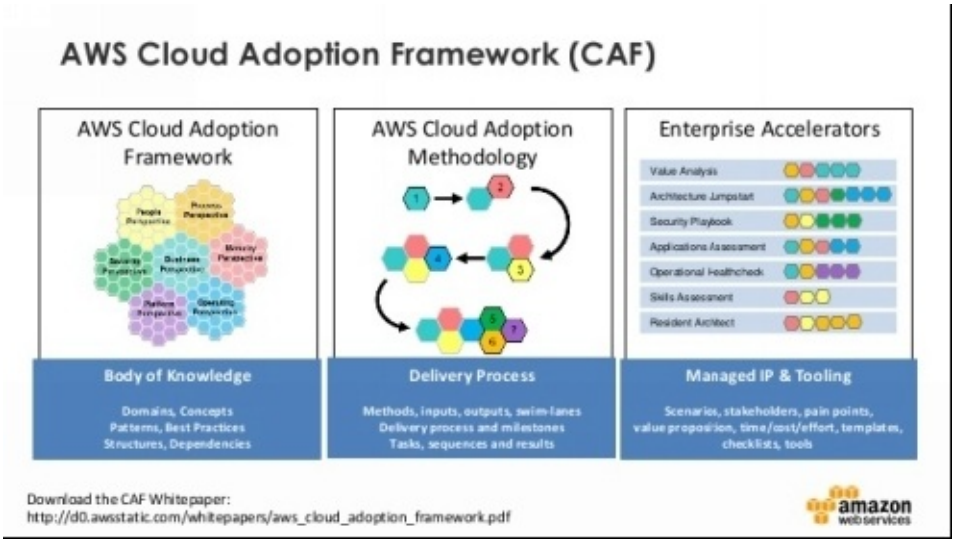
Cloud Computing Myths

There are a couple of misconceptions about the costs and benefits of cloud computing.

1. *Myth: Public Cloud computing would satisfy all the requirement: scalability, flexibility, pay per use, resilience, multitenancy, and security.* Depending upon the type of service selected (SaaS, IaaS, or PaaS), the service can satisfy specific subsets of these requirements.
2. *Myth: Cloud computing would be useful only if you are outsourcing your IT functions to an external service provider.* One could use a private cloud computing model for a section of IT applications to offer on-demand, scalable, and pay-per-use deployments within your enterprise's own data center.

Cloud Computing: Getting Started

Here below is a framework for cloud adoption. Learn more about the context for getting benefits from cloud computing. Select the right model and level of cloud capacity. Set up the applications and a monitoring system for those application and the total cloud footprint. Choose a service provider, say Amazon Web Services, the leading provider of cloud computing. Use Appendix A to install Hadoop on AWS EC2 public cloud infrastructure.



Conclusion

Cloud computing is a business model to provide shared, flexible, cost-effective IT infrastructure to get started quickly on building an application. For Big Data applications, it can be even more attractive to test the system using rented facilities, before making the determination of investing in dedicated IT infrastructure.

Review Questions

Q1: Describe Cloud Computing model.

Q2: What are the advantages of cloud computing over in-house computing

Q3: Describe the technical architecture for Cloud computing.

Q4: Name a few major providers of cloud computing services.

Section 3

This section covers the other relevant concepts and tutorials for effectively managing and utilizing Big Data.

Chapter 10 will bring all the tools together in a case study of developing web log analyzer, as an example of a useful Big Data application.

Chapter 11 will cover the overall view of Data Mining tools and techniques to extract benefit from Big Data.

Appendix 1 shows step by step, the way to install a Hadoop cluster on a cloud computing platform.

Appendix 2 is a tutorial on installing and running Spark.

Chapter 10 – Web Log Analyzer application case study

Introduction

A web log analyzer is an automated software tool that helps to analyze and make decisions on a number of issues regarding web application server logs. An ideal web log analyzer would analyze unlimited streams of data and help keep the entire universe running smoothly and without fault. This would be done by eliminating the need for manually accessing the logs, automating the flow of information, and alerting the system administrator as needed.

Client-Server Architecture

Every web-based application runs on a client-server architecture. Clients are entities that access servers, and servers are entities that respond to the client with a solution. A lot of clients simultaneously try to access servers. The servers may be database server, network server, the application server, or any server in the n-tier architecture. For each request, a log entry is generated. The speed of access requests determined the stream of log entries. This leads to a potentially huge log over time. The log can be processed as stream of data. This log can also be stored on the servers for later analysis.

Logs can be used for monitoring, audit and analysis purposes. It can help with error diagnostics in case a website becomes slow or it goes down. Logs can be analyzed to detect hacking activity. They can also be analyzed to summarize the popularity of webpages, and the distribution of the page requesters. It can help with access volumes, and for scaling up or down the infrastructure.

Web Log analyzer

The log analyzer received streaming logs from a server location, and analyzes multiple things using many algorithms to generate the desired results. The system is completely automated. The log is produced, and it is consumed it to make real-time reports. It is easy to imagine the massive dataflow produced by the log in the server environment while it is also being analyzed simultaneously on the administrator side. `

Requirements

This is a log analyzer to analyze a web application hosted on a server. It is a busy application owned by a big company. It receives more than 15000 web access requests per hour. All the access requests need to be logged, and dumped to Hadoop File system periodically. The analyzer is required to ingest real-time log data, and filter out a part of data for analyzing and dumping to HDFS. It has to do streaming data flow management as well as batch processing. The analyzer needs to process the data before it is dumped into HDFS, and also after it is put into HDFS. The system administrators should be alerted in real time about possible threats, overloads, delays, potentials errors, and any other damages. The results of all the analyses need to be stored in a database for later presentation in a graphical format. The results have to be made available for any period of time, without any missing time values. The log data has to be preserved for future without losing any log data.

Solution Architecture

Get streaming data using Apache Flume, and send it to HDFS. Use Apache Spark for data flow management platform and processing engine. Store the results of analysis in MongoDB. This is a safe solution, because the data gets stored into Hadoop cluster and is available for future requirements, even while it is being analyzed in real time. The results of real-time processing also go into MongoDB.

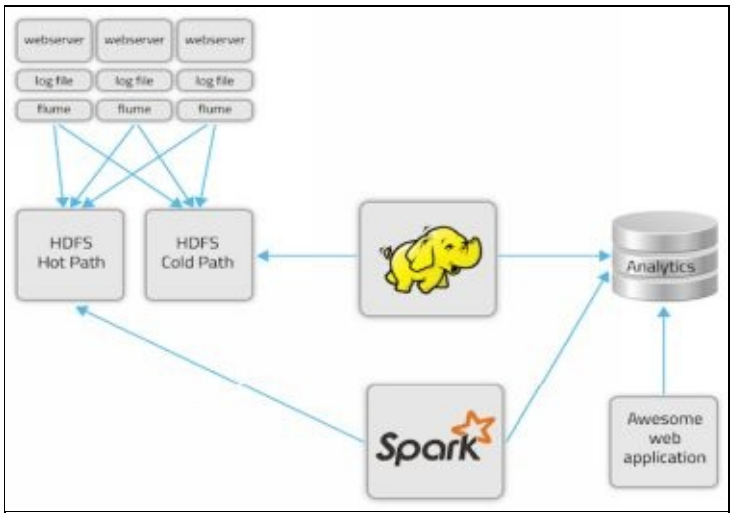


Fig 10.1 : Web Log Analyzer Architecture

Benefits of this solution

The advantages of this solution are:

1. Real time logging and analysis data generated on server is streamed directly to HDFS by Flume agent without delay. Every log entry generated over every single point of time is analyzed and used for monitoring and decision making,
2. Automatic log handling and storage. Loading data into HDFS normally requires manually running certain Hadoop commands. This log analyzer uses a Flume agent or spark streaming to handle all data on its own, without any externally managed efforts.
3. Easy and convenient implement using built-in and easy-to-customize machine learning algorithms in Spark.
4. Easy error handling, server request handling, and overall server performance optimization. It makes server smarter by keeping track of almost every aspects of server.

Technology stack

The technology stack used for this application is shown below. A brief of each component follows.

1. Apache Spark v2
2. Hadoop 2.6.0 cdh5
3. Apache Flume
4. Scala, Java
5. MongoDB
6. RestFul Webservices
7. Front UI tools
8. Linux Shell Scripts

Apache Spark

Spark is fast in-memory-based cluster computing technology, designed for fast and streaming computation. It is built on top of Hadoop and MapReduce system, and it extends MapReduce model to use more types of computation, which includes interactive queries and stream processing. It has lot of libraries and packages like machine learning (MLLib), graph computation (GraphX) etc. It claims to execute 10 to 100 times faster than Hadoop because of its in-memory computation model. It also supports multiple languages such as Scala, Python, Java, and R.

Spark Deployment

1. Standalone
2. Hadoop YARN
3. SIMR: Spark in mapReduce //Mesos

Components of Spark

Spark Sql: Data abstraction called schemaRDD, which provides support for structured and semi-structured data.

Spark Streaming: Ingests data in mini batch and perform RDD transformation on those mini-batches. Streaming data analytics using RDD

MLib (machine learning): It is a distributed machine learning framework, which operates in-memory at high speed, and offers many ML algorithms.

GraphX: This distributed graph processing framework provides API for many graph computation algorithms.

Spark Core: This is a general execution engine for spark platform upon which all other functionality is built. It takes care of task dispatching and scheduling, and basic I/O functionalities.

Spark-shell: It is a powerful tool to analyze data interactively. It is available on scala and python. Spark's primary data abstraction is an in-memory collections of items called RDD. It can be created from Hadoop input formats like HDFS, and by transforming existing RDDs using filters and maps into new RDDs.

Scripting and Programming model using SparkContext: One can use an IDE to develop and test the analytics code. One can then create a jar to run the analytics using Hadoop architecture. The jar can also be submitted using spark-submit utility to the Spark engine. For example:

spark-submit —class apache.accesslogs.ServerLogAnalyzer —master

* local ScalaSpark/Scala1/target/scala-2.10/Scala1-assembly-1.0.jar > output.txt

HDFS

HDFS is a distributed file system, that is at the core of Hadoop system.

- Deployed on low cost commodity hardware
- Fault tolerant
- Supports Batch Processing
- Designed for large dataset or large files
- Maintains coherence through write once read many times
- Moving computation to the location of the data.

MongoDB

It is document-oriented database. It came into existence as a NoSQL database.

Apache Flume

Flume is an open source tool for handling streaming logs or data. It is a distributed and reliable system for efficiently collecting, aggregating and moving large amount of data from many different sources to a centralized data store. It is a popular tool to assist with data flow and storage to HDFS. Flume is not restricted to log data. The data sources are customizable so it might be any source like event data, traffic data, social media data, or any other data source. The major Components of Flume are:

- Event
- Agent
- Data Generators
- Centralized Stores

Overall Application logic

The system reads access logs and presents the results in tabular and graphical form to end users. This system provides the following major functions:

1. Calculate content size
2. Count Response code
3. Analyze requesting IP-address
4. Manage End points

Technical Plan for the Application

Technically, the project follows the following structure:

1. Flume takes streaming log from running application server and stores in HDFS. Flume uses compression to store huge log files to speed up the data transfer and for storage efficiency.
2. Apache Spark uses HDFS as input source and analyzes data using MLLib. Apache Spark stores analyzed data in MongoDB
3. RESTful java service presents JSON objects fetching from MongoDB and sending to Front end. Graphical tools are used to present data.

Scala Spark code for log analysis

Note: This application is written in Scala language. Below is the operative part of the code. Visit github link below for the complete Scala code for this application.

```
//calculates size of log, and provides min, max and average size
// caching is done for repeatedly used factors
def calcContentSize(log: RDD[AccessLogs]) = {
    val size = log.map(log => log.contentSize).cache()
    val average = size.reduce(_ + _) / size.count()
    println("ContentSize:: Average :: " + average + " " +
    " || Maximum :: " + size.max() + "  || Minimum ::" + size.min() )
  }

//Send all the response code with its frequency of occurrence as Output
def responseCodeCount(log: RDD[AccessLogs]) = {
    val responseCount = log.map(log => (log.responseCode, 1))
    .reduceByKey(_ + _)
    .take(1000)
    println(s""""ResponseCodes Count : ${responseCount.mkString("[", ",", "]")} """")
  }

//filters ipaddresses that have more then 10 requests in server log
  def ipAddressFilter(log: RDD[AccessLogs]) = {
    val result = log.map(log => (log.ipAddr, 1))
    .reduceByKey(_ + _)
    .filter(count => count._2 > 1)
    // .map(_._1).take(10)
    .collect()
    println("IP Addresses Count :: ${result.mkString("[", ",", "]")}" )
  }  }
```

Sample Log data

Sample Input Data:

Input Fields (selected fields):

Certain fields have been omitted to make the code clear. The response code has been colored in red as it is the basis of the major reports.

1. ipAddress: String,
2. dateTime: String,
3. method: String,
4. endPoint: String,
5. protocol: String,
6. responseCode: Long,
7. contentSize: Long

Sample Input Rows of Data:

64.242.88.10 [07/Mar/2014:16:05:49 -0800] “GET /twiki/bin/edit/Main/Double_bounce_sender?topicparent=Main.ConfigurationVariables HTTP/1.1” 401 12846

64.242.88.10 [07/Mar/2014:16:06:51 -0800] “GET /twiki/bin/rdiff/TWiki/NewUserTemplate?rev1=1.3&rev2=1.2 HTTP/1.1” 200 4523

64.242.88.10 [07/Mar/2014:16:10:02 -0800] “GET /mailman/listinfo/hsdivision HTTP/1.1” 200 6291

64.242.88.10 [07/Mar/2014:16:11:58 -0800] “GET /twiki/bin/view/TWiki/WikiSyntax HTTP/1.1” 200 7352

64.242.88.10 [07/Mar/2014:16:20:55 -0800] “GET /twiki/bin/view/Main/DCCAndPostFix HTTP/1.1” 200 5253

64.242.88.10 [07/Mar/2014:16:23:12 -0800] “GET /twiki/bin/oops/TWiki/AppendixFileSystem?template=oopsmore&param1=1.12&param2=1.12 HTTP/1.1” 200 11382

Sample Output of Web Log Analysis

ContentSize:: Average:: 10101 || Maximum :: 138789 || Minimum ::0

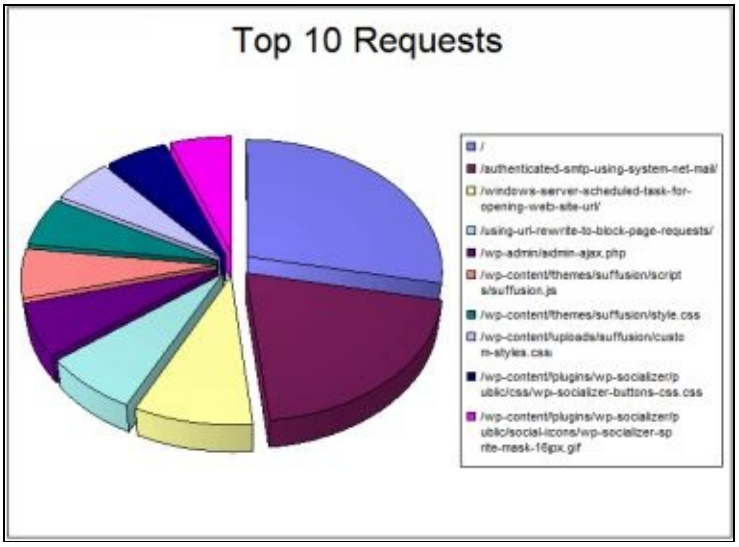
ResponseCodes Count: [(401,113), (200,591), (302,1)]

IP Addresses Count:: [(127.0.0.1, **31**), (207.195.59.160, **15**), (67.131.107.5, **3**), (203.147.138.233, **13**), (64.242.88.10, **452**), (10.0.0.153, **188**)]

EndPoints :: [(/wap/Project/login.php,15),(/cgi-bin/mailgraph.cgi/mailgraph_2.png,12), (/cgi-bin/mailgraph.cgi/mailgraph_0.png,12),(/wap/Project/loginsubmit.php,12),(/cgi-bin/mailgraph.cgi/mailgraph_2_err.png,12),(/cgi-bin/mailgraph.cgi/mailgraph_1.png,12), (/cgi-bin/mailgraph.cgi/mailgraph_0_err.png,12),(/cgi-bin/mailgraph.cgi/mailgraph_1_err.png,12),(/cgi-bin/mailgraph.cgi/mailgraph_3_err.png,12),(/cgi-bin/mailgraph.cgi/mailgraph_3.png,12)]

Intermediate data is stored in Hadoop File System in CSV format

To see detailed code, visit: https://github.com/databricks/reference-apps/blob/master/logs_analyzer/chapter1/scala/src/main/scala/com/databricks/apps/logs/ch



This web log analyzer can be enhanced in many ways. For example, it can analyze history of logs from previous years and discover web access trends. This application can also be made to discard data older than 5 years into permanent and backup storage.

Conclusion and Findings

There are more than 100 technologies around Apache ecosystem. Most basic is the MapReduce technique used by Hadoop engine. Many stacks are available on top of MapReduce. It is important to incorporate the right sets of elements to develop the right stack for the particular large scale data analytics. A few awesome technologies like HDFS, Spark, Hive, MongoDB, and Flume/Kafka is likely to make the big data application powerful and worthy.

It is also useful to experiment with many other technologies during the development of this log analyzer. Flume and Kafka are most powerful tools to handle streaming data. Spark has its own streaming API, but it’s not easy to incorporate with HDFS storage. Developing this application also helps to learn Linux based tasks and shell scripts along with some data handling tools like AWK and Stream Editor.

This application reduces burden of manual handling of logs on database, application or history servers. Moreover, it helps to present analyzed data in an impressive way that leads to easy decision making. This application came into development after doing much research on big data tools such as Apache Spark. That saved a lot time and cost later. It was developed using agile development practices.

Review Questions

Q1. Describe the advantages of a web log analyzer.

Q2. Describe the major challenges in developing this application.

Q3: Check out the references below. Identify 3-4 major lessons learned from the code and video.

Chapter 10: Data Mining Primer

Data mining is the art and science of discovering knowledge, insights, and patterns in data. It is the act of extracting useful patterns from an organized collection of data. Patterns must be valid, novel, potentially useful, and understandable. The implicit assumption is that data about the past can reveal patterns of activity that can be projected into the future.

Data mining is a multidisciplinary field that borrows techniques from a variety of fields. It utilizes the knowledge of data quality and data organizing from the databases area. It draws modeling and analytical techniques from statistics and computer science (artificial intelligence) areas. It also draws the knowledge of decision-making from the field of business management.

The field of data mining emerged in the context of pattern recognition in defense, such as identifying a friend-or-foe on a battlefield. Like many other defense-inspired technologies, it has evolved to help gain a competitive advantage in business.

For example, “customers who buy cheese and milk also buy bread 90 percent of the time” would be a useful pattern for a grocery store, which can then stock the products appropriately. Similarly, “people with blood pressure greater than 160 and an age greater than 65 were at a high risk of dying from a heart stroke” is of great diagnostic value for doctors, who can then focus on treating such patients with urgent care and great sensitivity.

Past data can be of predictive value in many complex situations, especially where the pattern may not be so easily visible without the modeling technique. Here is a dramatic case of a data-driven decision-making system that beats the best of human experts. Using past data, a decision tree model was developed to predict votes for Justice Sandra Day O’Connor, who had a swing vote in a 5–4 divided US Supreme Court. All her previous decisions were coded on a few variables. What emerged from data mining was a simple four-step decision tree that was able to accurately predict her votes 71 percent of the time. In contrast, the legal analysts could at best predict correctly 59 percent of the time. (Source: Martin et al. 2004)

Gathering and selecting data

To learn from data, quality data needs to be effectively gathered, cleaned and organized, and then efficiently mined. One requires the skills and technologies for consolidation and integration of data elements from many sources.

Gathering and curating data takes time and effort, particularly when it is unstructured or semistructured. Unstructured data can come in many forms like databases, blogs, images, videos, audio, and chats. There are streams of unstructured social media data from blogs, chats, and tweets. There are streams of machine-generated data from connected machines, RFID tags, the internet of things, and so on. Eventually the data should be *rectangularized*, that is, put in rectangular data shapes with clear columns and rows, before submitting it to data mining.

Knowledge of the business domain helps select the right streams of data for pursuing new insights. Only the data that suits the nature of the problem being solved should be gathered. The data elements should be relevant, and suitably address the problem being solved. They could directly impact the problem, or they could be a suitable proxy for the effect being measured. Select data could also be gathered from the data warehouse. Every industry and function will have its own requirements and constraints. The health care industry will provide a different type of data with different data names. The HR function would provide different kinds of data. There would be different issues of quality and privacy for these data.

Data cleansing and preparation

The quality of data is critical to the success and value of the data mining project. Otherwise, the situation will be of the kind of garbage in and garbage out (GIGO). The quality of incoming data varies by the source and nature of data. Data from internal operations is likely to be of higher quality, as it will be accurate and consistent. Data from social media and other public sources is less under the control of business, and is less likely to be reliable.

Data almost certainly needs to be cleansed and transformed before it can be used for data mining. There are many ways in what data may need to be cleansed – filling missing values, reigning in the effects of outliers, transforming fields, binning continuous variables, and much more – before it can be ready for analysis. Data cleansing and preparation is a labor-intensive or semi-automated activity that can take up to 60-80% of the time needed for a data mining project.

Outputs of Data Mining

Data mining techniques can serve different types of objectives. The outputs of data mining will reflect the objective being served. There are many ways of representing the outputs of data mining.

One popular form of data mining output is a decision tree. It is a hierarchically branched structure that helps visually follow the steps to make a model-based decision. The tree may have certain attributes, such as probabilities assigned to each branch. A related format is a set of business rules, which are if-then statements that show causality. A decision tree can be mapped to business rules. If the objective function is prediction, then a decision tree or business rules are the most appropriate mode of representing the output.

The output can be in the form of a regression equation or mathematical function that represents the best fitting curve to represent the data. This equation may include linear and nonlinear terms. Regression equations are a good way of representing the output of classification exercises. These are also a good representation of forecasting formulae.

Population “centroid” is a statistical measure for describing central tendencies of a collection of data points. These might be defined in a multidimensional space. For example, a centroid could be “middle-aged, highly educated, high-net worth professionals, married with two children, living in the coastal areas”. Or a population of “20-something, ivy-league-educated, tech entrepreneurs based in Silicon Valley”. Or it could be a collection of “vehicles more than 20 years old, giving low mileage per gallon, which failed environmental inspection”. These are typical representations of the output of a cluster analysis exercise.

Business rules are an appropriate representation of the output of a market basket analysis exercise. These rules are if-then statements with some probability parameters associated with each rule. For example, those that buy milk and bread will also buy butter (with 80 percent probability).

Evaluating Data Mining Results

There are two primary kinds of data mining processes: supervised learning and unsupervised learning. In supervised learning, a decision model can be created using past data, and the model can then be used to predict the correct answer for future data instances. Classification is the main category of supervised learning activity. There are many techniques for classification, decision trees being the most popular one. Each of these techniques can be implemented with many algorithms. A common metric for all of classification techniques is predictive accuracy.

Predictive Accuracy = (Correct Predictions) / Total Predictions

Suppose a data mining project has been initiated to develop a predictive model for cancer patients using a decision tree. Using a relevant set of variables and data instances, a decision tree model has been created. The model is then used to predict other data instances. When a true positive data point is positive, that is a correct prediction, called a true positive (TP). Similarly, when a true negative data point is classified as negative, that is a true negative (TN). On the other hand, when a true-positive data point is classified by the model as negative, that is an incorrect prediction, called a false negative (FN). Similarly, when a true-negative data point is classified as positive, that is classified as a false positive (FP). This is represented using the confusion matrix (Figure 4.1).

ConfusionMatrix		**True Class**	
		Positive	*Negative*
Predicted Class	Positive	**True Positive (TP)**	False Positive (FP)
Predicted class	Negative	False Negative (FN)	**True Negative (TN)**

Figure 10.1: Confusion Matrix

Thus the predictive accuracy can be specified by the following formula.

Predictive Accuracy = (TP +TN) / (TP + TN + FP + FN).

All classification techniques have a predictive accuracy associated with a predictive model. The highest value can be 100%. In practice, predictive models with more than 70% accuracy can be considered usable in business domains, depending upon the nature of the business.

There are no good objective measures to judge the accuracy of unsupervised learning techniques such as Cluster Analysis. There is no single right answer for the results of these techniques. For example, the value of the segmentation model depends upon the value the decision-maker sees in those results.

Data Mining Techniques

Data may be mined to help make more efficient decisions in the future. Or it may be used to explore the data to find interesting associative patterns. The right technique depends upon the kind of problem being solved (Figure 10.2).

Data Mining Techniques		
Supervised Learning (Predictive ability based on past data)	Classification – Machine Learning	Decision Trees Neural Networks
	Classification - Statistics	Regression
Unsupervised Learning (Exploratory analysis to discover patterns)	Clustering Analysis	
	Association Rules	

Figure 10.2: Important Data Mining Techniques

The most important class of problems solved using data mining are classification problems. Classification techniques are called supervised learning as there is a way to supervise whether the model is providing the right or wrong answers. These are problems where data from past decisions is mined to extract the few rules and patterns that would improve the accuracy of the decision making process in the future. The data of past decisions is organized and mined for decision rules or equations, that are then codified to produce more accurate decisions.

Decision trees are the most popular data mining technique, for many reasons.

1. Decision trees are easy to understand and easy to use, by analysts as well as executives. They also show a high predictive accuracy.
2. Decision trees select the most relevant variables automatically out of all the available variables for decision making.
3. Decision trees are tolerant of data quality issues and do not require much data preparation from the users.
4. Even non-linear relationships can be handled well by decision trees.

There are many algorithms to implement decision trees. Some of the popular

ones are C5, CART and CHAID.

Regression is a most popular statistical data mining technique. The goal of regression is to derive a smooth well-defined curve to best the data. Regression analysis techniques, for example, can be used to model and predict the energy consumption as a function of daily temperature. Simply plotting the data may show a non-linear curve. Applying a non-linear regression equation will fit the data very well with high accuracy. Once such a regression model has been developed, the energy consumption on any future day can be predicted using this equation. The accuracy of the regression model depends entirely upon the dataset used and not at all on the algorithm or tools used.

Artificial Neural Networks (ANN) is a sophisticated data mining technique from the Artificial Intelligence stream in Computer Science. It mimics the behavior of human neural structure: Neurons receive stimuli, process them, and communicate their results to other neurons successively, and eventually a neuron outputs a decision. A decision task may be processed by just one neuron and the result may be communicated soon. Alternatively, there could be many layers of neurons involved in a decision task, depending upon the complexity of the domain. The neural network can be trained by making a decision over and over again with many data points. It will continue to learn by adjusting its internal computation and communication parameters based on feedback received on its previous decisions. The intermediate values passed within the layers of neurons may not make any intuitive sense to an observer. Thus, the neural networks are considered a black-box system.

Cluster Analysis is an exploratory learning technique that helps in identifying a set of similar groups in the data. It is a technique used for automatic identification of natural groupings of things. Data instances that are similar to (or near) each other are categorized into one cluster, while data instances that are very different (or far away) from each other are categorized into separate clusters. There can be any number of clusters that could be produced by the data. The K-means technique is a popular technique and allows the user guidance in selecting the right number (K) of clusters from the data. Clustering is also known as the segmentation technique. It helps divide and conquer large data sets. The technique shows the clusters of things from past data. The output is the centroids for each cluster and the allocation of data points to their cluster. The centroid definition is used to assign new data instances can be assigned to their cluster homes. Clustering is also a part of the artificial intelligence family of techniques.

Association rules are a popular data mining method in business, especially

where selling is involved. Also known as market basket analysis, it helps in answering questions about cross-selling opportunities. This is the heart of the personalization engine used by ecommerce sites like Amazon.com and streaming movie sites like Netflix.com. The technique helps find interesting relationships (affinities) between variables (items or events). These are represented as rules of the form $X \Rightarrow Y$, where X and Y are sets of data items. A form of unsupervised learning, it has no dependent variable; and there are no right or wrong answers. There are just stronger and weaker affinities. Thus, each rule has a confidence level assigned to it. A part of the machine learning family, this technique achieved legendary status when a fascinating relationship was found in the sales of diapers and beers.

Mining Big Data

As data grows larger and larger, there are a few ways in which analyzing Big data is different.

From Causation to Correlation

There is more data available than there are theories and tools available to explain it. Historically, theories of human behavior, and theories of universe in general, have been intuited and tested using limited and sampled data, with some statistical confidence level. Now that data is available in extremely large quantities about many people and many factors, there may be too much noise in the data to articulate and test clean theories. In that case, it may suffice to value co-occurrences or correlation of events as significant without necessarily establishing strong causation.

From Sampling to the Whole

Pooling all the data together into a single big data system can help discover events, that help bring about a fuller picture of the situation, and highlight threats or opportunities that an organization faces. Working from the full dataset can enable discovering remote but extremely valuable insights. For example, an analysis of the purchasing habits of millions customers and their billions transactions at their thousands of stores can give an organization a vast, detailed and dynamic view of sales patterns in their company, which may not be available from the analysis of small samples of data by each store or region.

From Dataset to Data stream

A flowing stream has a perishable and unlimited connotation to it, while a dataset has a finitude and permanence about it. With any given infrastructure, one can only consume so much data at a time. Data streams are many, large and fast. Thus one has to choose which of the many streams of data does one want to engage with. It is equivalent to deciding which stream to fish in. The metrics used for analysis of streams tend to be relatively simple and relate to time dimension. Most of the metrics are statistical measures such as counts and means. For example, a company might want to monitor customer sentiment about its products. So they could create a social media listening platform that would read all tweets and blogposts about them in real-time. This platform would (a) keep a count of positive and negative sentiment messages every minute, and (b) flag any messages that merit attention such as sending an online advertisement or purchase offer to that customer.

Data Mining Best Practices

Effective and successful use of data mining activity requires both business and technology skills. The business aspects help understand the domain and the key questions. It also helps one imagine possible relationships in the data, and create hypotheses to test it. The IT aspects help fetch the data from many sources, clean up the data, assemble it to meet the needs of the business problem, and then run the data mining techniques on the platform.

An important element is to go after the problem iteratively. It is better to divide and conquer the problem with smaller amounts of data, and get closer to the heart of the solution in an iterative sequence of steps. There are several best practices learned from the use of data mining techniques over a long period of time. The Data Mining industry has proposed a Cross-Industry Standard Process for Data Mining (CRISP-DM). It has six essential steps (Figure 4.3):

1. *Business Understanding*: The first and most important step in data mining is asking the right business questions. A question is a good one if answering it would lead to large payoffs for the organization, financially and otherwise. In other words, selecting a data mining project is like any other project, in that it should show strong payoffs if the project is successful. There should be strong executive support for the data mining project, which means that the project aligns well with the business strategy. A related important step is to be creative and open in proposing imaginative hypotheses for the solution. Thinking outside the box is important, both in terms of a proposed model as well in the data sets available and required.

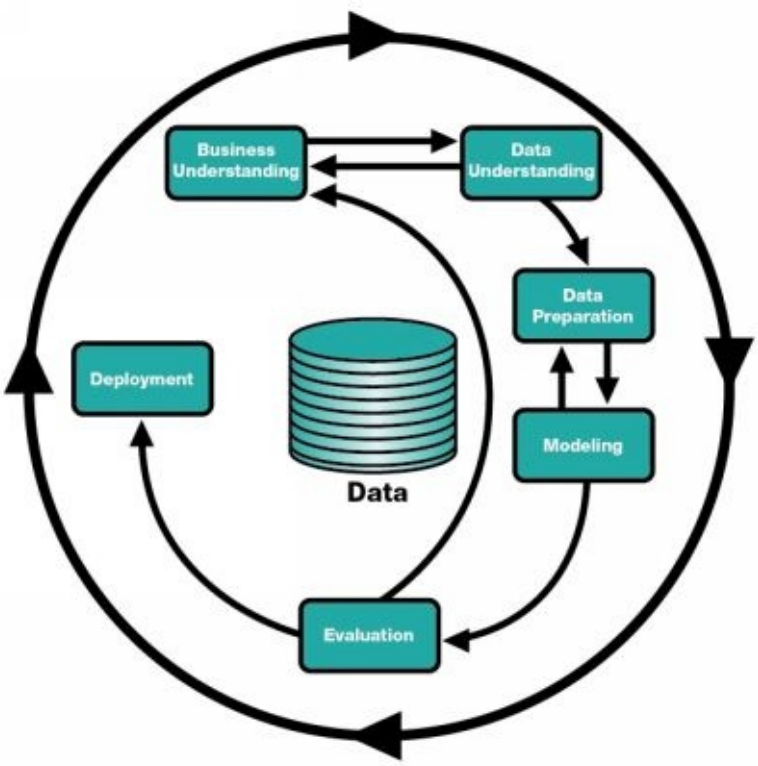


Figure 4.3: CRISP-DM Data Mining cycle

2. *Data Understanding*: A related important step is to understand the data available for mining. One needs to be imaginative in scouring for many elements of data through many sources in helping address the hypotheses to solve a problem. Without relevant data, the hypotheses cannot be tested.
3. *Data Preparation*: The data should be relevant, clean and of high quality. It's important to assemble a team that has a mix of technical and business skills, who understand the domain and the data. Data cleaning can take 60-70% of the time in a data mining project. It may be desirable to continue to experiment and add new data elements from external sources of data that could help improve predictive accuracy.
4. *Modeling*: This is the actual task of running many algorithms using the available data to discover if the hypotheses are supported. Patience is required in continuously engaging with the data until the data yields some good insights. A host of modeling tools and algorithms should be used. A tool could be tried with different options, such as running different decision tree algorithms.
5. *Model Evaluation*: One should not accept what the data says at first. It is better to triangulate the analysis by applying multiple data mining techniques, and conducting many what-if scenarios, to build confidence in the solution. One should evaluate and improve the model's predictive accuracy with more test data. When the accuracy has reached some satisfactory level, then the model should be deployed.

6. *Dissemination and rollout:* It is important that the data mining solution is presented to the key stakeholders, and is deployed in the organization. Otherwise the project will be a waste of time and will be a setback for establishing and supporting a data-based decision-process culture in the organization. The model should be eventually embedded in the organization's business processes.

Conclusion

Data Mining is like diving into the rough material to discover a valuable finished nugget. While the technique is important, domain knowledge is also important to provide imaginative solutions that can then be tested with data mining. The business objective should be well understood and should always be kept in mind to ensure that the results are beneficial to the sponsor of the exercise.

Review Questions

1. What is data mining? What are supervised and unsupervised learning techniques?
2. Describe the key steps in the data mining process. Why is it important to follow these processes?
3. What is a confusion matrix?
4. Why is data preparation so important and time consuming?
5. What are some of the most popular data mining techniques?
6. How is mining Big data different from traditional data mining?

Appendix 1: Hadoop Installation on Amazon Web Services (AWS) Elastic Compute Cluster (EC2)

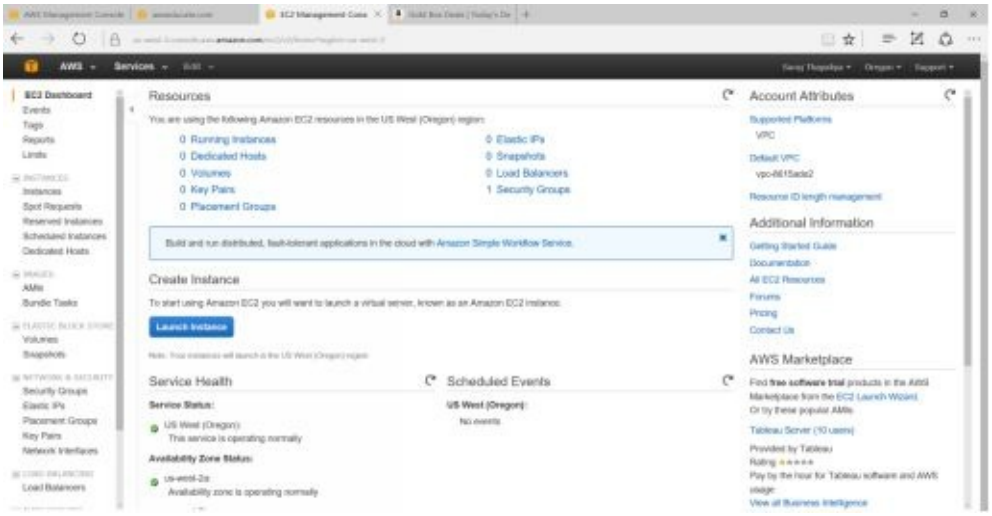
Creating Cluster server on AWS, Install Hadoop from CloudEra

The objective of this tutorial is to set up a big data processing infrastructure using cloud computing, and Hadoop and Spark software.

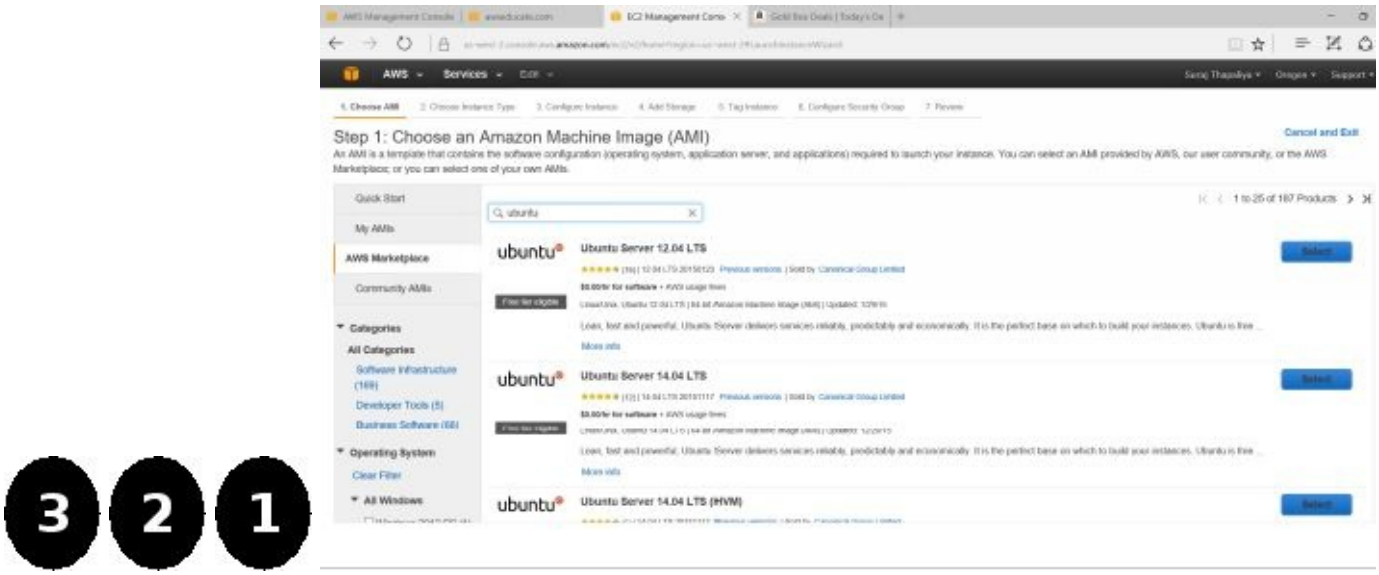
Step 1: Creating Amazon EC2 Servers.

1. Open https://aws.amazon.com/
2. Click on Services
3. Click on EC2

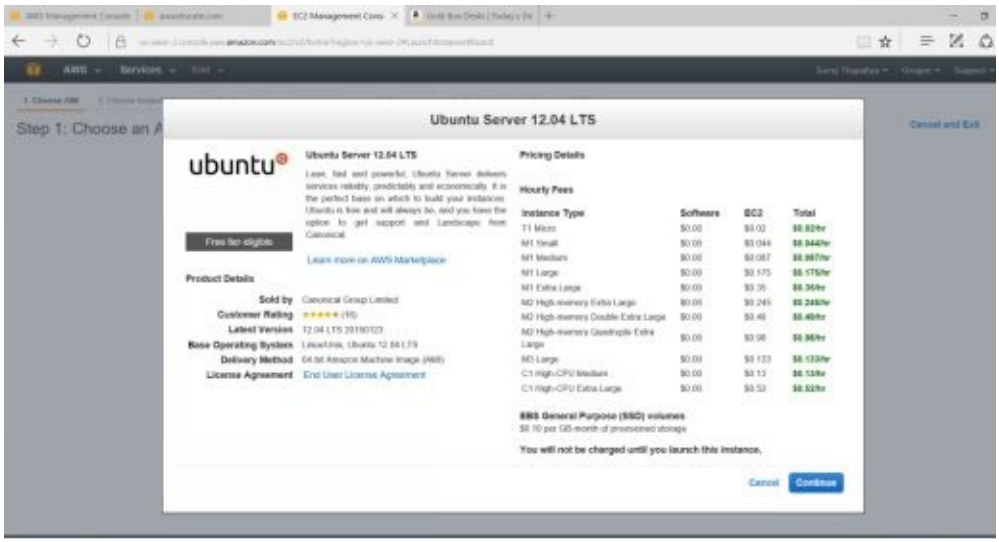
You can see the below result once you click on EC2. If you already have a server you can see the number of running servers, their volume and other information.



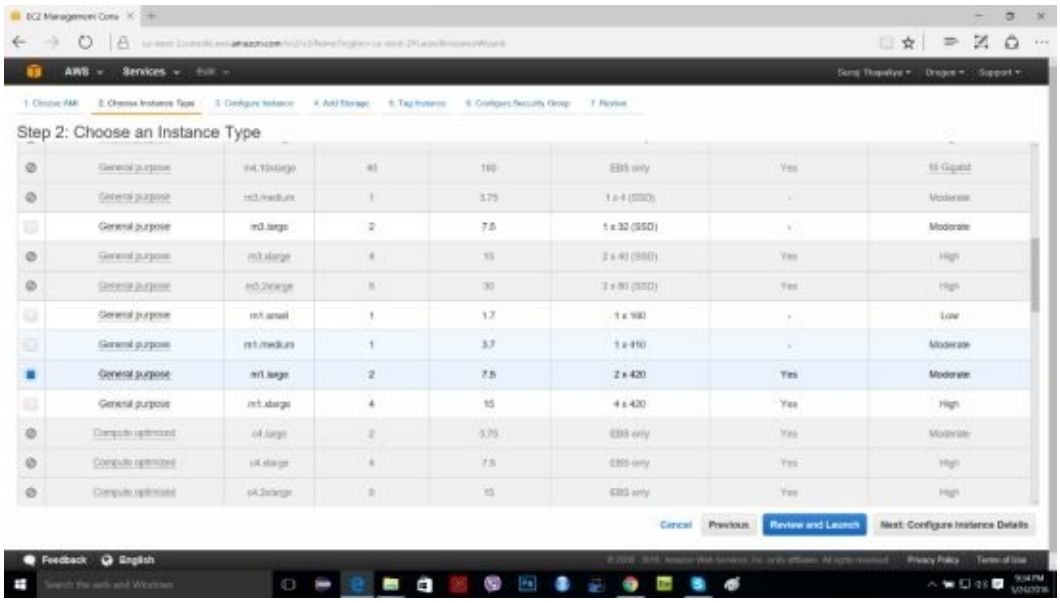
4. Click on Launch Instance Button. 



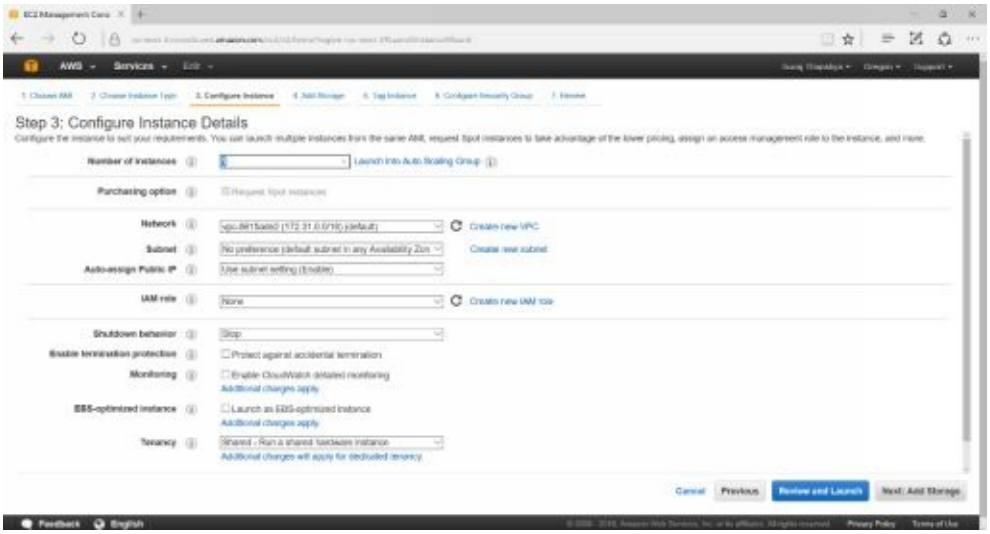
5. Click on **AWS MarketePlace**
6. Type Ubuntu in **search** text box.
7. Click on **Select** button



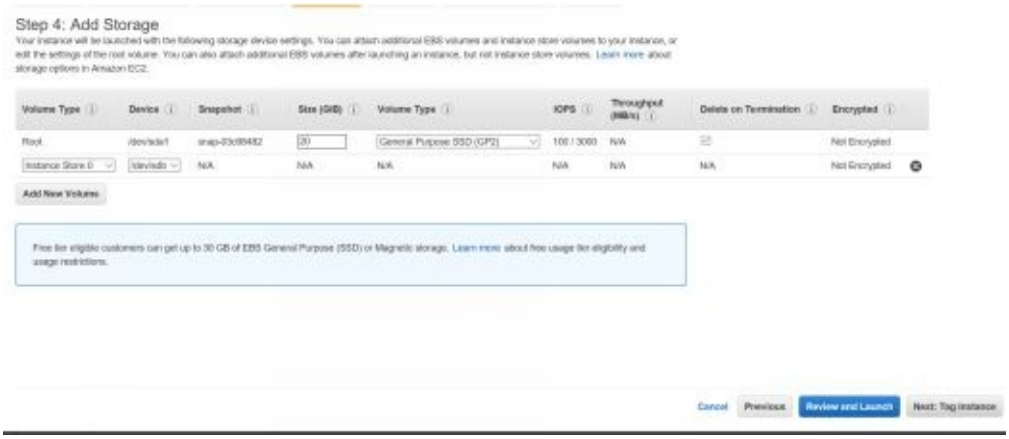
8. Ubuntu is free so you don't have to worry about the service price Click on **Continue** button.



9. Choose **General.purpose m1.large** and click on **Next:Configurare Instance Details**
   (Do not choose the **Micro Instances t1.micro** it is free but it will not able to handle the installation.)

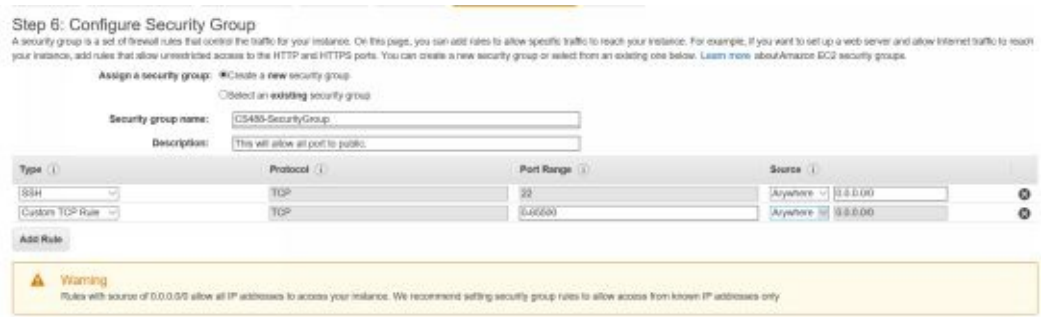


10. **Click on Next: Add Storage**

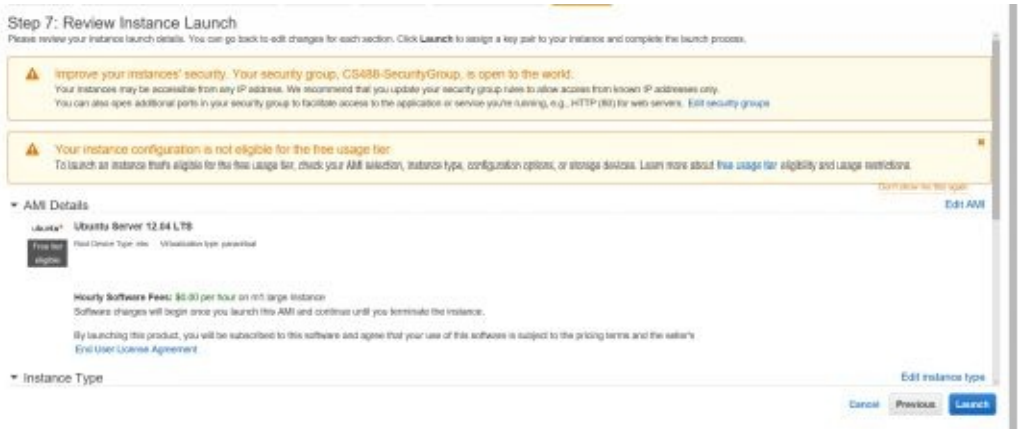


11. Specify the volume size 20GB (Default will be 8 but it will not sufficient) and Click on **Next: Tag Instance**

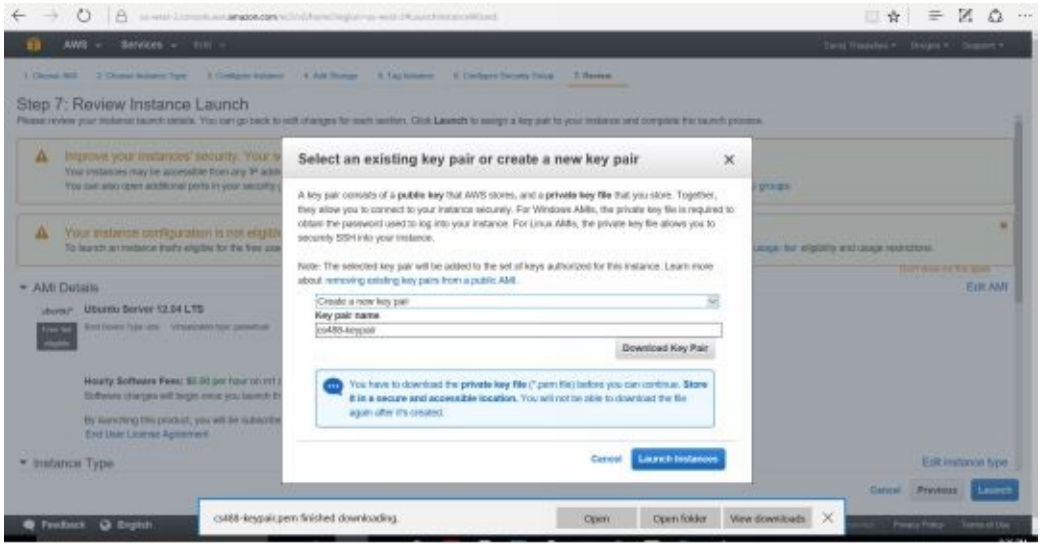
12. Type the name cs488-master (This is for label to know which one is master and slave) and click on Next: Security Group



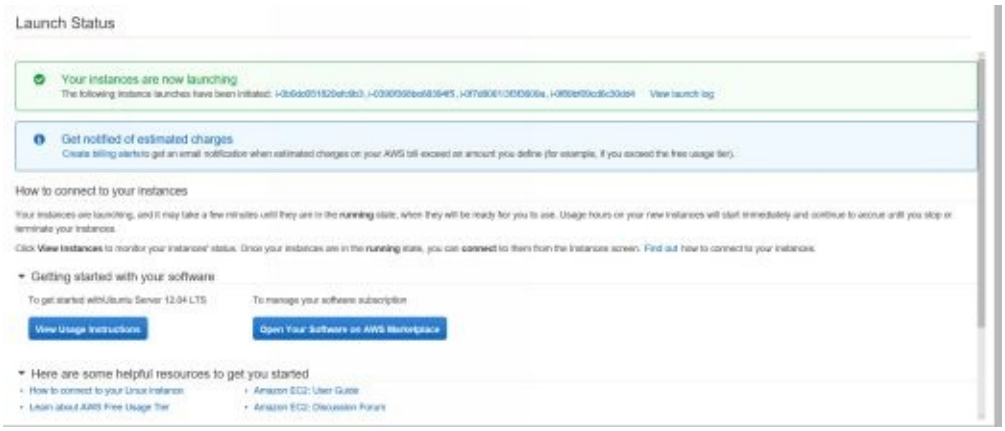
13. We need to open our server to the world including most of the port cause cloudera need to add more port.
    Specify the group name
    Type: Choose Custom TCP Rule
    Port Range 0-65500
    Source : AnyWhere
    And Click on **Review Instance**



14. The message shows the warning this is only that we open our server to world, So ignore it for now. Click on **Launch** button.



15. Type the key pair name and Click on Download Key Pair button (remember the location of downloaded file we need this file to log in to the server.) and Click on Launch Instances.



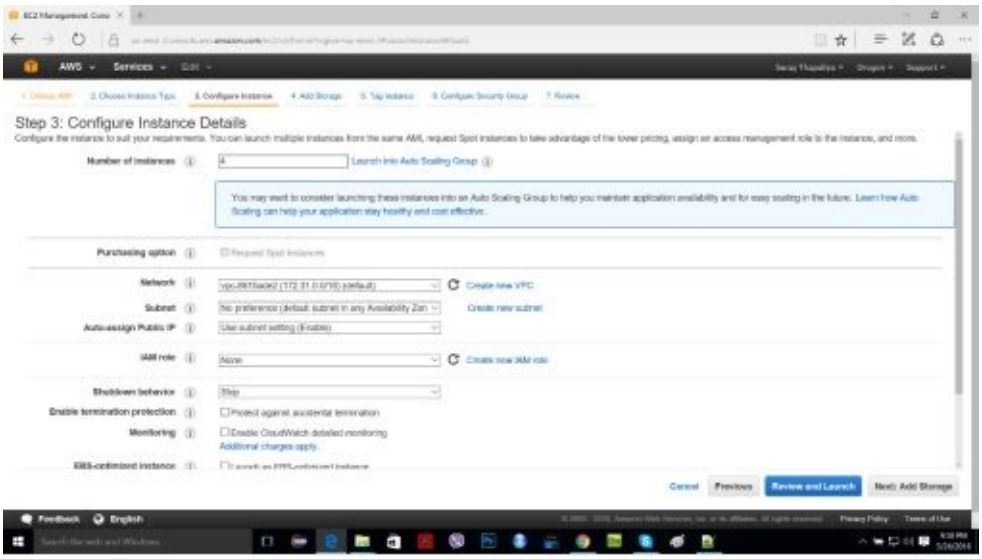
16. Now the master server is created.

Now, we need four more servers to make the clustering for that we don't need to do these process four times. We just increase the value of no of instance we need and we got the 4 servers.

Now we are going to launch 4 more server which is slaves.

Please repeat step 4-9

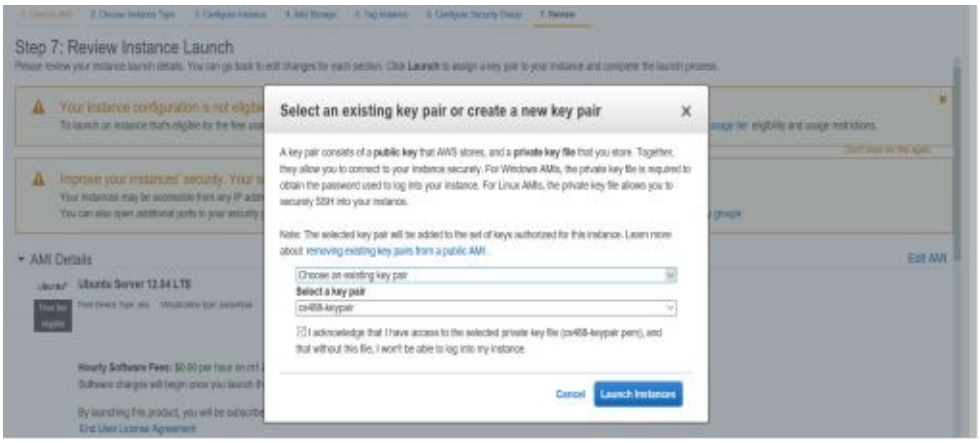
Go to amazon market place, choose Ubuntu, select the instance type (General.purpose)



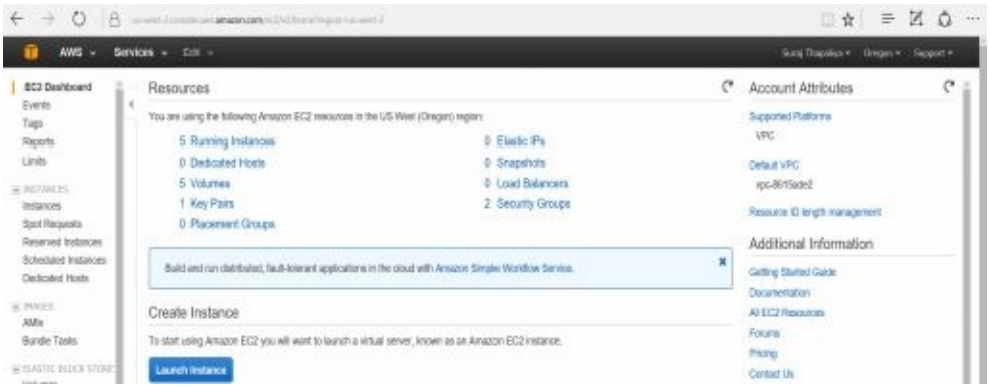
17. Type 4 in Number of Instances. Which will create the 4 more server for us.

18. Name the server cs488-slave

19. Select the previous created security group.



20. It is important that you need to choose the existing key pair for these server too.

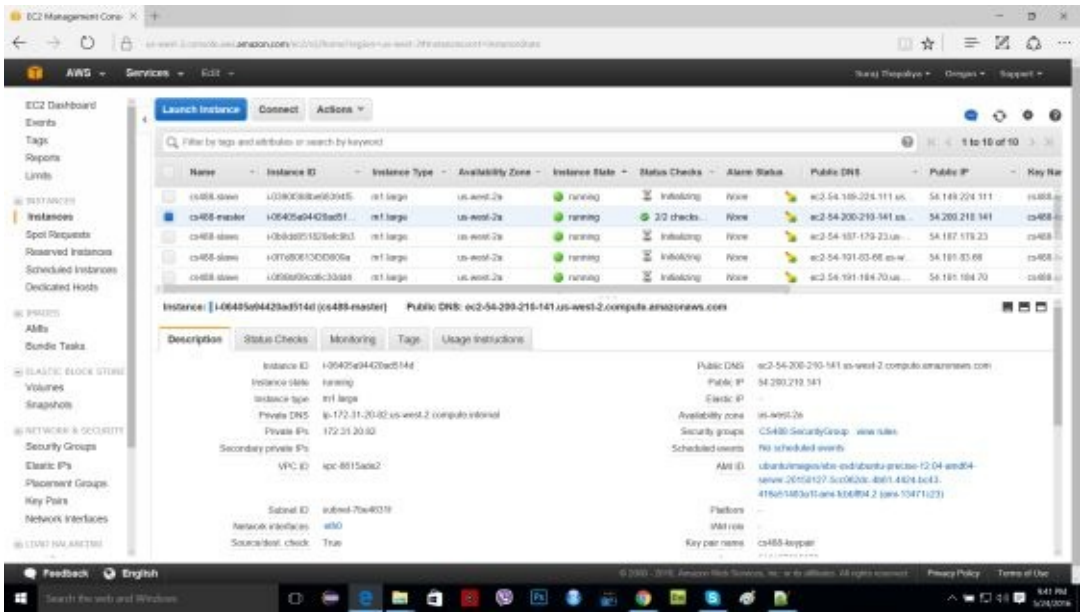


If everything goes well, you can see have 5 instances, 5 volumes, 1 key pair, 1 or 2 security groups.

We are now successfully created 5 servers.

Step 2: Connecting server and installing required Cloudera distribution of Hadoop

First of all take a note for all your server details, IP Address, DNS address. Master and slaves.



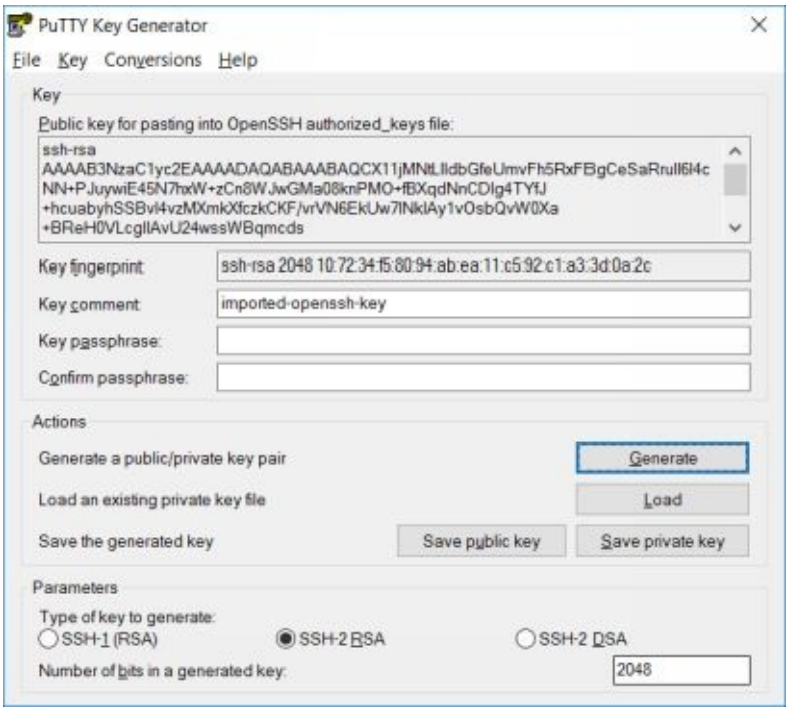
Master Public DNS Address: ec2-54-200-210-141.us-west-2.compute.amazonaws.com
Master Private IP Address: 172.31.20.82

Slave 1 Private IP: 172.31.26.245
Slave 2 Private IP: 172.31.26.242
Slave 3 Private IP: 172.31.26.243
Slave 4 Private IP: 172.31.26.244

Once you have these in recorded, you can connect to the server. If you are using linux as operating system you can use ssh command from terminal to connect it.

Connecting the server (Windows)

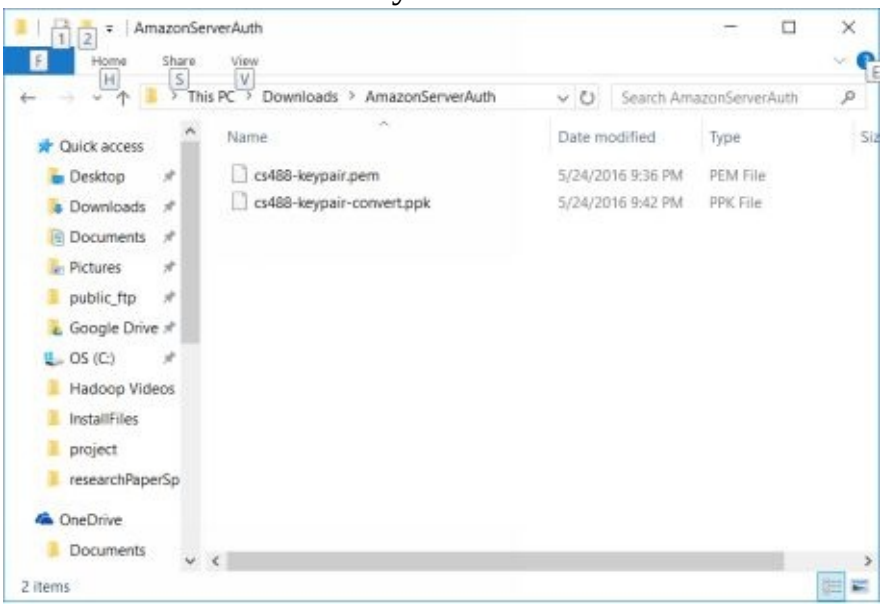
1. Download the ssh software (Putty) (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html)
   Also download puttygen to convert our authentication file .pem to .ppk



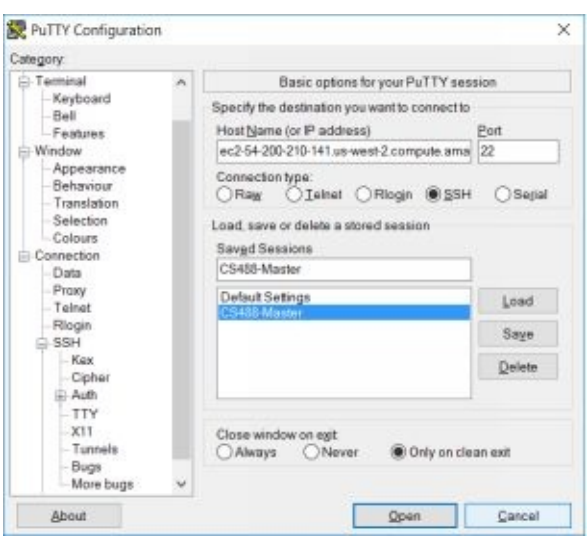
2. Open puttygen load the authentication file

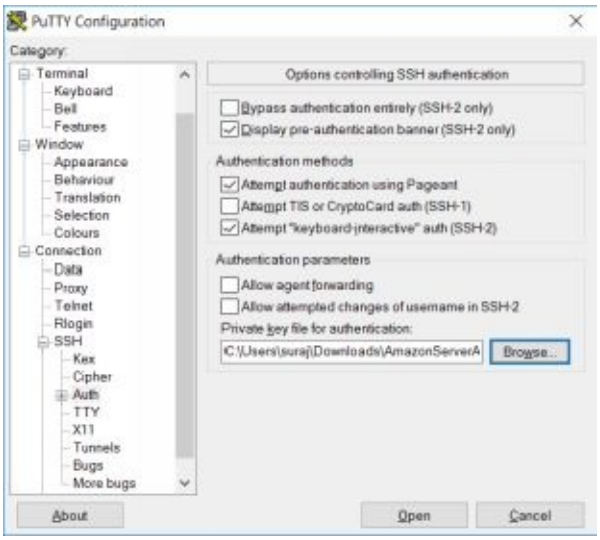
   Click on Save Private Key

Original File from amazon	Convert from puttygen

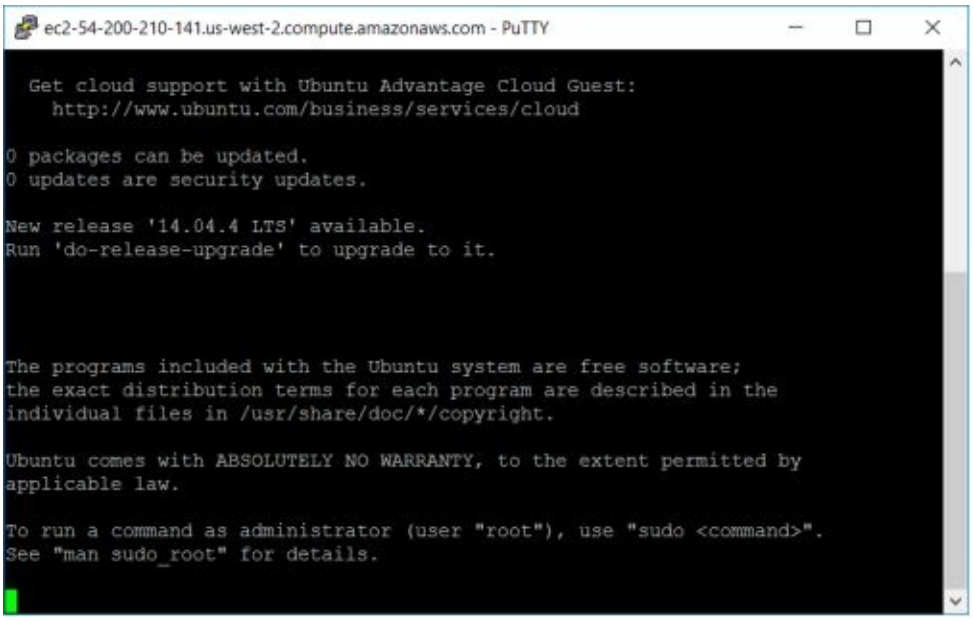


3. Open Putty type the master public dns address in host name and than click on SSH from left panel > Click on Auth >> Select the recent converted authentication file (.ppk) and finally click on Open button.

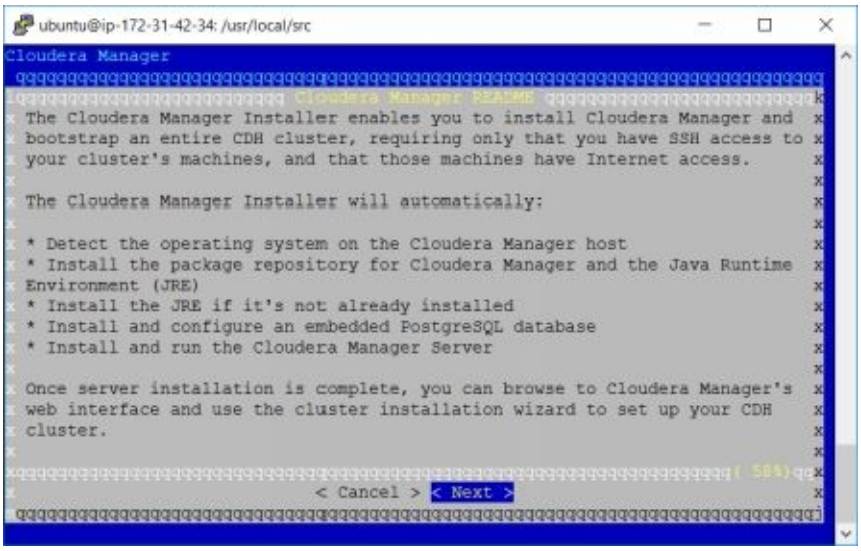




4. Now you will able to connect the server please type “ubuntu” the default username to login to the system.

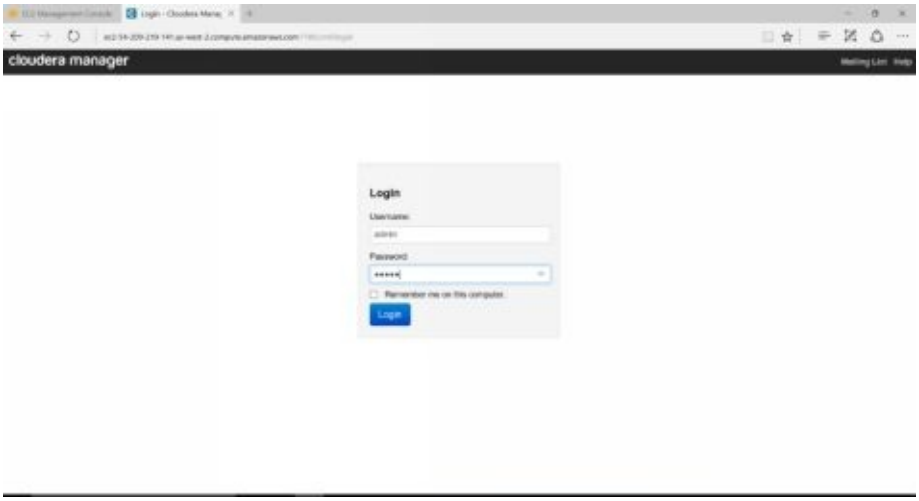


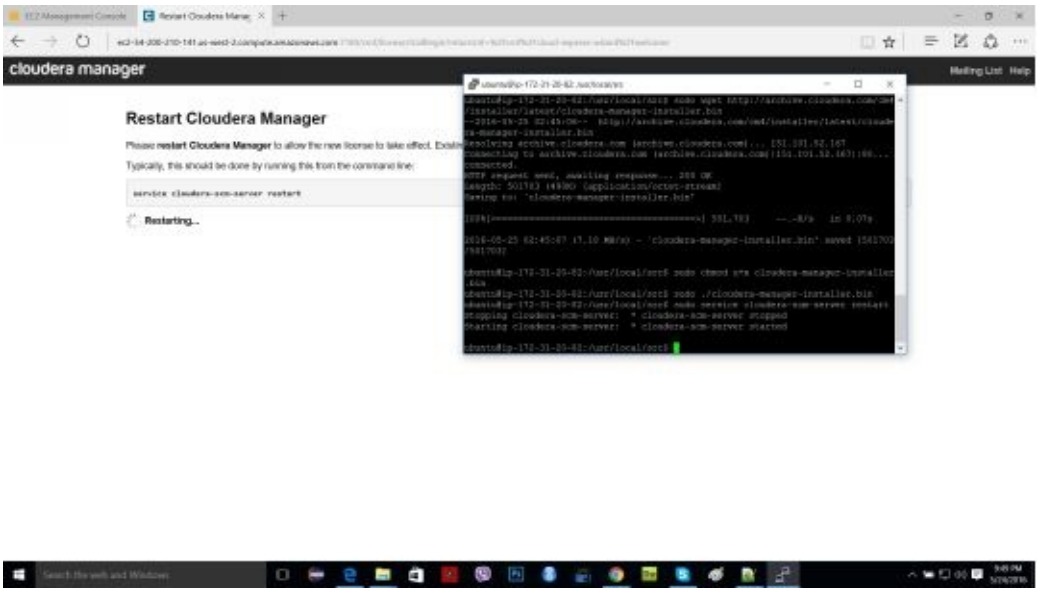
5. Once you connect type the following command into the terminal
6. sudo aptitude update
7. cd /usr/local/src/
8. sudo wget http://archive.cloudera.com/cm4/installer/latest/cloudera-manager-installer.bin
9. sudo chmod u+x cloudera-manager-installer.bin
10. sudo ./cloudera-manager-installer.bin



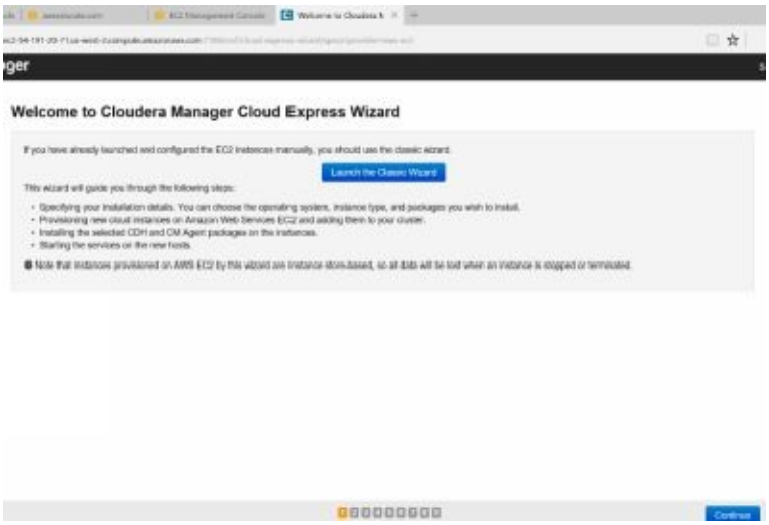
11. There is 4 more step where you click on Next and Yes for license agreement. Once you finish the installation you need to restart the service.
12. sudo service cloudera-scm-server restart

You are now able to connect the cloudera from you browser. The address will be http://<YOUR PUBLIC DNS SERVER>:7180 eg. http://ec2-54-200-210-141.us-west-2.compute.amazonaws.com:7180 and default username and password is admin/admin to login to the system.

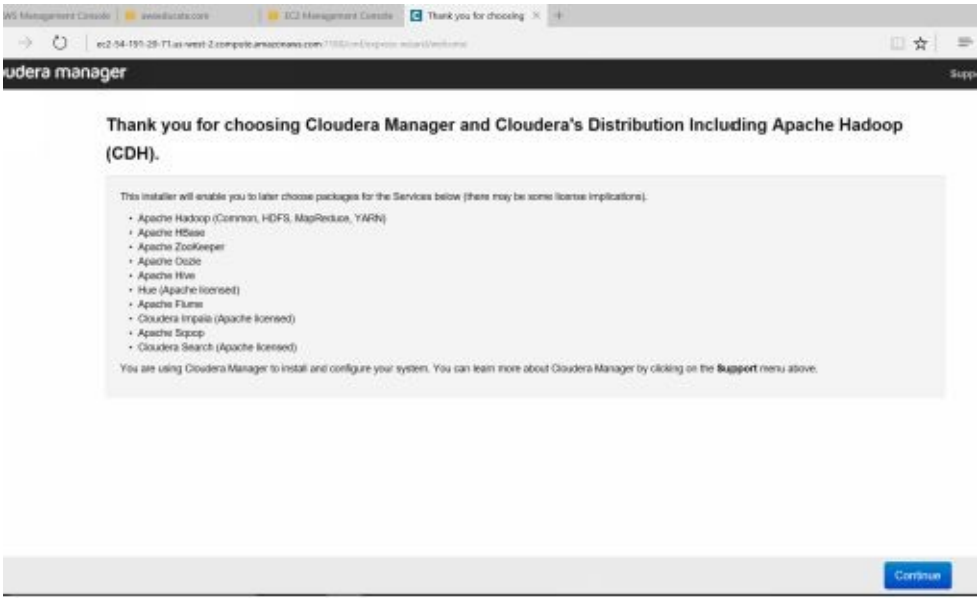




Once restart the server it will open the login screen again. The same username and password (admin/admin) is use to login to the system.

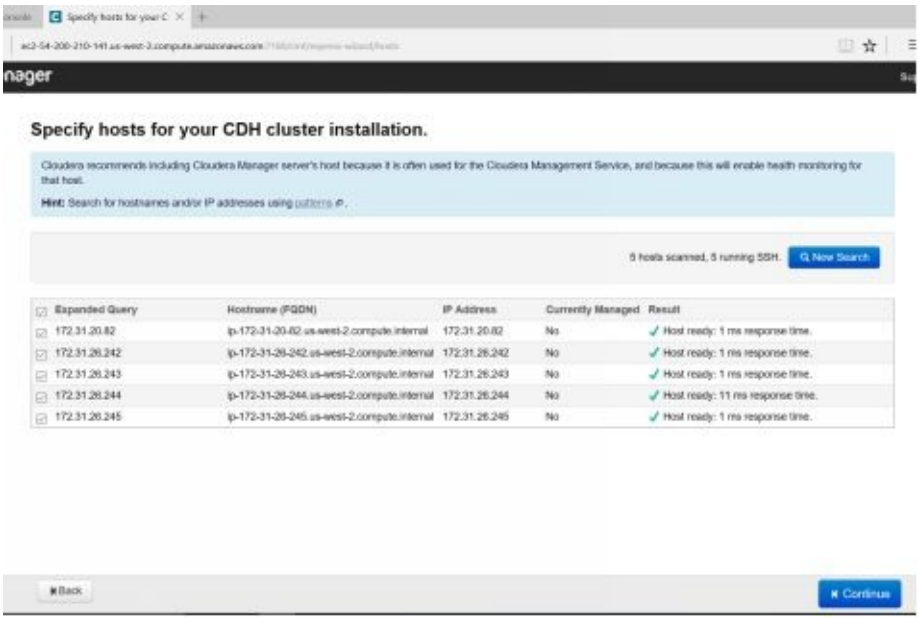


13. Click on **Launch the Classic wizard**

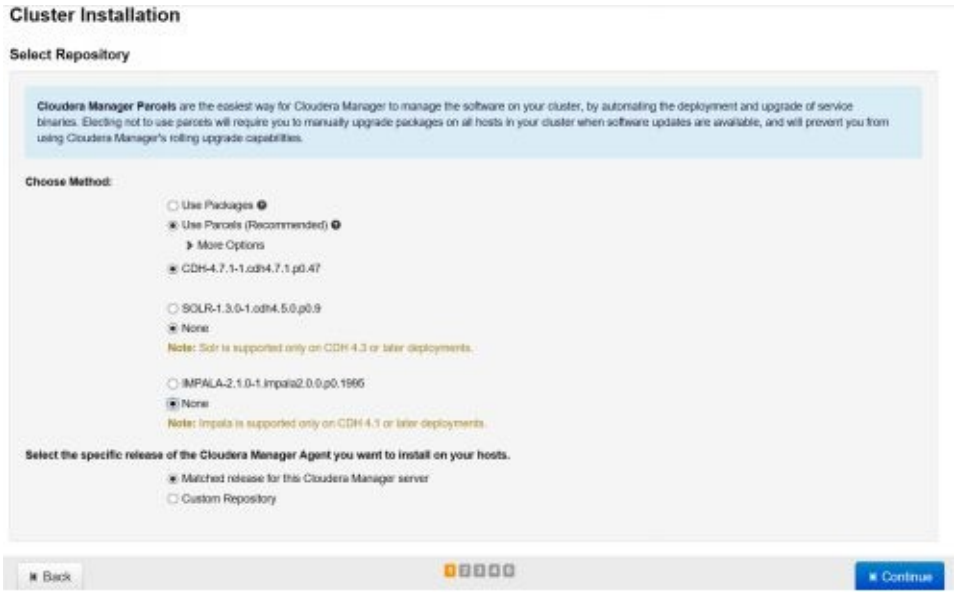


14. Click on Continue

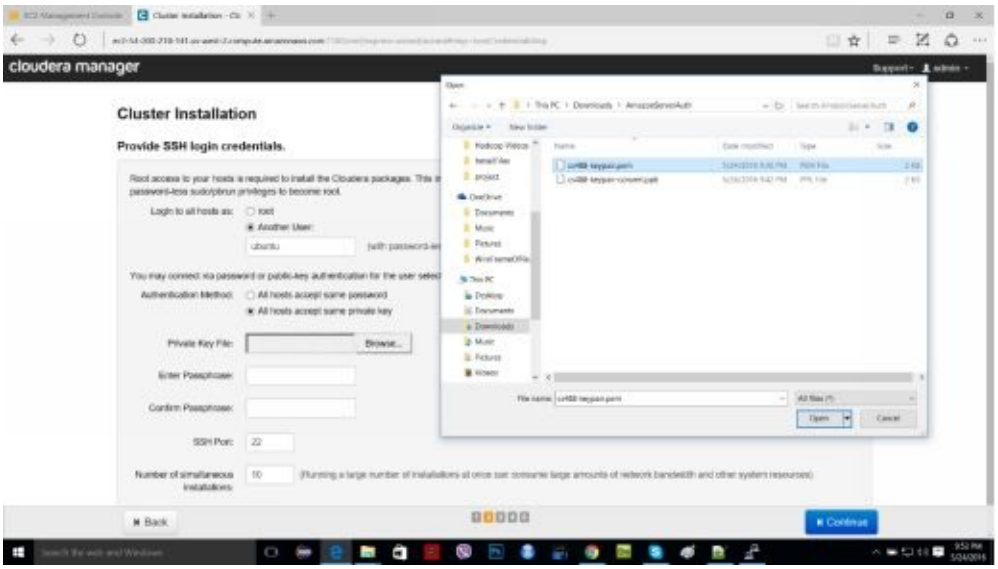
15. Provide all the Private IP address of master and slaves computers and click on Search button.



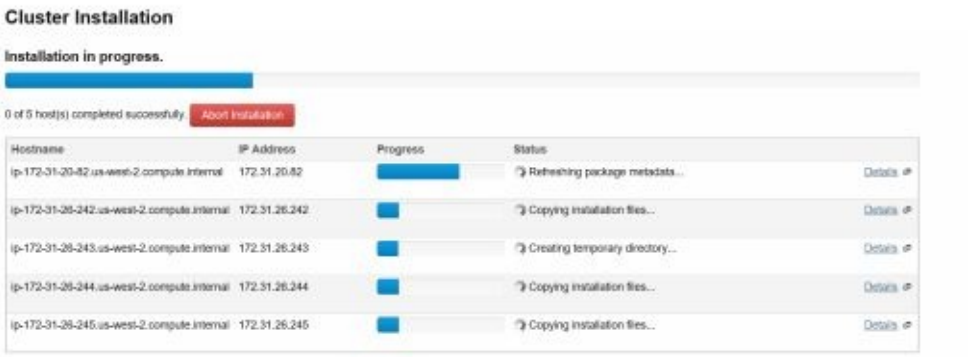
16. Click on Continue button.



17. Choose **None** for SOLR1…. And **None** for IMPAL…. And Click on Continue button.



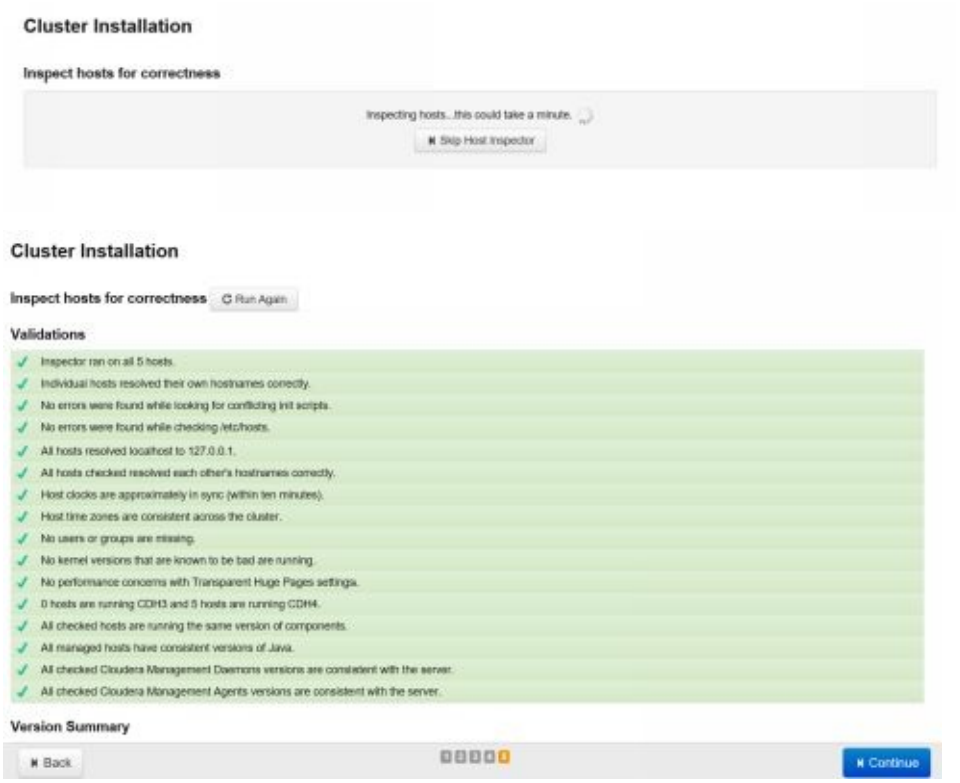
18. Click on **Another User** >> Type “ubuntu” and select **All hosts accept same private key** >> upload the authentication file .pem and click on Continue button.



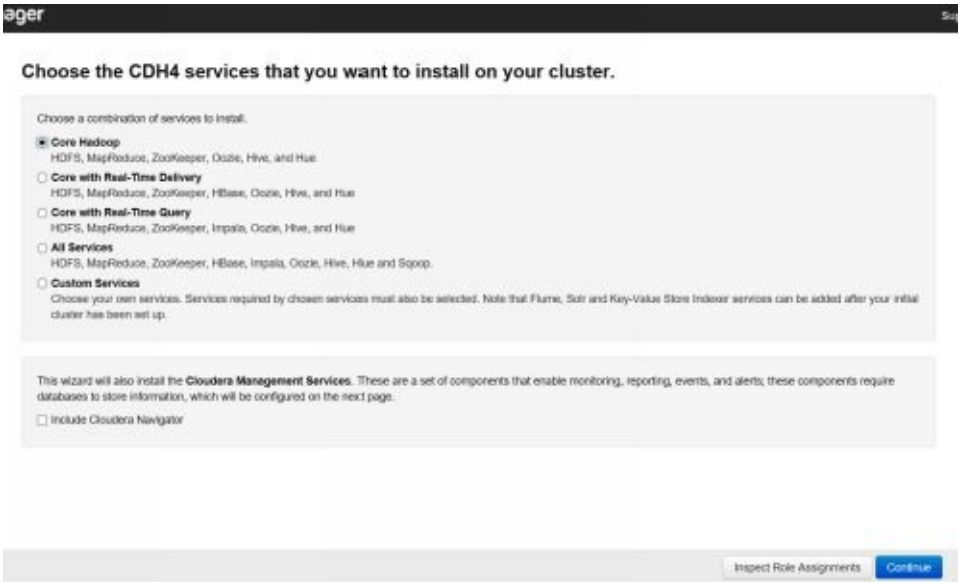
19. Now cloudera will install the software for each of our server.

20. Once the installation is complete click on continue button.

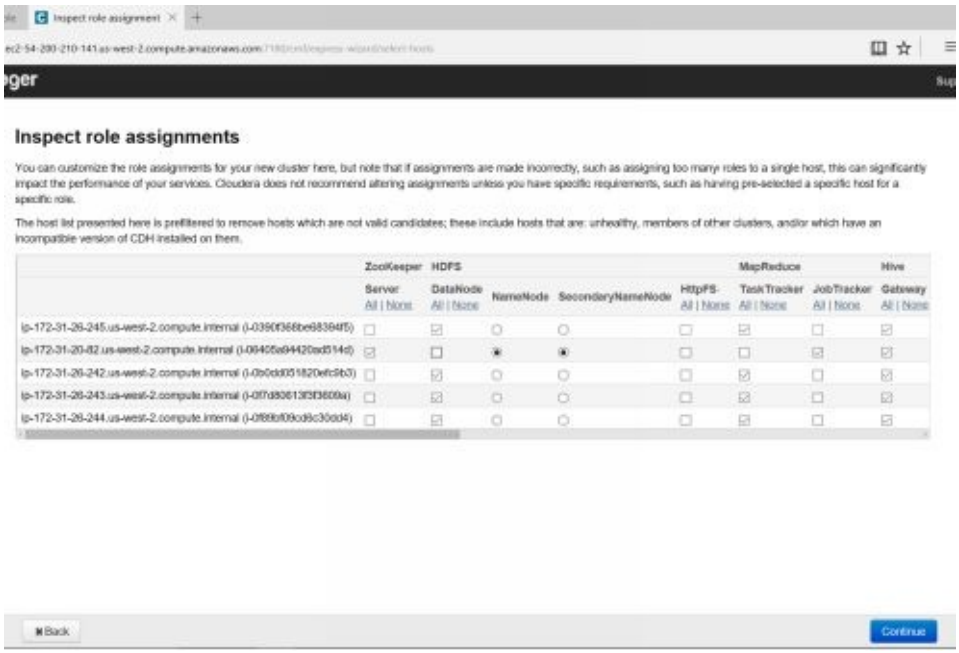
21. Once it reach to 100% click on continue button. Do not disconnect internet nor shut the machine, If the process will not complete that we need to re-create the whole process. Click on continue button.



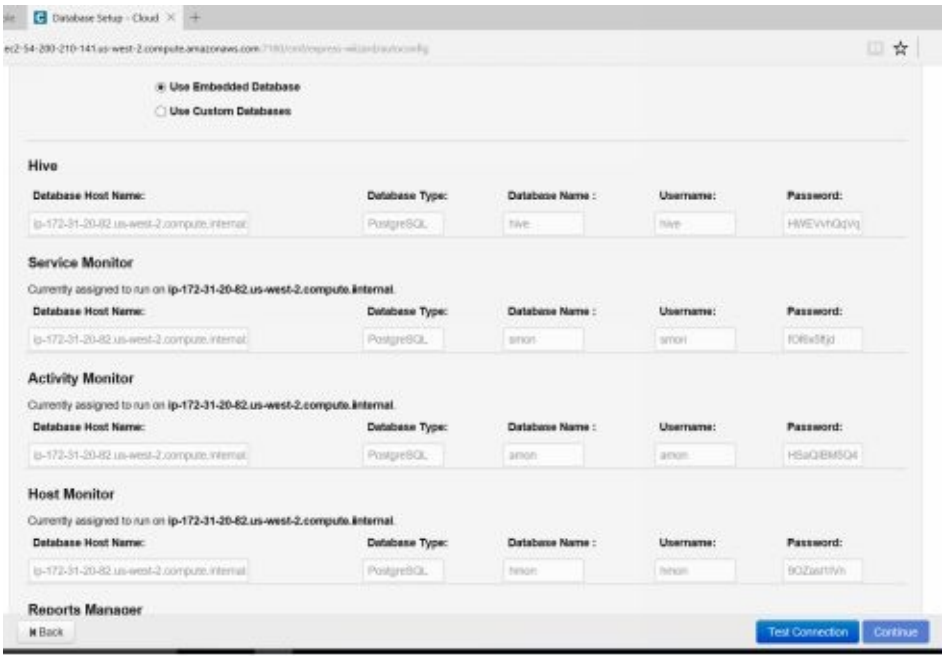
22. Click on Continue.



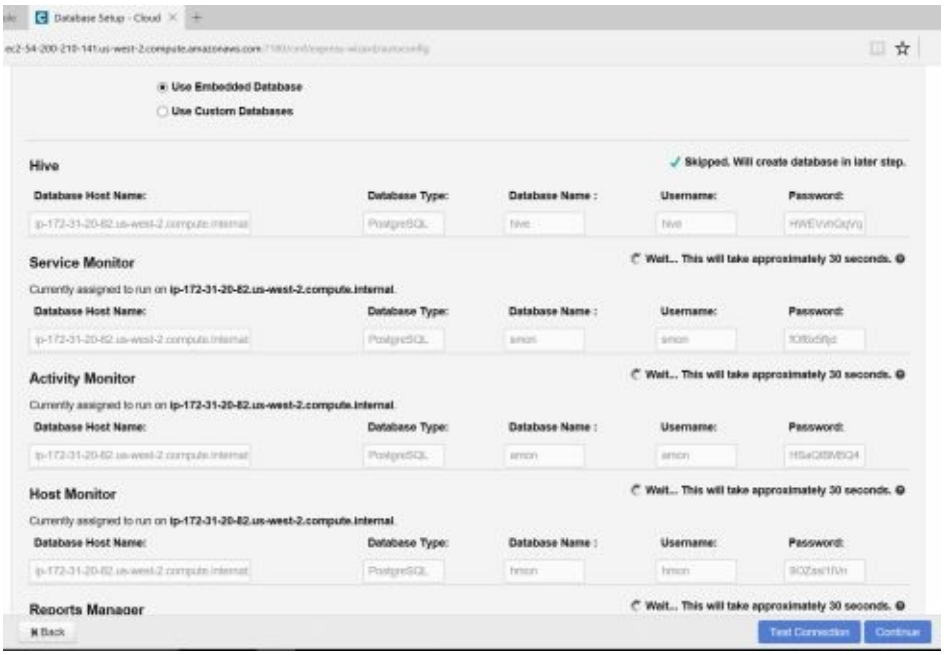
23. Choose Core Hadoop and Click on **Inspect Role Assignments** button

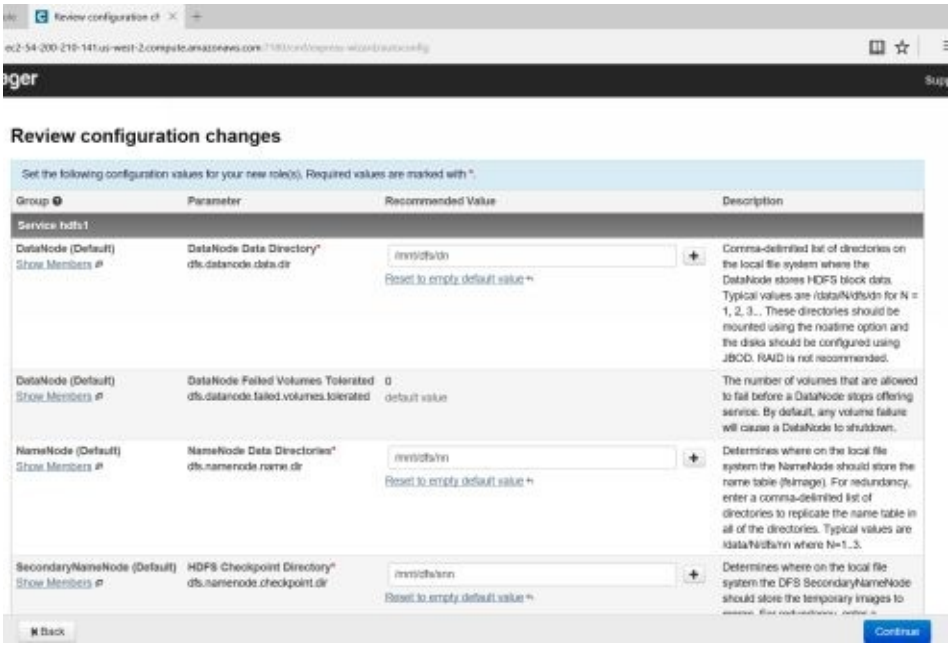


24. Now for you master IP it should have only Name Node selection and unchecked in Data Node. This is important to make the master and slave server.

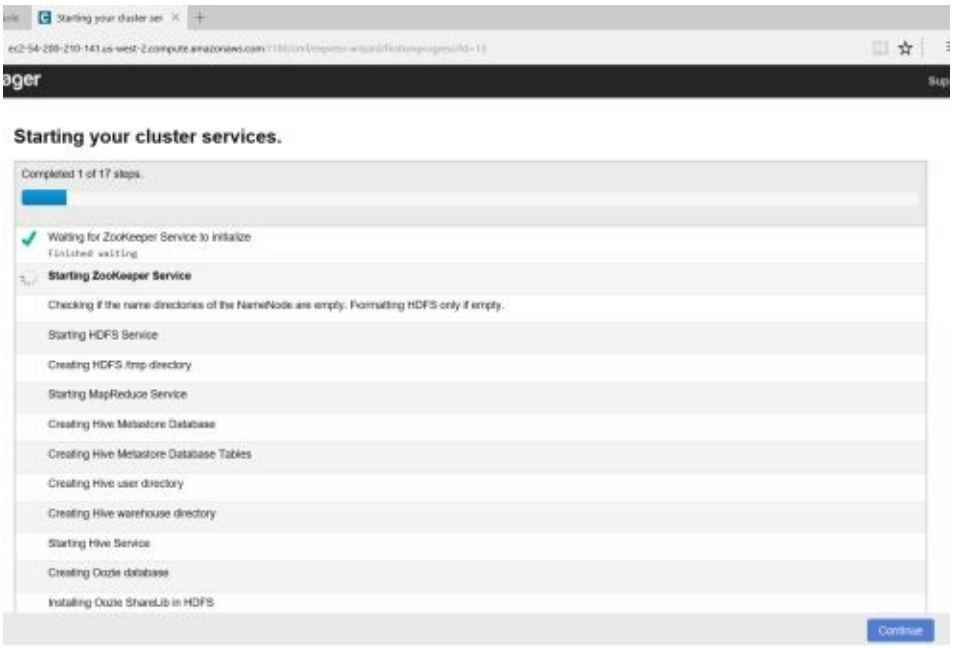


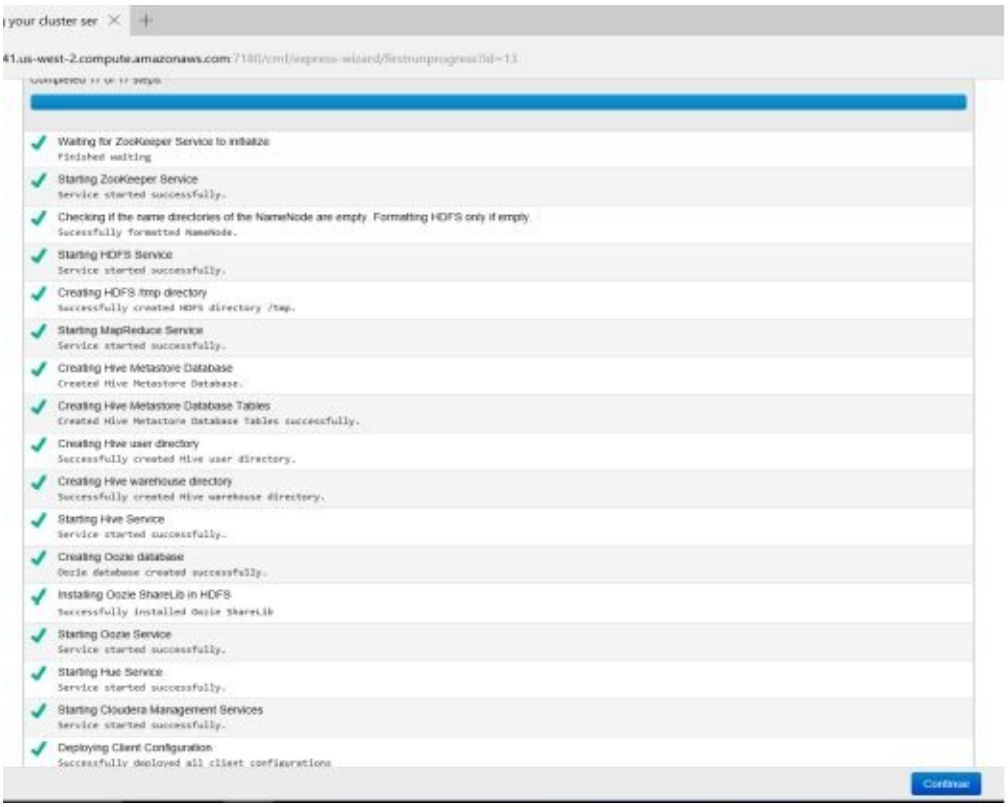
25. Now the cloudera will install the all the services for you future use you can record the username and password of each services. Click on Test Connection



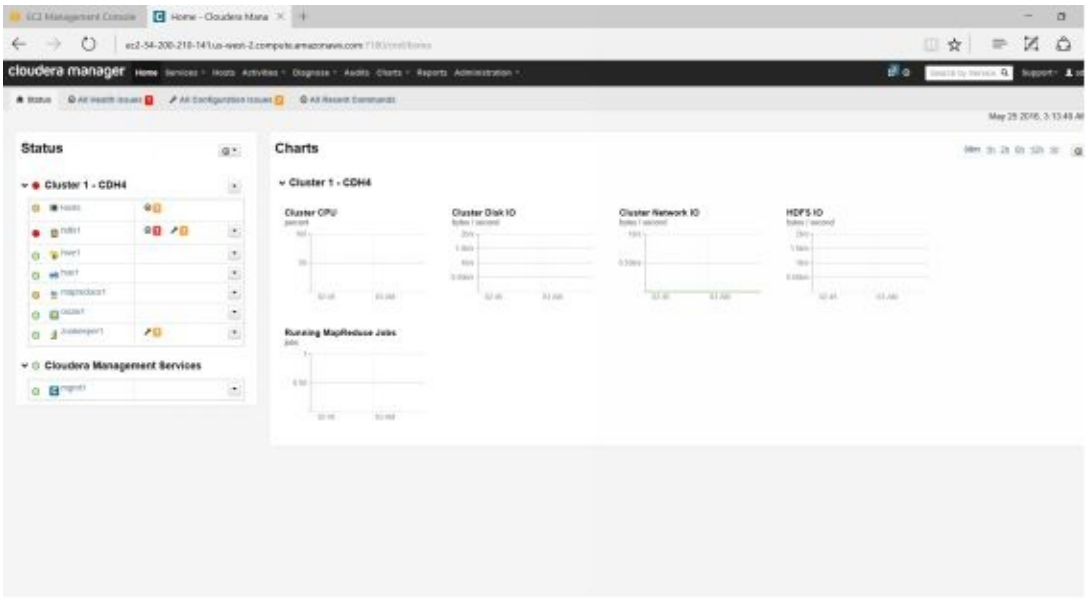


26. Click on Continue





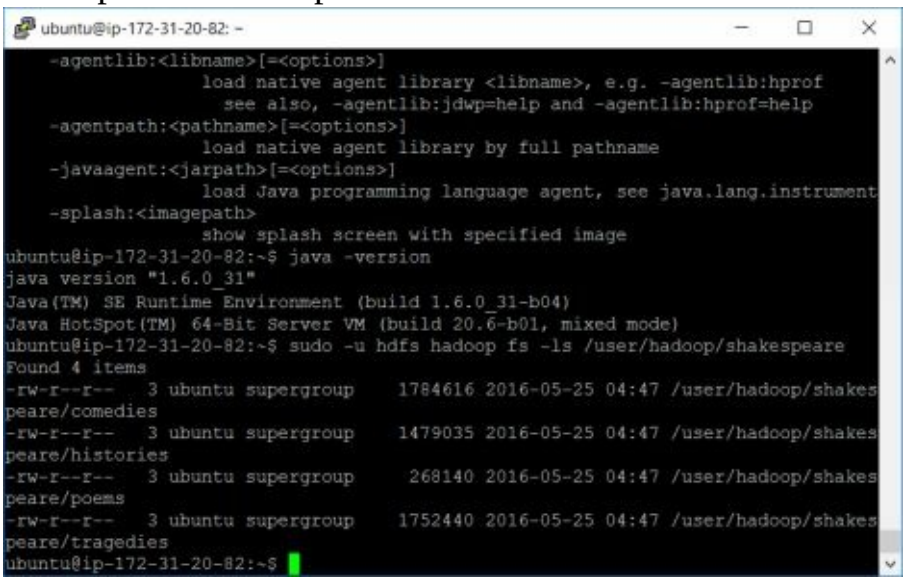
27. Now all the installation is complete you can now have 1 master node 4 data node.



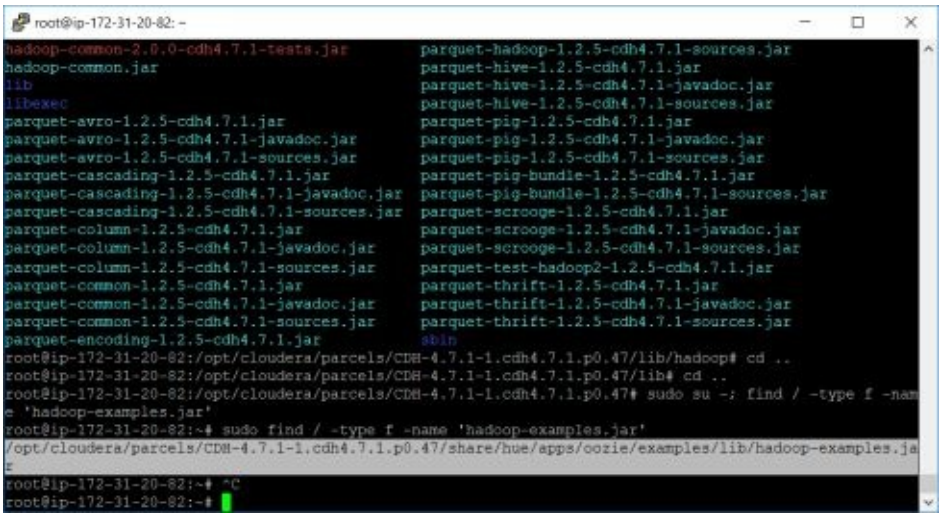
28. You should see the dashboard.

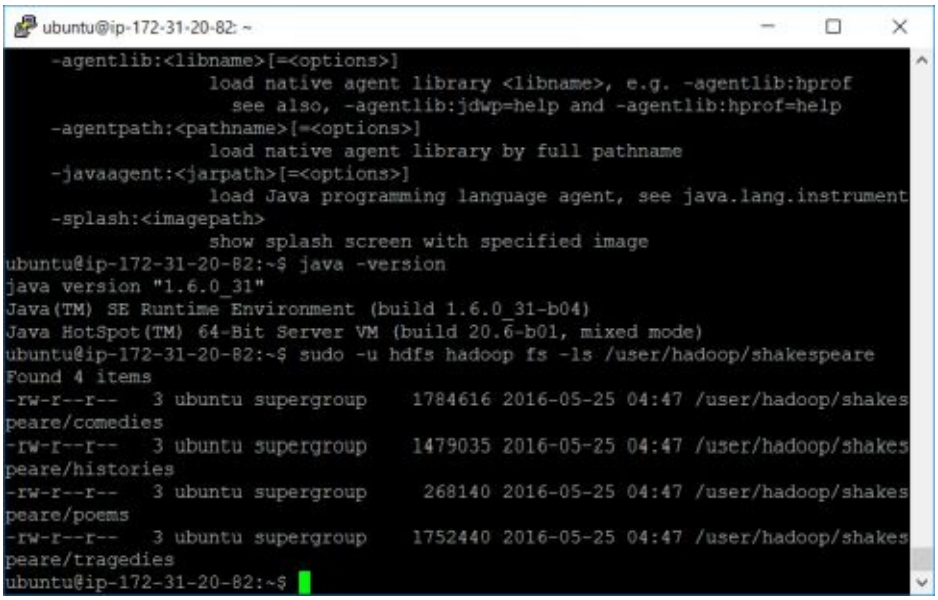
Step 3: WordCount using MapReduce

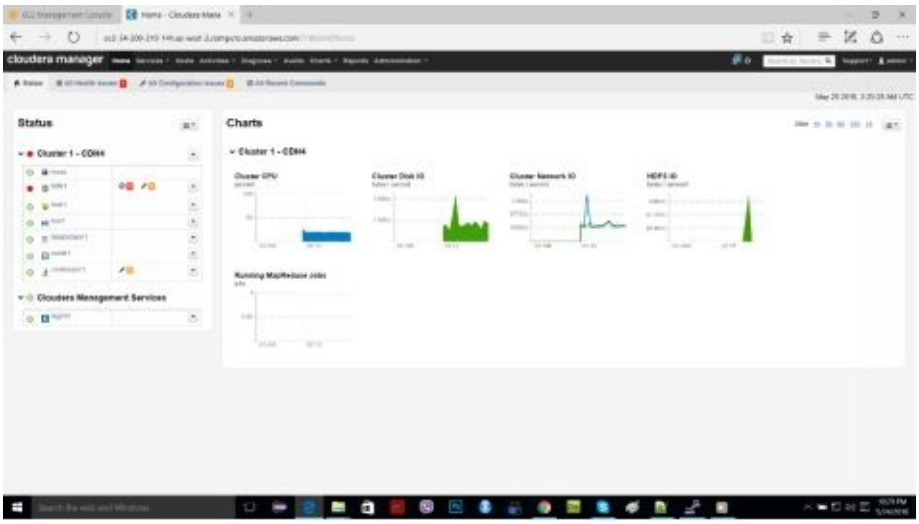
29. Now login to master server from putty.
30. Run the following command
31. cd ~/
32. mkdir code-and-data
33. cd code-and-data
34. sudo wget https://s3.amazonaws.com/learn-hadoop/hadoop-infiniteskills-richmorrow-class.tgz
35. sudo tar -xvzf hadoop-infiniteskills-richmorrow-class.tgz
36. cd data
37. sudo -u hdfs hadoop fs -mkdir /user/ubuntu
38. sudo -u hdfs hadoop fs -chown ubuntu /user/ubuntu
39. hadoop fs -put shakespeare shakespeare-hdfs
40. hadoop version
41. hadoop fs -ls shakespeare-hdfs



42. sudo hadoop jar /opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47/share/hue/apps/oozie/examples/lib/hadoop-examples.jar wordcount shakespeare-hdfs wordcount-output
43. hadoop jar /opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47/share/hue/apps/oozie/examples/lib/hadoop-examples.jar sleep -m 10 -r 10 -mt 20000 -rt 20000


root@ip-172-31-20-82: ~
hadoop-common-2.0.0-cdh4.7.1-tests.jar
hadoop-common.jar
lib
libexec
parquet-avro-1.2.5-cdh4.7.1.jar
parquet-avro-1.2.5-cdh4.7.1-javadoc.jar
parquet-avro-1.2.5-cdh4.7.1-sources.jar
parquet-cascading-1.2.5-cdh4.7.1.jar
parquet-cascading-1.2.5-cdh4.7.1-javadoc.jar
parquet-cascading-1.2.5-cdh4.7.1-sources.jar
parquet-column-1.2.5-cdh4.7.1.jar
parquet-column-1.2.5-cdh4.7.1-javadoc.jar
parquet-column-1.2.5-cdh4.7.1-sources.jar
parquet-common-1.2.5-cdh4.7.1.jar
parquet-common-1.2.5-cdh4.7.1-javadoc.jar
parquet-common-1.2.5-cdh4.7.1-sources.jar
parquet-encoding-1.2.5-cdh4.7.1.jar
parquet-hadoop-1.2.5-cdh4.7.1-sources.jar
parquet-hive-1.2.5-cdh4.7.1.jar
parquet-hive-1.2.5-cdh4.7.1-javadoc.jar
parquet-hive-1.2.5-cdh4.7.1-sources.jar
parquet-pig-1.2.5-cdh4.7.1.jar
parquet-pig-1.2.5-cdh4.7.1-javadoc.jar
parquet-pig-1.2.5-cdh4.7.1-sources.jar
parquet-pig-bundle-1.2.5-cdh4.7.1.jar
parquet-pig-bundle-1.2.5-cdh4.7.1-sources.jar
parquet-scrooge-1.2.5-cdh4.7.1.jar
parquet-scrooge-1.2.5-cdh4.7.1-javadoc.jar
parquet-scrooge-1.2.5-cdh4.7.1-sources.jar
parquet-test-hadoop2-1.2.5-cdh4.7.1.jar
parquet-thrift-1.2.5-cdh4.7.1.jar
parquet-thrift-1.2.5-cdh4.7.1-javadoc.jar
parquet-thrift-1.2.5-cdh4.7.1-sources.jar
sbin
root@ip-172-31-20-82:/opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47/lib/hadoop# cd ..
root@ip-172-31-20-82:/opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47/lib# cd ..
root@ip-172-31-20-82:/opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47# sudo su -; find / -type f -nam
e 'hadoop-examples.jar'
root@ip-172-31-20-82:~# sudo find / -type f -name 'hadoop-examples.jar'
/opt/cloudera/parcels/CDH-4.7.1-1.cdh4.7.1.p0.47/share/hue/apps/oozie/examples/lib/hadoop-examples.ja
r
root@ip-172-31-20-82:~# ^C
root@ip-172-31-20-82:~#


ubuntu@ip-172-31-20-82: ~
-agentlib:<libname>[=<options>]
load native agent library <libname>, e.g. -agentlib:hprof
see also, -agentlib:jdwp=help and -agentlib:hprof=help
-agentpath:<pathname>[=<options>]
load native agent library by full pathname
-javaagent:<jarpath>[=<options>]
load Java programming language agent, see java.lang.instrument
-splash:<imagepath>
show splash screen with specified image
ubuntu@ip-172-31-20-82:~$ java -version
java version "1.6.0_31"
Java(TM) SE Runtime Environment (build 1.6.0_31-b04)
Java HotSpot(TM) 64-Bit Server VM (build 20.6-b01, mixed mode)
ubuntu@ip-172-31-20-82:~$ sudo -u hdfs hadoop fs -ls /user/hadoop/shakespeare
Found 4 items
-rw-r--r-- 3 ubuntu supergroup 1784616 2016-05-25 04:47 /user/hadoop/shakespeare/comedies
-rw-r--r-- 3 ubuntu supergroup 1479035 2016-05-25 04:47 /user/hadoop/shakespeare/histories
-rw-r--r-- 3 ubuntu supergroup 268140 2016-05-25 04:47 /user/hadoop/shakespeare/poems
-rw-r--r-- 3 ubuntu supergroup 1752440 2016-05-25 04:47 /user/hadoop/shakespeare/tragedies
ubuntu@ip-172-31-20-82:~$


cloudera manager
Status
Charts
Cluster 1 - CDH4
Cloudera Management Services

Appendix 2: Spark Installation and Tutorial

This tutorial will help install Spark and get it running on a standalone machine. It will then help develop a simple analytical application using R language.

Step 1: Verifying Java Installation

Java installation is one of the mandatory things in installing Spark. Try the following command to verify the JAVA version.

```
$java -version
```

If Java is already, installed on your system, you get to see the following response −

```
java version “1.7.0_71”
Java(TM) SE Runtime Environment (build 1.7.0_71-b13)
Java HotSpot(TM) Client VM (build 25.0-b02, mixed mode)
```

In case you do not have Java installed on your system, then Install Java before proceeding to next step.

Step 2: Verifying Scala installation

Verify Scala installation using following command.

```
$scala -version
```

If Scala is already installed on your system, you get to see the following response −

```
Scala code runner version 2.11.6 — Copyright 2002-2013, LAMP/EPFL
```

In case you don’t have Scala installed on your system, then proceed to next step for Scala installation.

Step 3: Downloading Scala

Download the latest version of Scala by visit the following link Download Scala. For this tutorial, we are using scala-2.11.6 version. After downloading, you will find the Scala tar file in the download folder.

Step 4: Installing Scala

Follow the below given steps for installing Scala.

Extract the Scala tar file

Type the following command for extracting the Scala tar file.

```
$ tar xvf scala-2.11.6.tgz
```

Move Scala software files

Use the following commands for moving the Scala software files, to respective directory **(/usr/local/scala)**.

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv scala-2.11.6 /usr/local/scala
# exit
```

Set PATH for Scala

Use the following command for setting PATH for Scala.

```
$ export PATH = $PATH:/usr/local/scala/bin
```

Verifying Scala Installation

After installation, it is better to verify it. Use the following command for verifying Scala installation.

```
$scala -version
```

If Scala is already installed on your system, you get to see the following response −

```
Scala code runner version 2.11.6 — Copyright 2002-2013, LAMP/EPFL
```

Step 5: Downloading Spark

Download the latest version of Spark. For this tutorial, we are using **spark-1.3.1-bin-hadoop2.6** version. After downloading it, you will find the Spark tar file in the download folder.

Step 6: Installing Spark

Follow the steps given below for installing Spark.

Extracting Spark tar

The following command for extracting the spark tar file.

```
$ tar xvf spark-1.3.1-bin-hadoop2.6.tgz
```

Moving Spark software files

The following commands for moving the Spark software files to respective directory **(/usr/local/spark)**.

```
$ su –
Password:
# cd /home/Hadoop/Downloads/
# mv spark-1.3.1-bin-hadoop2.6 /usr/local/spark
# exit
```

Setting up the environment for Spark

Add the following line to ~/**.bashrc** file. It means adding the location, where the spark software file are located to the PATH variable.

```
export PATH = $PATH:/usr/local/spark/bin
```

Use the following command for sourcing the ~/.bashrc file.

```
$ source ~/.bashrc
```

Step 7: Verifying the Spark Installation

Write the following command for opening Spark shell.

```
$spark-shell
```

If spark is installed successfully then you will find the following output.

```
Spark assembly has been built with Hive, including Datanucleus jars on classpath
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
15/06/04 15:25:22 INFO SecurityManager: Changing view acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: Changing modify acls to: hadoop
15/06/04 15:25:22 INFO SecurityManager: SecurityManager: authentication disabled;
   ui acls disabled; users with view permissions: Set(hadoop); users with modify permissions: Set(hadoop)
15/06/04 15:25:22 INFO HttpServer: Starting HTTP Server
15/06/04 15:25:23 INFO Utils: Successfully started service 'HTTP class server' on port 43292.
Welcome to Spark version 1.4.0
Using Scala version 2.10.4 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_71)
Type in expressions to have them evaluated.
Spark context available as sc
scala>
```

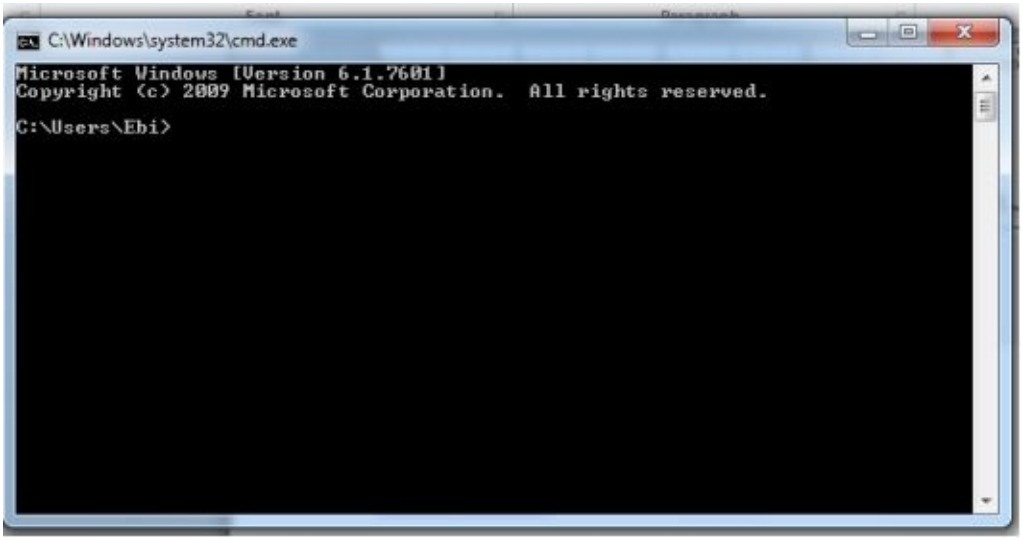
Here you can see the video:

How to install Spark

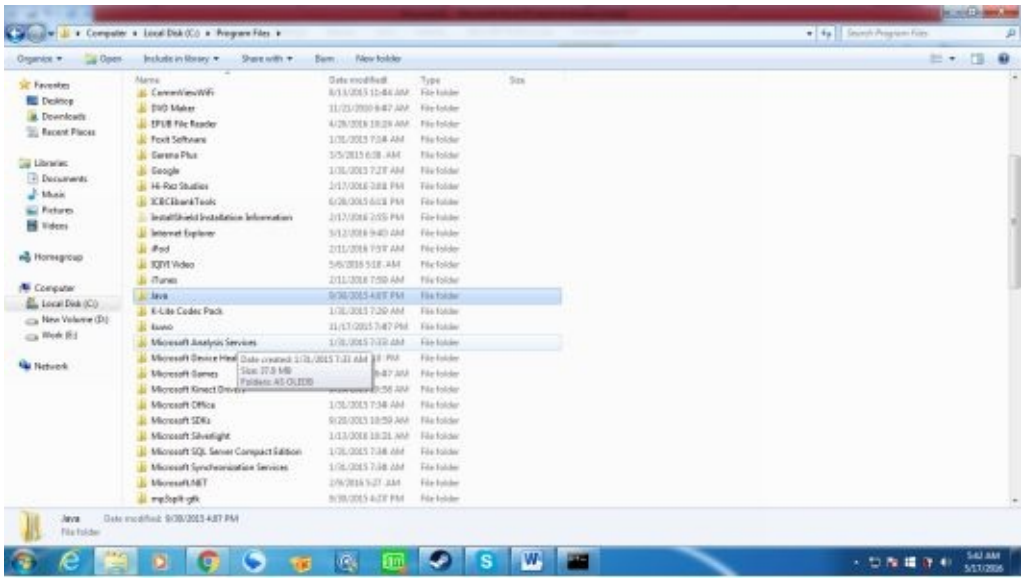
You might encounter "file specified not found error" when you are first installing SPARK stand alone:

To fix this you have to set up your JAVA_HOME

Step 1: Start->run->command prompt(cmd)

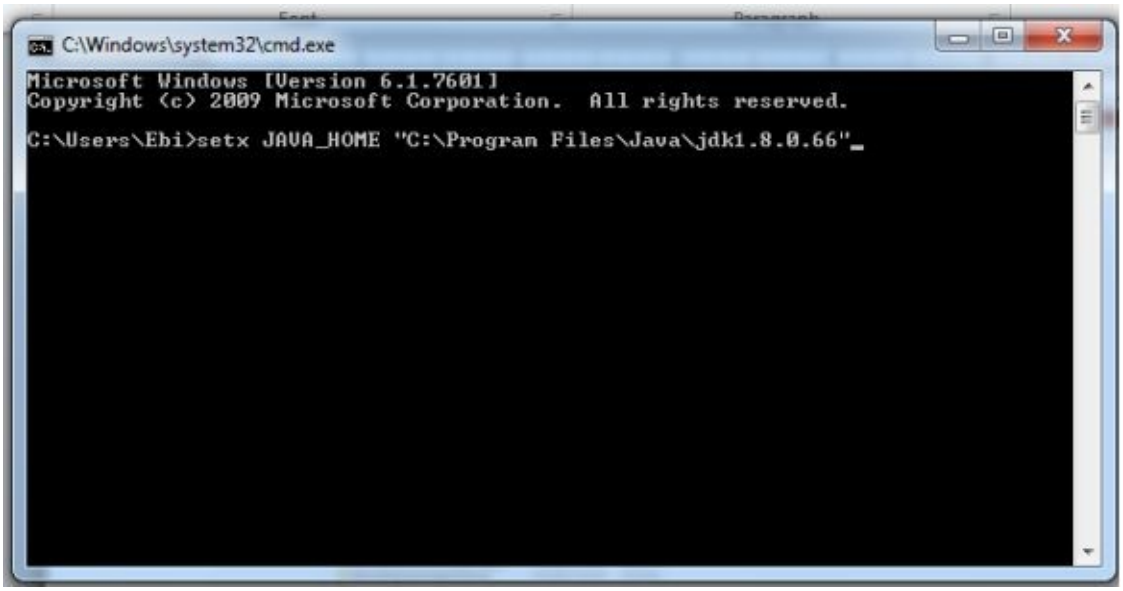


Step 2: Determine where is your JDK is located, by default it is in your C:\program files

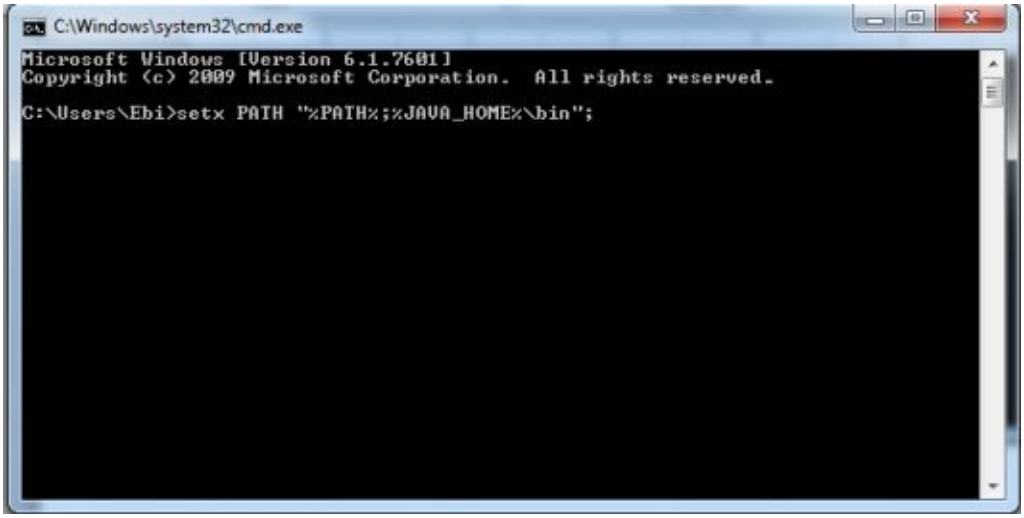


Step 3 Select your JDK to use in my case, I will use my JDK_8

Copy the directory to your clipboard and go to your CMD. And press enter.



Step 4: Add it to general PATH



And press enter.

Now go to your spark folder and go to BIN\spark_shell

You have installed spark let’s try to use it.

Step 8: Application: WordCount in Scala

Now we will do an example of word count in Scala:

```
text_file = sc.textFile("hdfs://…")
counts = text_file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://…")
```

NOTE: If you are working on a stand-alone Spark:

```
This counts.saveAsTextFile("hdfs://…") command will give you an error of NullPointerException.
Solution: counts.coalesce(1).saveAsTextFile()
```

For implementing word cloud we could use R in our spark console:

However, if you click on SparkR straight away you will get an error.

To fix this:

Step 1: Set up the environment variables.

In the PATH Variable add your path : I added -> ;C:\spark-1.5.1-bin-hadoop2.6\spark-1.5.1-bin-hadoop2.6\;C:\spark-1.5.1-bin-hadoop2.6\spark-1.5.1-bin-hadoop2.6\sbin;C:\spark-1.5.1-bin-hadoop2.6\spark-1.5.1-bin-hadoop2.6\bin

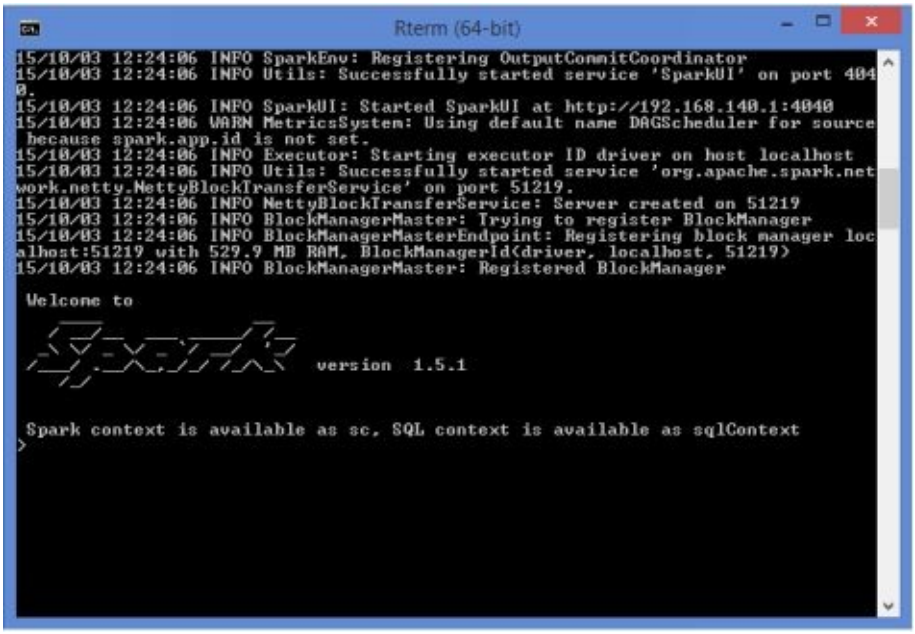
Step 2: Install R software and Rstudio. Then add the path of R software path to the PATH variable.

I added this to my existing path -> ;C:\Program Files\R\R-3.2.2\bin\x64\ (Remember each path that you add must be separated by semicolon and no spaces please)

Step 3: Run command prompt as an administrator.

Step 4: Now execute the command > "SparkR" from the command prompt. If successful you should see message "Spark context is available … " as seen

below. If you path is not set correctly you can alternatively navigate to the location where you have downloaded SparkR . In my case (C:\spark-1.5.1-bin-hadoop2.6\spark-1.5.1-bin-hadoop2.6\bin) and execute “SparkR” Command.



Step 5: Configuration inside the RStudio to connect to Spark!

Execute the below three commands in Rstudio everytime:

Here we are setting up SPARK_HOME environment variable

Sys.setenv(SPARK_HOME = “C:/spark-1.5.1-bin-hadoop2.6/spark-1.5.1-bin-hadoop2.6”)

Set the library path

.libPaths(c(file.path(Sys.getenv(“SPARK_HOME”),“R”,“lib”), .libPaths()))

Loading the SparkR Libary

library(SparkR)

If you see the below message then you are all set to start working with SparkR

```
> Sys.setenv(SPARK_HOME = "C:/spark-1.5.1-bin-hadoop2.6/spark-1.5.1-bin-hadoop2.6")
> .libPaths(c(file.path(Sys.getenv("SPARK_HOME"),"R","lib"), .libPaths()))
> library(SparkR)

Attaching package: 'SparkR'

The following objects are masked from 'package:stats':

    filter, na.omit

The following objects are masked from 'package:base':

    intersect, rbind, sample, subset, summary, table, transform

>
```

Now let’s Start Coding in R:

lords <- Corpus (DirSource(“temp/”))

To see what’s in that corpus, type the command

inspect(lords)

This should print out contents on the main screen. Next, we need to clean it up. Execute the following in the command line, one line at a time:

lords <- tm_map(lords, stripWhitespace)

lords <- tm_map(lords, tolower)

lords <- tm_map(lords, removeWords, stopwords(“english”))

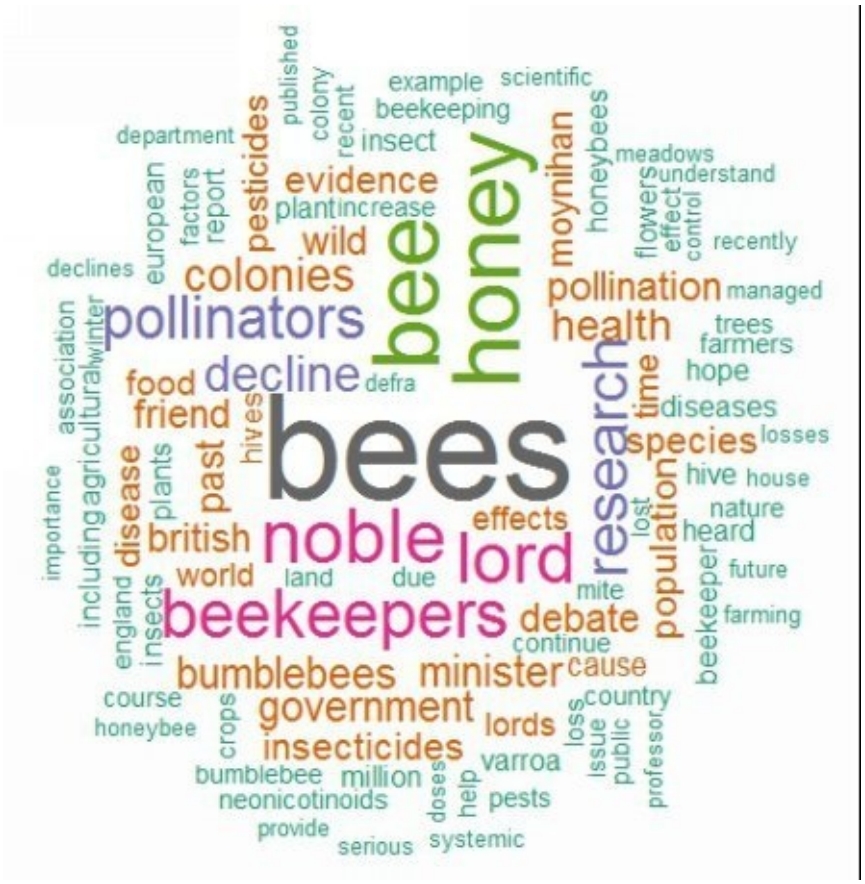
lords <- tm_map(lords, stemDocument)

The **tm_map** function comes with the tm package. The various commands are self-explanatory: strip unnecessary white space, convert everything to lower case (otherwise the wordcloud might highlight capitalised words separately), remove English common words like ‘the’ (so-called ‘stopwords’), and carry out text stemming for the final tidy-up. Depending on what you want to achieve you could also explicitly remove numbers and punctuation with the **removeNumbers** and **removePunctuation** arguments.

It is possible that you may get error messages whilst executing some of the commands, e.g. missing packages. If so install these as outlined above in Step 4, and repeat

If all is well then you should now be ready to create your first wordcloud! Try this:

wordcloud(lords,
scale=c(5,0.5), max.words=100, random.order=FALSE,rot.per=0.35,
use.r.layout=FALSE, colors=brewer.pal(8, “Dark2”))


bees
honey
bee
noble
lord
beekeepers
pollinators
research
decline
colonies
evidence
wild
pollination
health
species
population
bumblebees
minister
government
insecticides
lords
debate
effects
past
friend
food
disease
british
world
land
due
insect
pesticides
plant
increase
moynihan
honeybees
meadows
flowers
understand
effect
control
recently
managed
trees
farmers
hope
diseases
losses
hive
house
nature
heard
future
farming
beekeeper
time
lost
mite
continue
cause
country
loss
issue
public
professor
varroa
pests
million
bumblebee
neonicotinoids
provide
serious
systemic
help
doses
crops
honeybee
course
england
insects
including
agricultural
association
importance
plants
hives
winter
declines
european
factors
report
department
published
colony
recent
example
scientific
beekeeping

Additional Resources

Here are some other books, papers, video and other resources, for a deeper dive into the topics covered in this book.

1. Mayer-Schonberger, Viktor; Cukier, Kenneth (2013). **Big Data: A Revolution That Will Transform How We Live, Work, and Think** . Houghton Mifflin Harcourt.
2. McKinsey Global Institute Report (2011). **Big data: The next frontier for innovation, competition, and productivity.** Mckinsey.com
3. Silver, N. (2012). **The Signal and the Noise: Why So Many Predictions Fail but Some Don't.** Penguin Press.
4. Matei Zaharia and et. Al. (2010). "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing," University of California, Berkeley. OReilley.
5. Sandy Ryza, Uri Laserson et.al (2014). "Advanced-Analytics-with-Spark". OReilley.

Websites:

6. Apache Hadoop resources: https://hadoop.apache.org/docs/r2.7.2/
7. Apache HDFS: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
8. Hadoop API site: http://hadoop.apache.org/docs/current/api/
9. Apache Spark: http://spark.apache.org/docs/latest/
10. **https://www.biostat.wisc.edu/~kbroman/Rintro/Rwinpack.html**
11. **http://robjhyndman.com/hyndsight/building-r-packages-for-windows/**
12. **https://stevemosher.wordpress.com/ten-steps-to-building-an-r-package-under-windows/**
13. **http://www.inside-r.org/packages/cran/wordcloud/docs/wordcloud**
14. **https://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html**
15. **https://intellipaat.com/tutorial/spark-tutorial/**
16. **https://blogs.apache.org/foundation/entry/the_apache_software_fou**
17. https://en.wikipedia.org/wiki/NoSQL

18. http://www.planetcassandra.org/what-is-apache-cassandra/
19. http://www.datastax.com/nosql
20. https://www.sitepen.com/blog/2010/05/11/nosql-architecture/
21. http://nosql-database.org/
22. http://webpages.uncc.edu/xwu/5160/nosqldbs.pdf

Video Resources

23. Doug Cutting on ‘Hadoop at 10’: https://www.youtube.com/watch?v=yDZRDDu3CJo
24. Status of Apache community: https://www.youtube.com/watch?v=sOZnf8Nn3Fo.
25. Spark 2.0 updates showing a nice demo across R, Scala and SQL) using tweets and clustering. https://www.youtube.com/watch?v=9xSz0ppBtFg
26. https://www.youtube.com/watch?v=VwiGHUKAHWM
27. https://www.youtube.com/watch?v=L5QWO8QBG5c
28. https://www.youtube.com/watch?v=KvQto_b3sqw
29. https://www.youtube.com/watch?v=YW28qItH_tA

About the Author

Dr. Anil Maheshwari is a Professor of Computer Science and Information Systems, and the Director of Center for Data Analytics, at Maharishi University of Management. He teaches courses in data analytics, and helps with extracting deep insights from their data. He worked in a variety of leadership roles at IBM in Austin TX, and has also worked at many other companies including startups.

He has taught at the University of Cincinnati, City University of New York, University of Illinois, and others. He earned an Electrical Engineering degree from Indian Institute of Technology in Delhi, an MBA from Indian Institute of Management in Ahmedabad, and a Ph.D. from Case Western Reserve University. He is a practitioner of Transcendental Meditation technique.

He is the author of the #1 bestseller Data Analytics Made Accessible.

He blogs interesting stuff on IT and Enlightenment at anilmah.com

Instructors can reach him for course materials at akm2030@gmail.com. Speaking engagements are welcome.