Big Data Analytics

Anuradha

BHATIA

Table of Contents

Disclaimer

The content of the book is the copyright property of the author, to be used by the students for the reference for the subject “Big Data Analytics”, CPE8035 and BEITL802, Eighth Semester, for the Final Year Computer Engineering and Information Technology Mumbai University.

The complete set of the e-book is available on the author’s website www.anuradhabhatia.com, and students are allowed to download it free of charge from the same.

The author does not gain any monetary benefit from the same, it is only developed and designed to help the teaching and the student fraternity to enhance their knowledge with respect to the curriculum prescribed by Mumbai University.

The reference is taken from Stanford Info-Lab Manual for developing the content as prescribed in the curriculum.

1. Introduction to Big Data

CONTENTS

1.1 Introduction to Big Data

1. Big Data is becoming one of the most talked about technology trends nowadays.
2. The real challenge with the big organization is to get maximum out of the data already available and predict what kind of data to collect in the future.
3. How to take the existing data and make it meaningful that it provides us accurate insight in the past data is one of the key discussion points in many of the executive meetings in organizations.
4. With the explosion of the data the challenge has gone to the next level and now a Big Data is becoming the reality in many organizations.
5. The goal of every organization and expert is same to get maximum out of the data, the route and the starting point are different for each organization and expert.
6. As organizations are evaluating and architecting big data solutions they are also learning the ways and opportunities which are related to Big Data.
7. There is not a single solution to big data as well there is not a single vendor which can claim to know all about Big Data.
8. Big Data is too big a concept and there are many players – different architectures, different vendors and different technology.
9. The three Vs of Big data are Velocity, Volume and Variety.

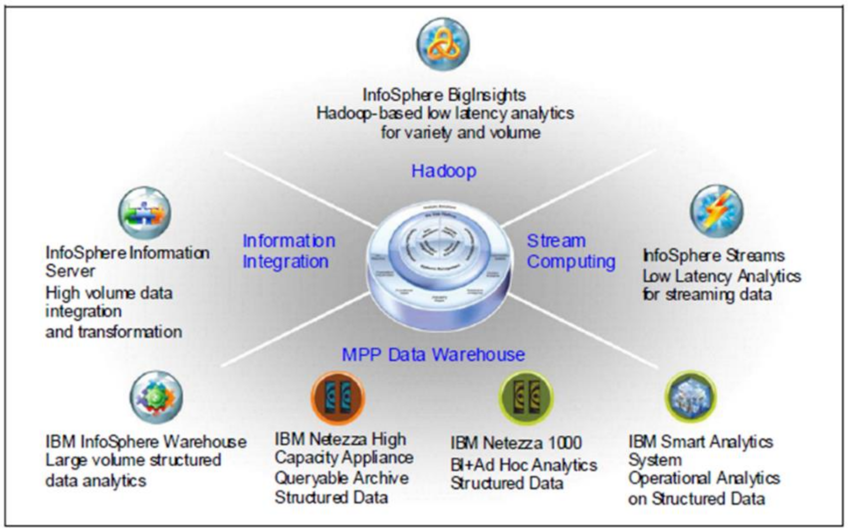

InfoSphere BigInsights
Hadoop-based low latency analytics for variety and volume
Hadoop
InfoSphere Information Server
High volume data integration and transformation
Information Integration
Stream Computing
InfoSphere Streams
Low Latency Analytics for streaming data
MPP Data Warehouse
IBM InfoSphere Warehouse
Large volume structured data analytics
IBM Netezza High Capacity Appliance
Queryable Archive Structured Data
IBM Netezza 1000
BI+Ad Hoc Analytics Structured Data
IBM Smart Analytics System
Operational Analytics on Structured Data

Figure 1.1: Big Data Sphere

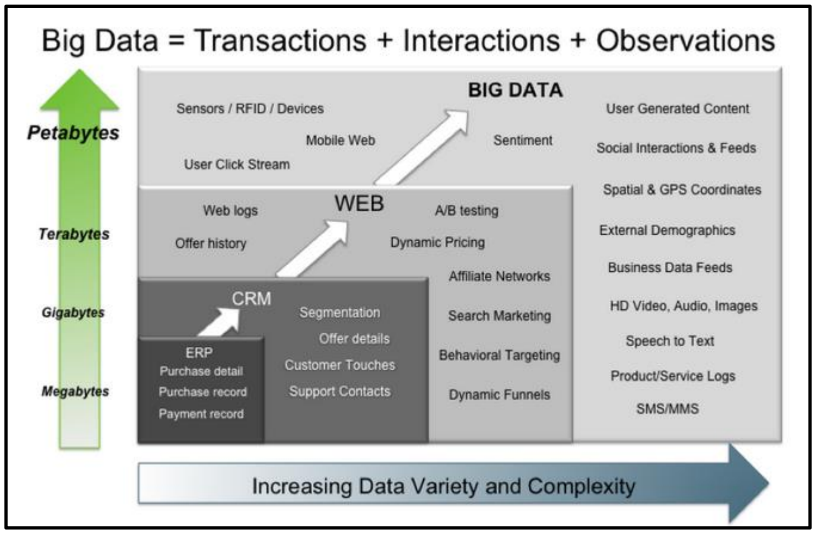


Figure 1.2: Big Data – Transactions, Interactions, Observations

1.2 Big data Characteristics

1. The three Vs of Big data are Velocity, Volume and Variety

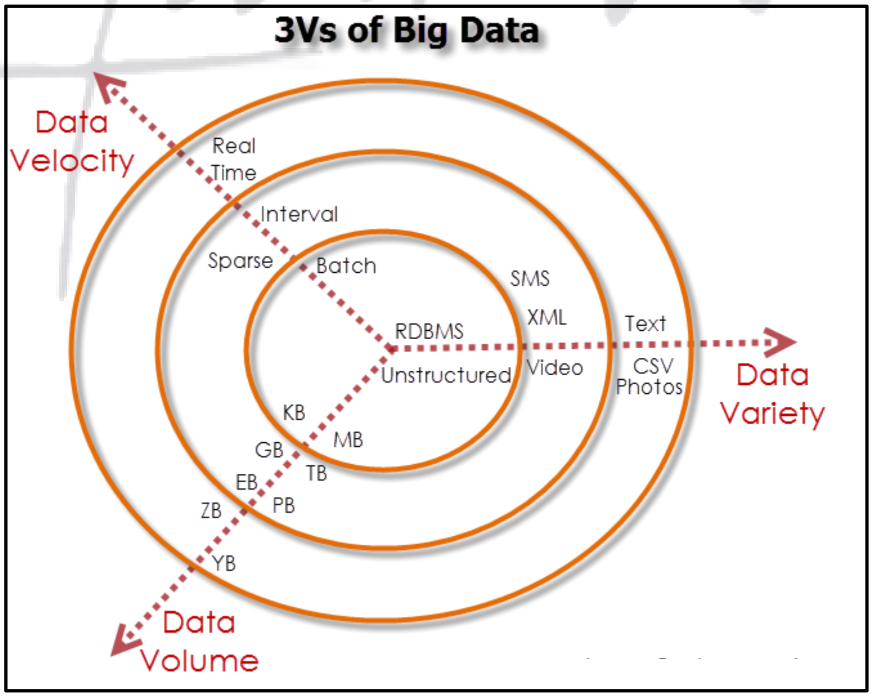


Figure 1.3: Characteristics of Big Data

i. Volume

- The exponential growth in the data storage as the data is now more than text data.
- The data can be found in the format of videos, music's and large images on our social media channels.
- It is very common to have Terabytes and Petabytes of the storage system for enterprises.
- As the database grows the applications and architecture built to support the data needs to be reevaluated quite often.
- Sometimes the same data is re-evaluated with multiple angles and even though the original data is the same the new found intelligence creates explosion of the data.
- The big volume indeed represents Big Data.

ii. Velocity

- The data growth and social media explosion have changed how we look at the data.
- There was a time when we used to believe that data of yesterday is recent.
- The matter of the fact newspapers is still following that logic.
- However, news channels and radios have changed how fast we receive the news.
- Today, people reply on social media to update them with the latest happening. On social media sometimes a few seconds old messages (a tweet, status updates etc.) is not something interests users.
- They often discard old messages and pay attention to recent updates. The data movement is now almost real time and the update window has reduced to fractions of the seconds.
- This high velocity data represent Big Data.

iii. Variety

- Data can be stored in multiple format. For example database, excel, csv, access or for the matter of the fact, it can be stored in a simple text file.
- Sometimes the data is not even in the traditional format as we assume, it may be in the form of video, SMS, pdf or something we might have not thought about it. It is the need of the organization to arrange it and make it meaningful.
- It will be easy to do so if we have data in the same format, however it is not the case most of the time. The real world have data in many different formats and that is the challenge we need to overcome with the Big Data. This variety of the data represent Big Data.

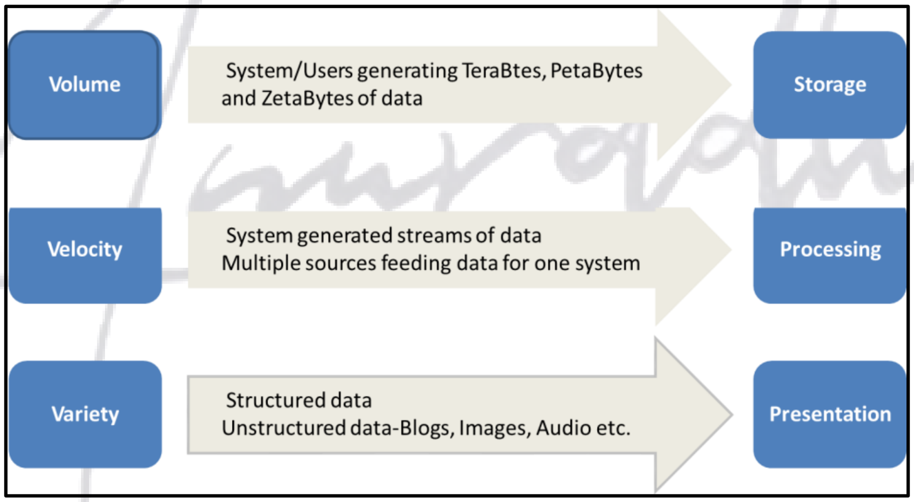

Volume
System/Users generating TeraBtes, PetaBytes and ZetaBytes of data
Storage
Velocity
System generated streams of data
Multiple sources feeding data for one system
Processing
Variety
Structured data
Unstructured data-Blogs, Images, Audio etc.
Presentation

Figure 1.4: Volume, Velocity, Variety

1.3 Types of Big data

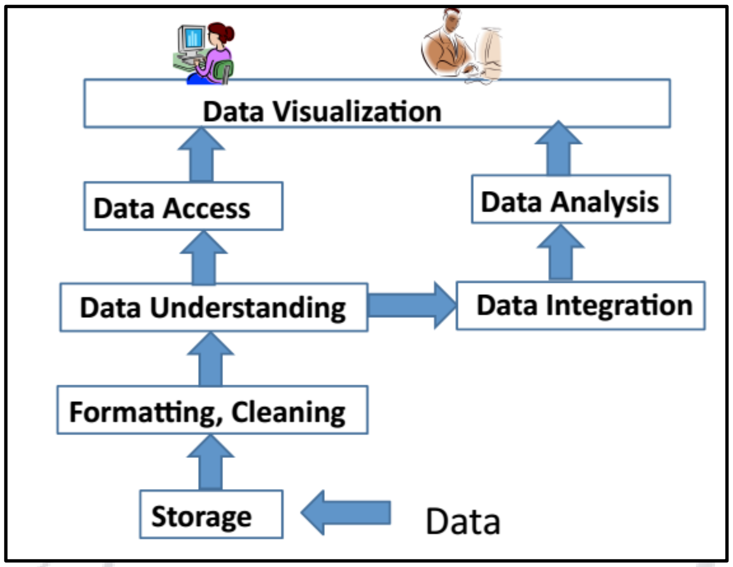


Figure 1.5: Big Data Layout

1. Apache Hadoop

- Apache Hadoop is one of the main supportive element in Big Data technologies. It simplifies the processing of large amount of structured or unstructured data in a cheap manner.
- Hadoop is an open source project from apache that is continuously improving over the years.
- "Hadoop is basically a set of software libraries and frameworks to manage and process big amount of data from a single server to thousands of machines.
- It provides an efficient and powerful error detection mechanism based on application layer rather than relying upon hardware."
- In December 2012 apache releases Hadoop 1.0.0, more information and installation guide can be found at Apache Hadoop Documentation. Hadoop is not a single project but includes a number of other technologies in it.

2. MapReduce

- MapReduce was introduced by google to create large amount of web search indexes.

- It is basically a framework to write applications that processes a large amount of structured or unstructured data over the web.
- MapReduce takes the query and breaks it into parts to run it on multiple nodes.
- By distributed query processing it makes it easy to maintain large amount of data by dividing the data into several different machines.
- Hadoop MapReduce is a software framework for easily writing applications to manage large amount of data sets with a highly fault tolerant manner.
- More tutorials and getting started guide can be found at Apache Documentation.

3. HDFS(Hadoop distributed file system)

- HDFS is a java based file system that is used to store structured or unstructured data over large clusters of distributed servers.
- The data stored in HDFS has no restriction or rule to be applied, the data can be either fully unstructured of purely structured.
- In HDFS the work to make data senseful is done by developer's code only.
- Hadoop distributed file system provides a highly fault tolerant atmosphere with a deployment on low cost hardware machines.
- HDFS is now a part of Apache Hadoop project, more information and installation guide can be found at Apache HDFS documentation.

4. Hive

- Hive was originally developed by Facebook, now it is made open source for some time.
- Hive works something like a bridge in between sql and Hadoop, it is basically used to make Sql queries on Hadoop clusters.
- Apache Hive is basically a data warehouse that provides ad-hoc queries, data summarization and analysis of huge data sets stored in Hadoop compatible file systems.

- Hive provides a SQL like called HiveQL query based implementation of huge amount of data stored in Hadoop clusters. In January 2013 apache releases Hive 0.10.0, more information and installation guide can be found at Apache Hive Documentation.

5. Pig

- Pig was introduced by yahoo and later on it was made fully open source.
- It also provides a bridge to query data over Hadoop clusters but unlike hive, it implements a script implementation to make Hadoop data access able by developers and business persons.
- Apache pig provides a high level programming platform for developers to process and analyses Big Data using user defined functions and programming efforts. In January 2013 Apache released Pig 0.10.1 which is defined for use with Hadoop 0.10.1 or later releases. More information and installation guide can be found at Apache Pig Getting Started Documentation.

1.4 Traditional Vs Big Data Business Approach

1. Schema less and Column oriented Databases (No Sql)

i. We are using table and row based relational databases over the years, these databases are just fine with online transactions and quick updates.

ii. When unstructured and large amount of data comes into the picture we needs some databases without having a hard code schema attachment.

iii. There are a number of databases to fit into this category, these databases can store unstructured, semi structured or even fully structured data.

iv. Apart from other benefits the finest thing with schema less databases is that it makes data migration very easy.

v. MongoDB is a very popular and widely used NoSQL database these days.

vi. NoSQL and schema less databases are used when the primary concern is to store a huge amount of data and not to maintain relationship between elements.

vii. "NoSQL (not only Sql) is a type of databases that does not primarily rely upon schema based structure and does not use Sql for data processing."

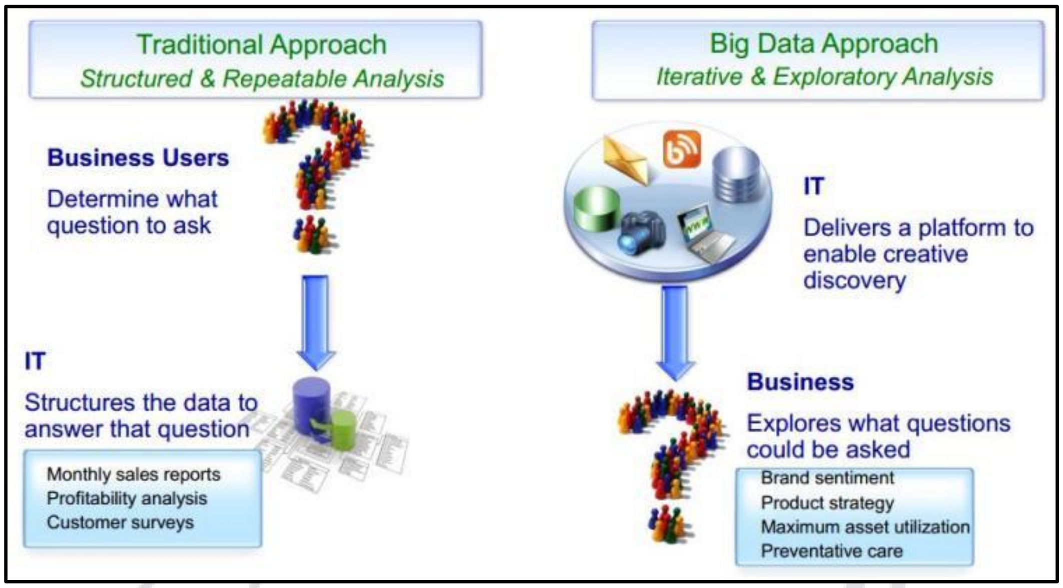

Traditional Approach
Structured & Repeatable Analysis
Business Users
Determine what question to ask
IT
Structures the data to answer that question
Monthly sales reports
Profitability analysis
Customer surveys
Big Data Approach
Iterative & Exploratory Analysis
IT
Delivers a platform to enable creative discovery
Business
Explores what questions could be asked
Brand sentiment
Product strategy
Maximum asset utilization
Preventative care

Figure 1.6: Big Data Architecture

viii. The traditional approach work on the structured data that has a basic layout and the structure provided.

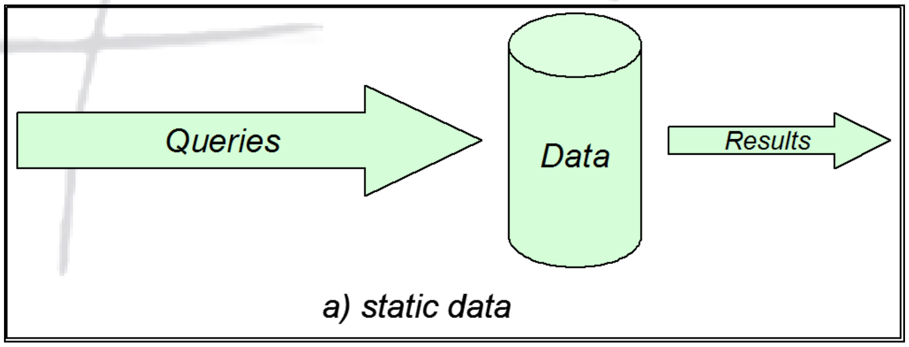


Figure 1.7: Static Data

ix. The structured approach designs the database as per the requirements in tuples and columns.

x. Working on the live coming data, which can be an input from the ever changing scenario cannot be dealt in the traditional approach.

xi. The Big data approach is iterative.

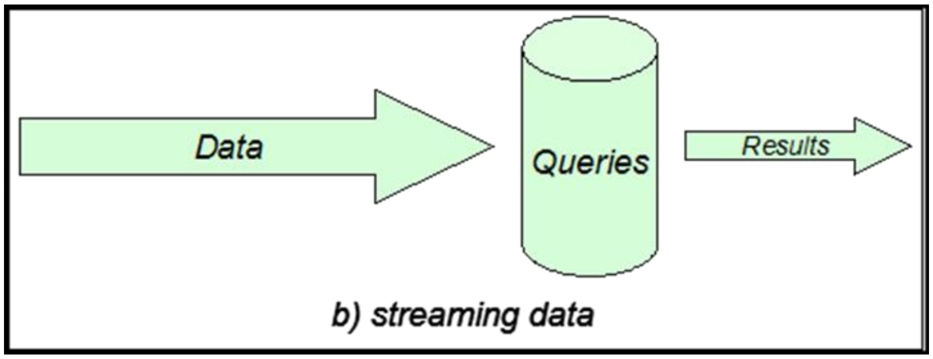


Figure 1.8: Streaming Data

xii. The Big data analytics work on the unstructured data, where no specific pattern of the data is defined.

xiii. The data is not organized in rows and columns.

xiv. The live flow of data is captured and the analysis is done on it.

xv. Efficiency increases when the data to be analyzed is large.

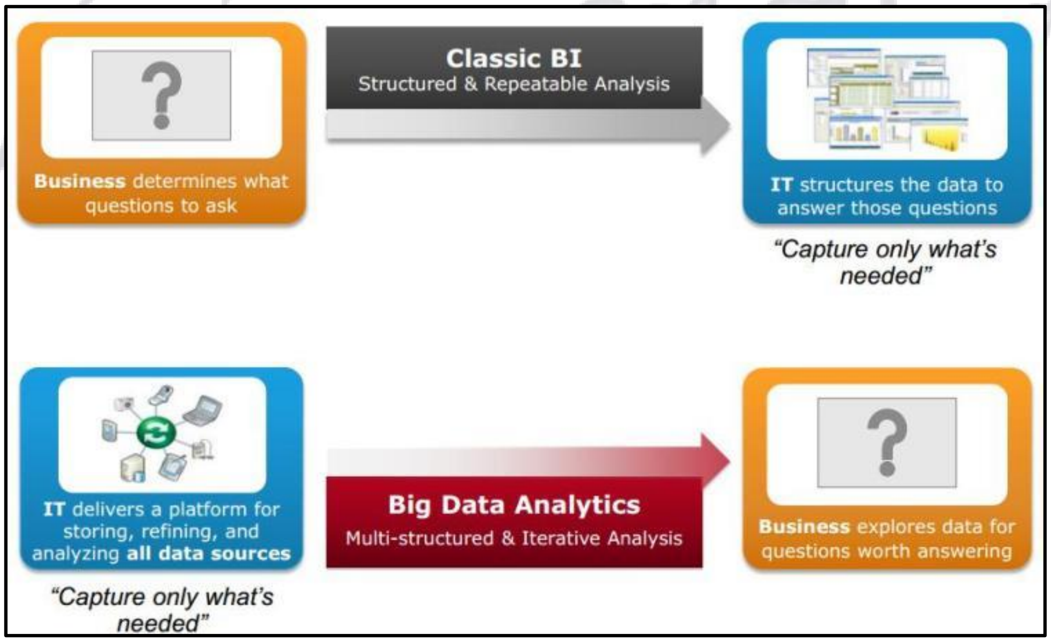

Business determines what questions to ask
Classic BI
Structured & Repeatable Analysis
IT structures the data to answer those questions
"Capture only what's needed"
IT delivers a platform for storing, refining, and analyzing all data sources
"Capture only what's needed"
Big Data Analytics
Multi-structured & Iterative Analysis
Business explores data for questions worth answering

Figure 1.9: Big Data Architecture

1.5 Case Study of Big Data Solutions

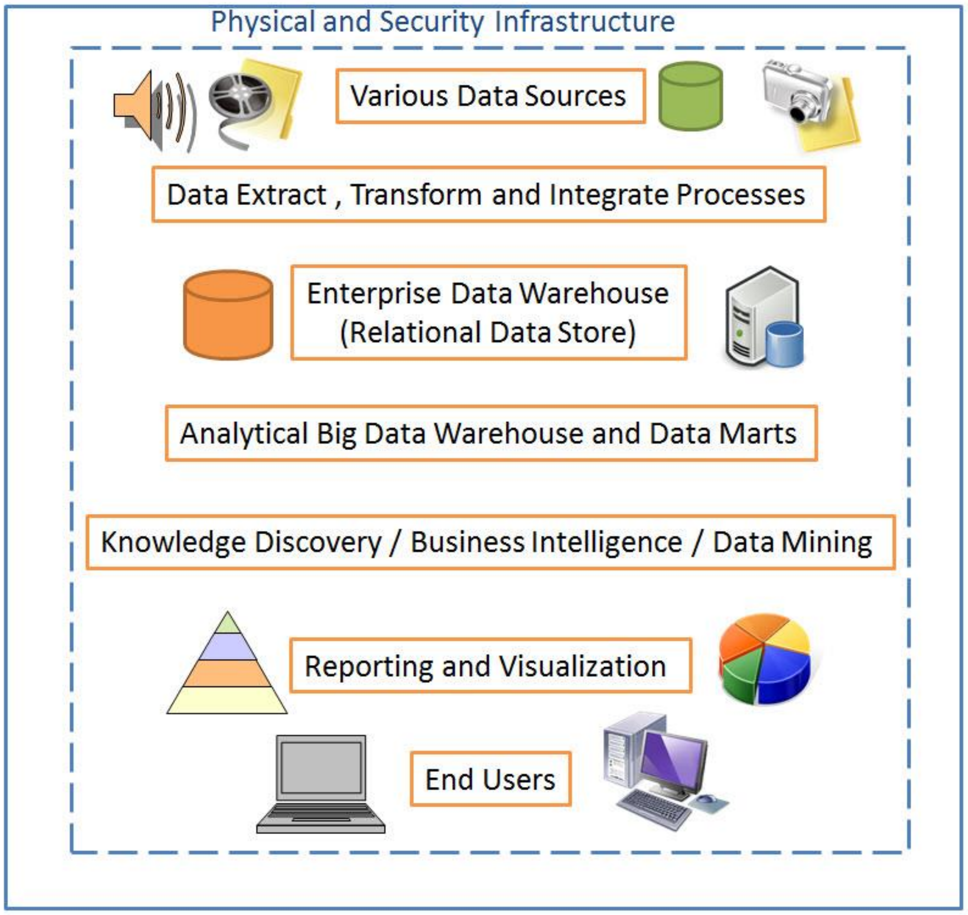

Physical and Security Infrastructure
Various Data Sources
Data Extract , Transform and Integrate Processes
Enterprise Data Warehouse (Relational Data Store)
Analytical Big Data Warehouse and Data Marts
Knowledge Discovery / Business Intelligence / Data Mining
Reporting and Visualization
End Users

Figure 1.10: Big Data Infrastructure

1. Above image gives good overview of how in Big Data Infrastructure various components are associated with each other.
2. In Big Data various different data sources are part of the architecture hence extract, transform and integration are one of the most essential layers of the architecture.
3. Most of the data is stored in relational as well as non-relational data marts and data warehousing solutions.
4. As per the business need various data are processed as well converted to proper reports and visualizations for end users.

5. Just like software the hardware is almost the most important part of the Big Data Infrastructure.
6. In the big data architecture hardware infrastructure is extremely important and failure over instances as well as redundant physical infrastructure is usually implemented.

Life cycle of Data

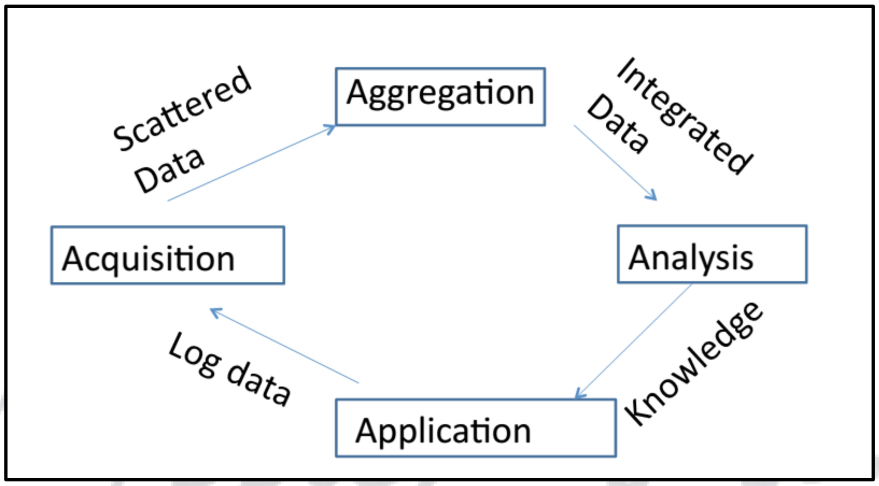


Figure 1.11: Life Cycle of Data

i. The life cycle of the data is as shown in Figure 1.11.
ii. The analysis of data is done from the knowledge experts and the expertise is applied for the development of an application.
iii. The streaming of data after the analysis and the application, the data log is created for the acquisition of data.
iv. The data id mapped and clustered together on the data log.
v. The clustered data from the data acquisition is then aggregated by applying various aggregation algorithms.
vi. The integrated data again goes for an analysis.
vii. The complete steps are repeated till the desired, and expected output is produced.

2. Introduction to Hadoop

CONTENTS

2.1 What is Hadoop?

1. Hadoop is an open source framework that supports the processing of large data sets in a distributed computing environment.
2. Hadoop consists of MapReduce, the Hadoop distributed file system (HDFS) and a number of related projects such as Apache Hive, HBase and Zookeeper. MapReduce and Hadoop distributed file system (HDFS) are the main component of Hadoop.
3. Apache Hadoop is an open-source, free and Java based software framework offers a powerful distributed platform to store and manage Big Data.
4. It is licensed under an Apache V2 license.
5. It runs applications on large clusters of commodity hardware and it processes thousands of terabytes of data on thousands of the nodes. Hadoop is inspired from Google's MapReduce and Google File System (GFS) papers.
6. The major advantage of Hadoop framework is that it provides reliability and high availability.

2.2 Why Use Hadoop?

There are many advantages of using Hadoop:

1. **Robust and Scalable** – We can add new nodes as needed as well modify them.
2. **Affordable and Cost Effective** – We do not need any special hardware for running Hadoop. We can just use commodity server.
3. **Adaptive and Flexible** – Hadoop is built keeping in mind that it will handle structured and unstructured data.
4. **Highly Available and Fault Tolerant** – When a node fails, the Hadoop framework automatically fails over to another node.

2.3 Core Hadoop Components

There are two major components of the Hadoop framework and both of them does two of the important task for it.

1. **Hadoop MapReduce** is the method to split a larger data problem into smaller chunk and distribute it to many different commodity servers. Each server have their own set of resources and they have processed them locally. Once the commodity server has processed the data they send it back collectively to main server. This is effectively a process where we process large data effectively and efficiently
2. **Hadoop Distributed File System (HDFS)** is a virtual file system. There is a big difference between any other file system and Hadoop. When we move a file on HDFS, it is automatically split into many small pieces. These small chunks of the file are replicated and stored on other servers (usually 3) for the fault tolerance or high availability.
3. **Namenode**: Namenode is the heart of the Hadoop system. The NameNode manages the file system namespace. It stores the metadata information of the data blocks. This metadata is stored permanently on to local disk in the form of namespace image and edit log file. The NameNode also knows the location of the data blocks on the data node. However the NameNode does not store this information persistently. The NameNode creates the block to DataNode mapping when it is restarted. If the NameNode crashes, then the entire Hadoop system goes down. Read more about Namenode
4. **Secondary Namenode**: The responsibility of secondary name node is to periodically copy and merge the namespace image and edit log. In case if the name node crashes, then the namespace image stored in secondary NameNode can be used to restart the NameNode.
5. **DataNode**: It stores the blocks of data and retrieves them. The DataNodes also reports the blocks information to the NameNode periodically.

6. **Job Tracker**: Job Tracker responsibility is to schedule the client's jobs. Job tracker creates map and reduce tasks and schedules them to run on the DataNodes (task trackers). Job Tracker also checks for any failed tasks and reschedules the failed tasks on another DataNode. Job tracker can be run on the NameNode or a separate node.
7. **Task Tracker**: Task tracker runs on the DataNodes. Task trackers responsibility is to run the map or reduce tasks assigned by the NameNode and to report the status of the tasks to the NameNode.

   Besides above two core components Hadoop project also contains following modules as well.

1. **Hadoop Common**: Common utilities for the other Hadoop modules
2. **Hadoop Yarn**: A framework for job scheduling and cluster resource management

2.4 Hadoop Ecosystem

Distributed Filesystem		
Apache HDFS	➢ The Hadoop Distributed File System (HDFS) offers a way to store large files across multiple machines. Hadoop and HDFS was derived from Google File System (GFS) paper. ➢ Prior to Hadoop 2.0.0, the NameNode was a single point of failure (SPOF) in an HDFS cluster. ➢ With Zookeeper the HDFS High Availability feature addresses this problem by providing the option of running two redundant NameNodes in the same cluster in an Active/Passive configuration with a hot standby.	1. hadoop.apache.org 2. Google FileSystem - GFS Paper 3. Cloudera Why HDFS 4. Hortonworks Why HDFS
Red Hat GlusterFS	➢ GlusterFS is a scale-out network-attached storage file system. ➢ GlusterFS was developed originally by Gluster, Inc., then by Red Hat, Inc., after their purchase of Gluster in 2011. ➢ In June 2012, Red Hat Storage Server was announced as a commercially-supported integration of GlusterFS with Red Hat Enterprise Linux. ➢ Gluster File System, known now as Red Hat Storage Server.	1. www.gluster.org 2. Red Hat Hadoop Plugin
Quantcast File System QFS	➢ QFS is an open-source distributed file system software package for large-scale MapReduce or other batch-processing workloads. ➢ It was designed as an alternative to Apache Hadoop's HDFS, intended to deliver better performance and cost-efficiency for large-scale processing clusters. ➢ It is written in C++ and has fixed-footprint memory management. QFS uses Reed-Solomon error correction as method for assuring reliable access to data.	1. QFS site 2. GitHub QFS 3. HADOOP-8885
Ceph Filesystem	➢ Ceph is a free software storage platform designed to present object, block, and file storage from a single distributed computer cluster. ➢ Ceph's main goals are to be completely distributed without a single point of failure, scalable to the exabyte level, and freely-available. ➢ The data is replicated, making it fault tolerant.	1. Ceph Filesystem site 2. Ceph and Hadoop 3. HADOOP-6253
Lustre file system	➢ The Lustre filesystem is a high-performance distributed filesystem intended for larger network and high-availability environments.	1. wiki.lustre.org/ 2. Hadoop with Lustre 3. Intel HPC Hadoop

	➢ Traditionally, Lustre is configured to manage remote data storage disk devices within a Storage Area Network (SAN), which is two or more remotely attached disk devices communicating via a Small Computer System Interface (SCSI) protocol. ➢ This includes Fibre Channel, Fibre Channel over Ethernet (FCoE), Serial Attached SCSI (SAS) and even iSCSI. ➢ With Hadoop HDFS the software needs a dedicated cluster of computers on which to run. ➢ But folks who run high performance computing clusters for other purposes often don't run HDFS, which leaves them with a bunch of computing power, tasks that could almost certainly benefit from a bit of map reduce and no way to put that power to work running Hadoop.	
Tachyon	➢ Tachyon is an open source memory-centric distributed storage system enabling reliable data sharing at memory-speed across cluster jobs, possibly written in different computation frameworks, such as Apache Spark and Apache MapReduce. ➢ In the big data ecosystem, Tachyon lies between computation frameworks or jobs, such as Apache Spark, Apache MapReduce, or Apache Flink, and various kinds of storage systems, such as Amazon S3, OpenStack Swift, GlusterFS, HDFS, or Ceph. ➢ Tachyon brings significant performance improvement to the stack; for example, Baidu uses Tachyon to improve their data analytics performance by 30 times. ➢ Beyond performance, Tachyon bridges new workloads with data stored in traditional storage systems. Users can run Tachyon using its standalone cluster mode, for example on Amazon EC2, or launch Tachyon with Apache Mesos or Apache Yarn. ➢ Tachyon is Hadoop compatible. This means that existing Spark and MapReduce programs can run on top of Tachyon without any code changes. ➢ The project is open source (Apache License 2.0) and is deployed at multiple companies. It is one of the fastest growing open source projects.	1. Tachyon site
GridGain	➢ GridGain is open source project licensed under Apache 2.0. One of the main pieces of this platform is the In-Memory Apache Hadoop	1. GridGain site

	Accelerator which aims to accelerate HDFS and Map/Reduce by bringing both, data and computations into memory. ➢ This work is done with the GGFS - Hadoop compliant in-memory file system. For I/O intensive jobs GridGain GGFS offers performance close to 100x faster than standard HDFS. ➢ Paraphrasing Dmitriy Setrakyan from GridGain Systems talking about GGFS regarding Tachyon ➢ GGFS allows read-through and write-through to/from underlying HDFS or any other Hadoop compliant file system with zero code change. Essentially GGFS entirely removes ETL step from integration. ➢ GGFS has ability to pick and choose what folders stay in memory, what folders stay on disc, and what folders get synchronized with underlying (HD) FS either synchronously or asynchronously. ➢ GridGain is working on adding native MapReduce component which will provide native complete Hadoop integration without changes in API, like Spark currently forces you to do. Essentially GridGain MR+GGFS will allow to bring Hadoop completely or partially in-memory in Plug-n-Play fashion without any API changes.	
XtreemFS	➢ XtreemFS is a general purpose storage system and covers most storage needs in a single deployment. ➢ It is open-source, requires no special hardware or kernel modules, and can be mounted on Linux, Windows and OS X. ➢ XtreemFS runs distributed and offers resilience through replication. XtreemFS Volumes can be accessed through a FUSE component that offers normal file interaction with POSIX like semantics. ➢ An implementation of Hadoop File System interface is included which makes XtreemFS available for use with Hadoop, Flink and Spark out of the box. XtreemFS is licensed under the New BSD license. ➢ The XtreemFS project is developed by Zuse Institute Berlin.	1. XtreemFS site 2. Flink on XtreemFS. Spark XtreemFS
Distributed Programming		
Apache MapReduce	➢ MapReduce is a programming model for processing large data sets with a parallel, distributed algorithm on a cluster. ➢ Apache MapReduce was derived from Google MapReduce: Simplified Data Processing on Large Clusters paper. The current Apache	1. Apache MapReduce 2. Google MapReduce paper 3. Writing YARN applications

	MapReduce version is built over Apache YARN Framework. YARN stands for “Yet-Another-Resource-Negotiator”. ➢ It is a new framework that facilitates writing arbitrary distributed processing frameworks and applications. YARN’s execution model is more generic than the earlier MapReduce implementation. ➢ YARN can run applications that do not follow the MapReduce model, unlike the original Apache Hadoop MapReduce (also called MR1). Hadoop YARN is an attempt to take Apache Hadoop beyond MapReduce for data-processing.	
Apache Pig	➢ Pig provides an engine for executing data flows in parallel on Hadoop. ➢ It includes a language, Pig Latin, for expressing these data flows. ➢ Pig Latin includes operators for many of the traditional data operations (join, sort, filter, etc.), as well as the ability for users to develop their own functions for reading, processing, and writing data. ➢ Pig runs on Hadoop. It makes use of both the Hadoop Distributed File System, HDFS, and Hadoop’s processing system, MapReduce. ➢ Pig uses MapReduce to execute all of its data processing. ➢ It compiles the Pig Latin scripts that users write into a series of one or more MapReduce jobs that it then executes. ➢ Pig Latin looks different from many of the programming languages you have seen. ➢ There are no if statements or for loops in Pig Latin. ➢ This is because traditional procedural and object-oriented programming languages describe control flow, and data flow is a side effect of the program. Pig Latin instead focuses on data flow.	1. pig.apache.org/ 2. Pig examples by Alan Gates
JAQL	➢ JAQL is a functional, declarative programming language designed especially for working with large volumes of structured, semi-structured and unstructured data. ➢ As its name implies, a primary use of JAQL is to handle data stored as JSON documents, but JAQL can work on various types of data. ➢ For example, it can support XML, comma-separated values (CSV) data and flat files. ➢ A "SQL within JAQL" capability lets programmers work with structured SQL data while employing a JSON data model that's less restrictive than its Structured Query Language counterparts.	1. JAQL in Google Code 2. What is Jaql? by IBM

	➢ Specifically, Jaql allows you to select, join, group, and filter data that is stored in HDFS, much like a blend of Pig and Hive. Jaql's query language was inspired by many programming and query languages, including Lisp, SQL, XQuery, and Pig. ➢ JAQL was created by workers at IBM Research Labs in 2008 and released to open source. While it continues to be hosted as a project on Google Code, where a downloadable version is available under an Apache 2.0 license, the major development activity around JAQL has remained centered at IBM. ➢ The company offers the query language as part of the tools suite associated with Infosphere BigInsights, its Hadoop platform. Working together with a workflow orchestrator, JAQL is used in BigInsights to exchange data between storage, processing and analytics jobs. It also provides links to external data and services, including relational databases and machine learning data.	
Apache Spark	➢ Data analytics cluster computing framework originally developed in the AMPLab at UC Berkeley. Spark fits into the Hadoop open-source community, building on top of the Hadoop Distributed File System (HDFS). ➢ Spark provides an easier to use alternative to Hadoop MapReduce and offers performance up to 10 times faster than previous generation systems like Hadoop MapReduce for certain applications. ➢ Spark is a framework for writing fast, distributed programs. ➢ Spark solves similar problems as Hadoop MapReduce does but with a fast in-memory approach and a clean functional style API. ➢ With its ability to integrate with Hadoop and inbuilt tools for interactive query analysis (Shark), large-scale graph processing and analysis (Bagel), and real-time analysis (Spark Streaming), it can be interactively used to quickly process and query big data sets. ➢ To make programming faster, Spark provides clean, concise APIs in Scala, Java and Python. ➢ It sis interactively used from the Scala and Python shells to rapidly query big datasets. Spark is also the engine behind Shark, a fully Apache Hive-compatible data warehousing system that can run 100x faster than Hive.	1. Apache Spark 2. Mirror of Spark on GitHub 3. RDDs - Paper 4. Spark: Cluster Computing Paper Spark Research

Apache Storm	➢ Storm is a complex event processor (CEP) and distributed computation framework written predominantly in the Clojure programming language. Is a distributed real-time computation system for processing fast, large streams of data. Storm is an architecture based on master-workers paradigma. So a Storm cluster mainly consists of a master and worker nodes, with coordination done by Zookeeper. ➢ Storm makes use of zeromq (0mq, zeromq), an advanced, embeddable networking library. It provides a message queue, but unlike message-oriented middleware (MOM), a 0MQ system can run without a dedicated message broker. The library is designed to have a familiar socket-style API.	1. Storm Project
Apache Flink	➢ Apache Flink (formerly called Stratosphere) features powerful programming abstractions in Java and Scala, a high-performance runtime, and automatic program optimization. It has native support for iterations, incremental iterations, and programs consisting of large DAGs of operations. ➢ Flink is a data processing system and an alternative to Hadoop's MapReduce component. It comes with its own runtime, rather than building on top of MapReduce. ➢ As such, it can work completely independently of the Hadoop ecosystem. However, Flink can also access Hadoop's distributed file system (HDFS) to read and write data, and Hadoop's next-generation resource manager (YARN) to provision cluster resources. ➢ Since most Flink users are using Hadoop HDFS to store their data, it ships already the required libraries to access HDFS.	1. Apache Flink incubator page 2. Stratosphere site
Apache Apex	➢ Apache Apex is an enterprise grade Apache YARN based big data-in-motion platform that unifies stream processing as well as batch processing. ➢ It processes big data in-motion in a highly scalable, highly performant, fault tolerant, stateful, secure, distributed, and an easily operable way. It provides a simple API that enables users to write or re-use generic Java code, thereby lowering the expertise needed to write big data applications.	1. Apache Apex from DataTorrent 2. Apache Apex main page 3. Apache Apex Proposal

	➢ The Apache Apex platform is supplemented by Apache Apex-Malhar, which is a library of operators that implement common business logic functions needed by customers who want to quickly develop applications. ➢ These operators provide access to HDFS, S3, NFS, FTP, and other file systems; Kafka, ActiveMQ, RabbitMQ, JMS, and other message systems; MySql, Cassandra, MongoDB, Redis, HBase, CouchDB and other databases along with JDBC connectors. ➢ The library also includes a host of other common business logic patterns that help users to significantly reduce the time it takes to go into production. Ease of integration with all other big data technologies is one of the primary missions of Apache Apex-Malhar.	
Netflix PigPen	➢ PigPen is map-reduce for Clojure which compiles to Apache Pig. Clojure is dialect of the Lisp programming language created by Rich Hickey, so is a functional general-purpose language, and runs on the Java Virtual Machine, Common Language Runtime, and JavaScript engines. ➢ In PigPen there are no special user defined functions (UDFs). Define Clojure functions, anonymously or named, and use them like you would in any Clojure program. ➢ This tool is open sourced by Netflix, Inc. the American provider of on-demand Internet streaming media.	1. PigPen on GitHub
AMPLab SIMR	➢ Apache Spark was developed thinking in Apache YARN. ➢ It has been relatively hard to run Apache Spark on Hadoop MapReduce v1 clusters, i.e. clusters that do not have YARN installed. ➢ Typically, users would have to get permission to install Spark/Scala on some subset of the machines, a process that could be time consuming. SIMR allows anyone with access to a Hadoop MapReduce v1 cluster to run Spark out of the box. ➢ A user can run Spark directly on top of Hadoop MapReduce v1 without any administrative rights, and without having Spark or Scala installed on any of the nodes.	1. SIMR on GitHub
Facebook Corona	➢ “The next version of Map-Reduce" from Facebook, based in own fork of Hadoop. The current Hadoop implementation of the MapReduce technique uses a single job tracker, which causes scaling issues for very large data sets.	1. Corona on Github

	➢ The Apache Hadoop developers have been creating their own next-generation MapReduce, called YARN, which Facebook engineers looked at but discounted because of the highly-customised nature of the company's deployment of Hadoop and HDFS. Corona, like YARN, spawns multiple job trackers (one for each job, in Corona's case).	
Apache Twill	➢ Twill is an abstraction over Apache Hadoop® YARN that reduces the complexity of developing distributed applications, allowing developers to focus more on their business logic. ➢ Twill uses a simple thread-based model that Java programmers will find familiar. ➢ YARN can be viewed as a compute fabric of a cluster, which means YARN applications like Twill will run on any Hadoop 2 cluster.YARN is an open source application that allows the Hadoop cluster to turn into a collection of virtual machines. ➢ Weave, developed by Continuuity and initially housed on Github, is a complementary open source application that uses a programming model similar to Java threads, making it easy to write distributed applications. In order to remove a conflict with a similarly named project on Apache, called "Weaver," Weave's name changed to Twill when it moved to Apache incubation. ➢ Twill functions as a scaled-out proxy. Twill is a middleware layer in between YARN and any application on YARN. When you develop a Twill app, Twill handles APIs in YARN that resemble a multi-threaded application familiar to Java. It is very easy to build multi-processed distributed applications in Twill.	1. Apache Twill Incubator
Damballa Parkour	➢ Library for develop MapReduce programs using the LISP like language Clojure. Parkour aims to provide deep Clojure integration for Hadoop. ➢ Programs using Parkour are normal Clojure programs, using standard Clojure functions instead of new framework abstractions. ➢ Programs using Parkour are also full Hadoop programs, with complete access to absolutely everything possible in raw Java Hadoop MapReduce.	1. Parkour GitHub Project
Apache Hama	➢ Apache Top-Level open source project, allowing you to do advanced analytics beyond MapReduce.	1. Hama site

	➢ Many data analysis techniques such as machine learning and graph algorithms require iterative computations, this is where Bulk Synchronous Parallel model can be more effective than "plain" MapReduce.	
Datasalt Pangool	➢ A new MapReduce paradigm. A new API for MR jobs, in higher level than Java.	1.Pangool 2.GitHub Pangool
NoSQL Databases		
Column Data Model		
Apache HBase	➢ Google BigTable Inspired. Non-relational distributed database. Ramdom, real-time r/w operations in column-oriented very large tables (BDDB: Big Data Data Base). ➢ It's the backing system for MR jobs outputs. It's the Hadoop database. It's for backing Hadoop MapReduce jobs with Apache HBase tables	1. Apache HBase Home 2. Mirror of HBase on Github
Apache Cassandra	➢ Distributed Non-SQL DBMS, it's a BDDB. MR can retrieve data from Cassandra. This BDDB can run without HDFS, or on-top of HDFS (DataStax fork of Cassandra). ➢ HBase and its required supporting systems are derived from what is known of the original Google BigTable and Google File System designs (as known from the Google File System paper Google published in 2003, and the BigTable paper published in 2006). ➢ Cassandra on the other hand is a recent open source fork of a standalone database system initially coded by Facebook, which while implementing the BigTable data model, uses a system inspired by Amazon's Dynamo for storing data (in fact much of the initial development work on Cassandra was performed by two Dynamo engineers recruited to Facebook from Amazon).	1. Apache HBase Home 2. Cassandra on GitHub 3. Training Resources 4. Cassandra - Paper
Document Data Model		
MongoDB	➢ Document-oriented database system. It is part of the NoSQL family of database systems. Instead of storing data in tables as is done in a "classical" relational database, MongoDB stores structured data as JSON-like documents	1. Mongodb site
RethinkDB	➢ RethinkDB is built to store JSON documents, and scale to multiple machines with very little effort. It has a pleasant query language that supports really useful queries like table joins and group by, and is easy to setup and learn.	1. RethinkDB site

ArangoDB	➢ An open-source database with a flexible data model for documents, graphs, and key-values. Build high performance applications using a convenient sql-like query language or JavaScript extensions.	1. ArangoDB site
Stream Data Model		
EventStore	➢ An open-source, functional database with support for Complex Event Processing. It provides a persistence engine for applications using event-sourcing, or for storing time-series data. Event Store is written in C#, C++ for the server which runs on Mono or the .NET CLR, on Linux or Windows. Applications using Event Store can be written in JavaScript. ➢ Event sourcing (ES) is a way of persisting your application's state by storing the history that determines the current state of your application.	1. EventStore site
Key-Value Data Model		
Redis DataBase	➢ Redis is an open-source, networked, in-memory, data structures store with optional durability. It is written in ANSI C. In its outer layer, the Redis data model is a dictionary which maps keys to values. ➢ One of the main differences between Redis and other structured storage systems is that Redis supports not only strings, but also abstract data types. Sponsored by Redis Labs. It's BSD licensed.	1. Redis site 2. Redis Labs site
Linkedin Voldemort	Distributed data store that is designed as a key-value store used by LinkedIn for high-scalability storage.	1. Voldemort site
RocksDB	➢ RocksDB is an embeddable persistent key-value store for fast storage. RocksDB can also be the foundation for a client-server database but our current focus is on embedded workloads.	1. RocksDB site
OpenTSDB	➢ OpenTSDB is a distributed, scalable Time Series Database (TSDB) written on top of HBase. OpenTSDB was written to address a common need: store, index and serve metrics collected from computer systems (network gear, operating systems, applications) at a large scale, and make this data easily accessible and graphable.	1. OpenTSDB site
Graph Data Model		
ArangoDB	➢ An open-source database with a flexible data model for documents, graphs, and key-values. Build high performance applications using a convenient sql-like query language or JavaScript extensions.	1. ArangoDB site

Neo4j	➢ An open-source graph database writting entirely in Java. It is an embedded, disk-based, fully transactional Java persistence engine that stores data structured in graphs rather than in tables.	1. Neo4j site
TitanDB	➢ TitanDB is a highly scalable graph database optimized for storing and querying large graphs with billions of vertices and edges distributed across a multi-machine cluster. Titan is a transactional database that can support thousands of concurrent users.	1. Titan site
NewSQL Databases		
TokuDB	➢ TokuDB is a storage engine for MySQL and MariaDB that is specifically designed for high performance on write-intensive workloads. It achieves this via Fractal Tree indexing. TokuDB is a scalable, ACID and MVCC compliant storage engine. TokuDB is one of the technologies that enable Big Data in MySQL.	TODO
HandlerSocket	➢ HandlerSocket is a NoSQL plugin for MySQL/MariaDB (the storage engine of MySQL). It works as a daemon inside the mysqld process, accepting TCP connections, and executing requests from clients. HandlerSocket does not support SQL queries. Instead, it supports simple CRUD operations on tables. HandlerSocket can be much faster than mysqld/libmysql in some cases because it has lower CPU, disk, and network overhead.	TODO
Akiban Server	➢ Akiban Server is an open source database that brings document stores and relational databases together. Developers get powerful document access alongside surprisingly powerful SQL.	TODO
Drizzle	➢ Drizzle is a re-designed version of the MySQL v6.0 codebase and is designed around a central concept of having a microkernel architecture. Features such as the query cache and authentication system are now plugins to the database, which follow the general theme of "pluggable storage engines" that were introduced in MySQL 5.1. It supports PAM, LDAP, and HTTP AUTH for authentication via plugins it ships. Via its plugin system it currently supports logging to files, syslog, and remote services such as RabbitMQ and Gearman. Drizzle is an ACID-compliant relational database that supports transactions via an MVCC design	TODO

Haeinsa	➢ Haeinsa is linearly scalable multi-row, multi-table transaction library for HBase. Use Haeinsa if you need strong ACID semantics on your HBase cluster. Is based on Google Perlocator concept.	
SenseiDB	➢ Open-source, distributed, realtime, semi-structured database. Some Features: Full-text search, Fast realtime updates, Structured and faceted search, BQL: SQL-like query language, Fast key-value lookup, High performance under concurrent heavy update and query volumes, Hadoop integration	1. SenseiDB site
Sky	➢ Sky is an open source database used for flexible, high performance analysis of behavioral data. For certain kinds of data such as clickstream data and log data, it can be several orders of magnitude faster than traditional approaches such as SQL databases or Hadoop.	1. SkyDB site
BayesDB	➢ BayesDB, a Bayesian database table, lets users query the probable implications of their tabular data as easily as an SQL database lets them query the data itself. Using the built-in Bayesian Query Language (BQL), users with no statistics training can solve basic data science problems, such as detecting predictive relationships between variables, inferring missing values, simulating probable observations, and identifying statistically similar database entries.	1. BayesDB site
InfluxDB	➢ InfluxDB is an open source distributed time series database with no external dependencies. It's useful for recording metrics, events, and performing analytics. It has a built-in HTTP API so you don't have to write any server side code to get up and running. InfluxDB is designed to be scalable, simple to install and manage, and fast to get data in and out. It aims to answer queries in real-time. That means every data point is indexed as it comes in and is immediately available in queries that should return under 100ms.	1. InfluxDB site
SQL-On-Hadoop		
Apache Hive	➢ Data Warehouse infrastructure developed by Facebook. Data summarization, query, and analysis. It's provides SQL-like language (not SQL92 compliant): HiveQL.	1. Apache HIVE site 2. Apache HIVE GitHub Project
Apache HCatalog	➢ HCatalog's table abstraction presents users with a relational view of data in the Hadoop Distributed File System (HDFS) and ensures that users need not worry about where or in what format their data is	

	stored. Right now HCatalog is part of Hive. Only old versions are separated for download.	
Trafodion: Transactional SQL-on-HBase	➢ Trafodion is an open source project sponsored by HP, incubated at HP Labs and HP-IT, to develop an enterprise-class SQL-on-HBase solution targeting big data transactional or operational workloads.	1. Trafodion wiki
Apache HAWQ	➢ Apache HAWQ is a Hadoop native SQL query engine that combines key technological advantages of MPP database evolved from Greenplum Database, with the scalability and convenience of Hadoop.	1. Apache HAWQ site 2. HAWQ GitHub Project
Apache Drill	➢ Drill is the open source version of Google's Dremel system which is available as an infrastructure service called Google BigQuery. In recent years open source systems have emerged to address the need for scalable batch processing (Apache Hadoop) and stream processing (Storm, Apache S4). Apache Hadoop, originally inspired by Google's internal MapReduce system, is used by thousands of organizations processing large-scale datasets. Apache Hadoop is designed to achieve very high throughput, but is not designed to achieve the sub-second latency needed for interactive data analysis and exploration. Drill, inspired by Google's internal Dremel system, is intended to address this need	1. Apache Incubator Drill
Cloudera Impala	➢ The Apache-licensed Impala project brings scalable parallel database technology to Hadoop, enabling users to issue low-latency SQL queries to data stored in HDFS and Apache HBase without requiring data movement or transformation. It's a Google Dremel clone (Big Query google).	1. Cloudera Impala site 2. Impala GitHub Project
Data Ingestion		
Apache Flume	➢ Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data. It has a simple and flexible architecture based on streaming data flows. It is robust and fault tolerant with tunable reliability mechanisms and many failover and recovery mechanisms. It uses a simple extensible data model that allows for online analytic application.	1. Apache Flume project site
Apache Sqoop	➢ System for bulk data transfer between HDFS and structured datastores as RDBMS. Like Flume but from HDFS to RDBMS.	1. Apache Sqoop project site
Facebook Scribe	➢ Log agregator in real-time. It's a Apache Thrift Service.	TODO
Apache Chukwa	➢ Large scale log aggregator, and analytics.	TODO

Apache Kafka	➢ Distributed publish-subscribe system for processing large amounts of streaming data. Kafka is a Message Queue developed by LinkedIn that persists messages to disk in a very performant manner. Because messages are persisted, it has the interesting ability for clients to rewind a stream and consume the messages again. Another upside of the disk persistence is that bulk importing the data into HDFS for offline analysis can be done very quickly and efficiently. Storm, developed by BackType (which was acquired by Twitter a year ago), is more about transforming a stream of messages into new streams.	1. Apache Kafka 2. GitHub source code
Netflix Suro	➢ Suro has its roots in Apache Chukwa, which was initially adopted by Netflix. Is a log agregattor like Storm, Samza.	TODO
Apache Samza	➢ Apache Samza is a distributed stream processing framework. It uses Apache Kafka for messaging, and Apache Hadoop YARN to provide fault tolerance, processor isolation, security, and resource management. Developed by http://www.linkedin.com/in/jaykreps Linkedin.	TODO
Cloudera Morphline	➢ Cloudera Morphlines is a new open source framework that reduces the time and skills necessary to integrate, build, and change Hadoop processing applications that extract, transform, and load data into Apache Solr, Apache HBase, HDFS, enterprise data warehouses, or analytic online dashboards.	TODO
HIHO	➢ This project is a framework for connecting disparate data sources with the Apache Hadoop system, making them interoperable. HIHO connects Hadoop with multiple RDBMS and file systems, so that data can be loaded to Hadoop and unloaded from Hadoop	TODO
Apache NiFi	➢ Apache NiFi is a dataflow system that is currently under incubation at the Apache Software Foundation. NiFi is based on the concepts of flow-based programming and is highly configurable. NiFi uses a component based extension model to rapidly add capabilities to complex dataflows. Out of the box NiFi has several extensions for dealing with file-based dataflows such as FTP, SFTP, and HTTP integration as well as integration with HDFS. One of NiFi's unique features is a rich, web-based interface for designing, controlling, and monitoring a dataflow.	1. Apache NiFi
Apache ManifoldCF	➢ Apache ManifoldCF provides a framework for connecting source content repositories like file systems, DB, CMIS, SharePoint, FileNet ... to target repositories or indexes, such as Apache Solr or ElasticSearch.	1. Apache ManifoldCF

	It's a kind of crawler for multi-content repositories, supporting a lot of sources and multi-format conversion for indexing by means of Apache Tika Content Extractor transformation filter.	
Service Programming		
Apache Thrift	➢ A cross-language RPC framework for service creations. It's the service base for Facebook technologies (the original Thrift contributor). Thrift provides a framework for developing and accessing remote services. ➢ It allows developers to create services that can be consumed by any application that is written in a language that there are Thrift bindings for. ➢ Thrift manages serialization of data to and from a service, as well as the protocol that describes a method invocation, response, etc. Instead of writing all the RPC code -- you can just get straight to your service logic. Thrift uses TCP and so a given service is bound to a particular port.	1. Apache Thrift
Apache Zookeeper	➢ It's a coordination service that gives you the tools you need to write correct distributed applications. ➢ ZooKeeper was developed at Yahoo! Research. Several Hadoop projects are already using ZooKeeper to coordinate the cluster and provide highly-available distributed services. ➢ Perhaps most famous of those are Apache HBase, Storm, Kafka. ZooKeeper is an application library with two principal implementations of the APIs—Java and C—and a service component implemented in Java that runs on an ensemble of dedicated servers. ➢ Zookeeper is for building distributed systems, simplifies the development process, making it more agile and enabling more robust implementations. Back in 2006, Google published a paper on "Chubby", a distributed lock service which gained wide adoption within their data centers. ➢ Zookeeper, not surprisingly, is a close clone of Chubby designed to fulfill many of the same roles for HDFS and other Hadoop infrastructure.	1. Apache Zookeeper 2. Google Chubby paper
Apache Avro	➢ Apache Avro is a framework for modeling, serializing and making Remote Procedure Calls (RPC). ➢ Avro data is described by a schema, and one interesting feature is that the schema is stored in the same file as the data it describes, so files are self-describing.	1. Apache Avro

	➢ Avro does not require code generation. This framework can compete with other similar tools like: ➢ Apache Thrift, Google Protocol Buffers, ZeroC ICE, and so on.	
Apache Curator	➢ Curator is a set of Java libraries that make using Apache ZooKeeper much easier.	TODO
Apache karaf	➢ Apache Karaf is an OSGi runtime that runs on top of any OSGi framework and provides you a set of services, a powerful provisioning concept, an extensible shell and more.	TODO
Twitter Elephant Bird	➢ Elephant Bird is a project that provides utilities (libraries) for working with LZOP-compressed data. It also provides a container format that supports working with Protocol Buffers, Thrift in MapReduce, Writables, Pig LoadFuncs, Hive SerDe, HBase miscellanea. This open source library is massively used in Twitter.	1. Elephant Bird GitHub
Linkedin Norbert	➢ Norbert is a library that provides easy cluster management and workload distribution. ➢ With Norbert, you can quickly distribute a simple client/server architecture to create a highly scalable architecture capable of handling heavy traffic. ➢ Implemented in Scala, Norbert wraps ZooKeeper, Netty and uses Protocol Buffers for transport to make it easy to build a cluster aware application. ➢ A Java API is provided and pluggable load balancing strategies are supported with round robin and consistent hash strategies provided out of the box.	1. Linedin Project 2. GitHub source code
Scheduling		
Apache Oozie	➢ Workflow scheduler system for MR jobs using DAGs (Direct Acyclical Graphs). Oozie Coordinator can trigger jobs by time (frequency) and data availability	1. Apache Oozie 2. GitHub source code
Linkedin Azkaban	➢ Hadoop workflow management. A batch job scheduler can be seen as a combination of the cron and make Unix utilities combined with a friendly UI.	TODO
Apache Falcon	➢ Apache Falcon is a data management framework for simplifying data lifecycle management and processing pipelines on Apache Hadoop. ➢ It enables users to configure, manage and orchestrate data motion, pipeline processing, disaster recovery, and data retention workflows.	TODO

	Instead of hard-coding complex data lifecycle capabilities, Hadoop applications can now rely on the well-tested Apache Falcon framework for these functions. ➢ Falcon's simplification of data management is quite useful to anyone building apps on Hadoop. Data Management on Hadoop encompasses data motion, process orchestration, lifecycle management, data discovery, etc. among other concerns that are beyond ETL. ➢ Falcon is a new data processing and management platform for Hadoop that solves this problem and creates additional opportunities by building on existing components within the Hadoop ecosystem (ex. Apache Oozie, Apache Hadoop DistCp etc.) without reinventing the wheel.	
Schedoscope	➢ Schedoscope is a new open-source project providing a scheduling framework for painfree agile development, testing, (re)loading, and monitoring of your datahub, lake, or whatever you choose to call your Hadoop data warehouse these days. Datasets (including dependencies) are defined using a scala DSL, which can embed MapReduce jobs, Pig scripts, Hive queries or Oozie workflows to build the dataset. The tool includes a test framework to verify logic and a command line utility to load and reload data.	GitHub source code
Machine Learning		
Apache Mahout	➢ Machine learning library and math library, on top of MapReduce.	TODO
WEKA	➢ Weka (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. Weka is free software available under the GNU General Public License.	TODO
Cloudera Oryx	➢ The Oryx open source project provides simple, real-time large-scale machine learning / predictive analytics infrastructure. ➢ It implements a few classes of algorithm commonly used in business applications: collaborative filtering / recommendation, classification / regression, and clustering.	1. Oryx at GitHub 2. Cloudera forum for Machine Learning
MADlib	➢ The MADlib project leverages the data-processing capabilities of an RDBMS to analyze data. The aim of this project is the integration of statistical data analysis into databases.	1. MADlib Community

	➢ The MADlib project is self-described as the Big Data Machine Learning in SQL for Data Scientists. The MADlib software project began the following year as a collaboration between researchers at UC Berkeley and engineers and data scientists at EMC/Greenplum (now Pivotal)	
Benchmarking And QA Tools		
Apache Hadoop Benchmarking	➢ There are two main JAR files in Apache Hadoop for benchmarking. This JAR are micro-benchmarks for testing particular parts of the infrastructure, for instance TestDFSIO analyzes the disk system, TeraSort evaluates MapReduce tasks, WordCount measures cluster performance, etc. ➢ Micro-Benchmarks are packaged in the tests and exmaples JAR files, and you can get a list of them, with descriptions, by invoking the JAR file with no arguments. ➢ With regards Apache Hadoop 2.2.0 stable version we have available the following JAR files for test, examples and benchmarking. ➢ The Hadoop micro-benchmarks, are bundled in this JAR files: hadoop-mapreduce-examples-2.2.0.jar, hadoop-mapreduce-client-jobclient-2.2.0-tests.jar.	1. MAPREDUCE-3561 umbrella ticket to track all the issues related to performance
Yahoo Gridmix3	➢ Hadoop cluster benchmarking from Yahoo engineer team.	TODO
PUMA Benchmarking	➢ Benchmark suite which represents a broad range of MapReduce applications exhibiting application characteristics with high/low computation and high/low shuffle volumes. There are a total of 13 benchmarks, out of which Tera-Sort, Word-Count, and Grep are from Hadoop distribution. ➢ The rest of the benchmarks were developed in-house and are currently not part of the Hadoop distribution. ➢ The three benchmarks from Hadoop distribution are also slightly modified to take number of reduce tasks as input from the user and generate final time completion statistics of jobs.	1. MAPREDUCE-5116 2. Faraz Ahmad researcher 3. PUMA Docs
Security		
Apache Ranger	➢ Apache Argus Ranger (formerly called Apache Argus or HDP Advanced Security) delivers comprehensive approach to central security policy administration across the core enterprise security requirements of authentication, authorization, accounting and data protection.	1. Apache Ranger 2. Apache Ranger Hortonworks web

	➢ It extends baseline features for coordinated enforcement across Hadoop workloads from batch, interactive SQL and real–time and leverages the extensible architecture to apply policies consistently against additional Hadoop ecosystem components (beyond HDFS, Hive, and HBase) including Storm, Solr, Spark, and more.	
System Deployment		
Apache Ambari	➢ Intuitive, easy-to-use Hadoop management web UI backed by its RESTful APIs. Apache Ambari was donated by Hortonworks team to the ASF. It's a powerful and nice interface for Hadoop and other typical applications from the Hadoop ecosystem. ➢ Apache Ambari is under a heavy development, and it will incorporate new features in a near future. ➢ For example Ambari is able to deploy a complete Hadoop system from scratch, however is not possible use this GUI in a Hadoop system that is already running. ➢ The ability to provisioning the operating system could be a good addition, however probably is not in the roadmap..	1. Apache Ambari
Cloudera HUE	➢ Web application for interacting with Apache Hadoop. ➢ It's not a deploment tool, is an open-source Web interface that supports Apache Hadoop and its ecosystem, licensed under the Apache v2 license. ➢ HUE is used for Hadoop and its ecosystem user operations. For example HUE offers editors for Hive, Impala, Oozie, Pig, notebooks for Spark, Solr Search dashboards, HDFS, YARN, HBase browsers..	1. HUE home page
Apache Whirr	➢ Apache Whirr is a set of libraries for running cloud services. It allows you to use simple commands to boot clusters of distributed systems for testing and experimentation. ➢ Apache Whirr makes booting clusters easy.	TODO
Applications		
Apache Nutch	➢ Highly extensible and scalable open source web crawler software project. A search engine based on Lucene: A Web crawler is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing. Web crawlers can copy all the pages they visit for later processing by a search engine that indexes the downloaded pages so that users can search them much more quickly.	TODO

Sphnix Search Server	➢ Sphinx lets you either batch index and search data stored in an SQL database, NoSQL storage, or just files quickly and easily — or index and search data on the fly, working with Sphinx pretty much as with a database server.	TODO
Apache OODT	➢ OODT was originally developed at NASA Jet Propulsion Laboratory to support capturing, processing and sharing of data for NASA's scientific archives	TODO
HIPI Library	➢ HIPI is a library for Hadoop's MapReduce framework that provides an API for performing image processing tasks in a distributed computing environment.	TODO
PivotalR	➢ PivotalR is a package that enables users of R, the most popular open source statistical programming language and environment to interact with the Pivotal (Greenplum) Database as well as Pivotal HD / HAWQ and the open-source database PostgreSQL for Big Data analytics. R is a programming language and data analysis software: you do data analysis in R by writing scripts and functions in the R programming language. R is a complete, interactive, object-oriented language: designed by statisticians, for statisticians. The language provides objects, operators and functions that make the process of exploring, modeling, and visualizing data a natural one.	1. PivotalR on GitHub
Development Frameworks		
Jumbune	➢ Jumbune is an open source product that sits on top of any Hadoop distribution and assists in development and administration of MapReduce solutions. The objective of the product is to assist analytical solution providers to port fault free applications on production Hadoop environments. ➢ Jumbune supports all active major branches of Apache Hadoop namely 1.x, 2.x, 0.23.x and commercial MapR, HDP 2.x and CDH 5.x distributions of Hadoop. It has the ability to work well with both Yarn and non-Yarn versions of Hadoop. ➢ It has four major modules MapReduce Debugger, HDFS Data Validator, On-demand cluster monitor and MapReduce job profiler. Jumbune can be deployed on any remote user machine and uses a lightweight agent on the NameNode of the cluster to relay relevant information to and fro.	

2.5 Physical Architecture

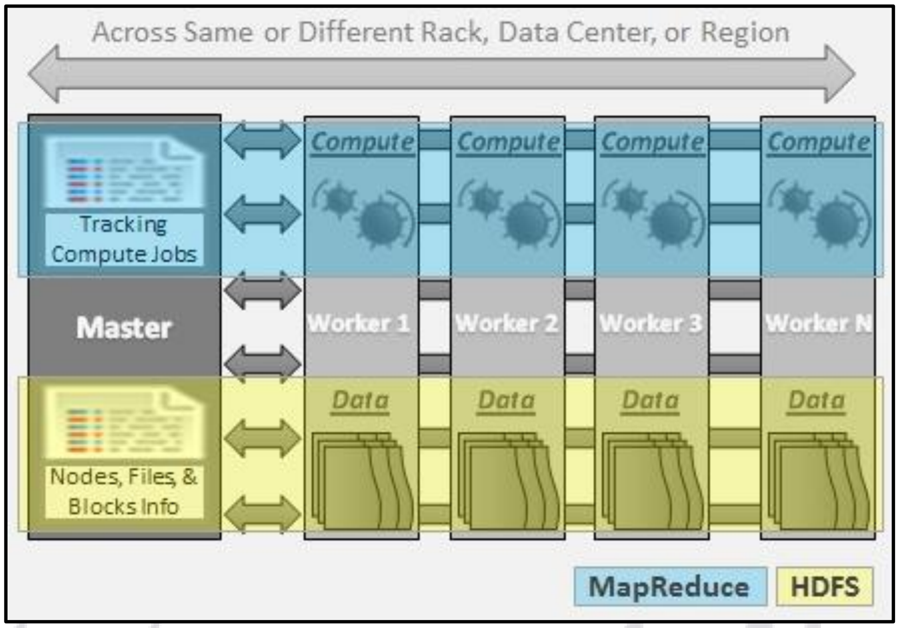


Figure 2.1: Physical Architecture

I. Hadoop Cluster - Architecture, Core Components and Work-flow

1. The architecture of Hadoop Cluster
2. Core Components of Hadoop Cluster
3. Work-flow of How File is Stored in Hadoop

A. Hadoop Cluster

i. Hadoop cluster is a special type of computational cluster designed for storing and analyzing vast amount of unstructured data in a distributed computing environment.

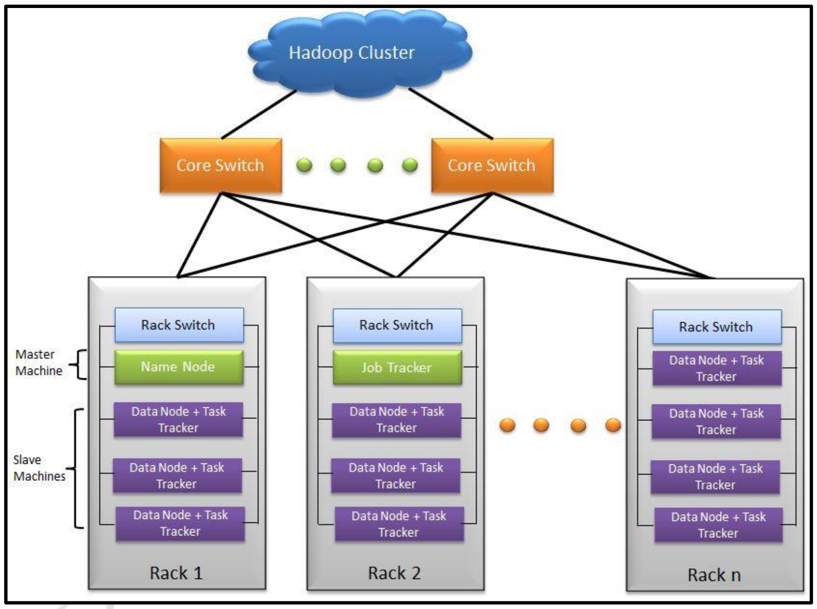


Figure 2.2: Hadoop Cluster

ii. These clusters run on low cost commodity computers.

iii. Hadoop clusters are often referred to as "shared nothing" systems because the only thing that is shared between nodes is the network that connects them.

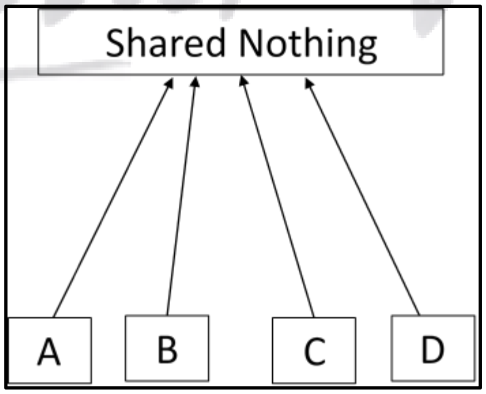


Figure 2.3: Shared Nothing

iv. Large Hadoop Clusters are arranged in several racks. Network traffic between different nodes in the same rack is much more desirable than network traffic across the racks.

A Real Time Example: Yahoo's Hadoop cluster. They have more than 10,000 machines running Hadoop and nearly 1 petabyte of user data.

Figure 2.4: Yahoo Hadoop Cluster

v. A small Hadoop cluster includes a single master node and multiple worker or slave node. As discussed earlier, the entire cluster contains two layers.

vi. One of the layer of MapReduce Layer and another is of HDFS Layer.

vii. Each of these layer have its own relevant component.

viii. The master node consists of a JobTracker, TaskTracker, NameNode and DataNode.

ix. A slave or worker node consists of a DataNode and TaskTracker.

x. It is also possible that slave node or worker node is only data or compute node. The matter of the fact that is the key feature of the Hadoop.

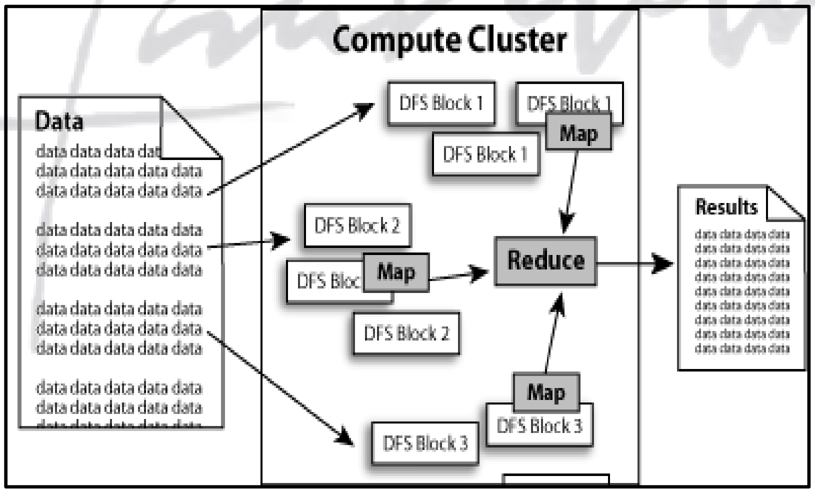


Figure 2.4: NameNode Cluster

B. Hadoop Cluster Architecture:

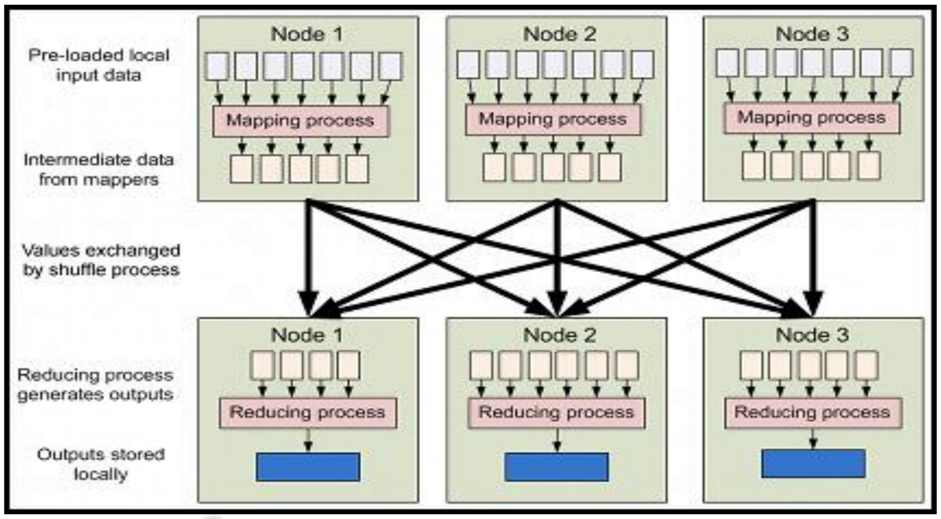

Node 1
Node 2
Node 3
Pre-loaded local input data
Mapping process
Intermediate data from mappers
Values exchanged by shuffle process
Reducing process generates outputs
Reducing process
Outputs stored locally

Figure 2.5: Hadoop Cluster Architecture

Hadoop Cluster would consists of

- 110 different racks
- Each rack would have around 40 slave machine
- At the top of each rack there is a rack switch
- Each slave machine(rack server in a rack) has cables coming out it from both the ends
- Cables are connected to rack switch at the top which means that top rack switch will have around 80 ports
- There are global 8 core switches
- The rack switch has uplinks connected to core switches and hence connecting all other racks with uniform bandwidth, forming the Cluster
- In the cluster, you have few machines to act as Name node and as JobTracker. They are referred as Masters. These masters have different configuration favoring more DRAM and CPU and less local storage.

Hadoop cluster has 3 components:

1. Client
2. Master
3. Slave

The role of each components are shown in the below image.

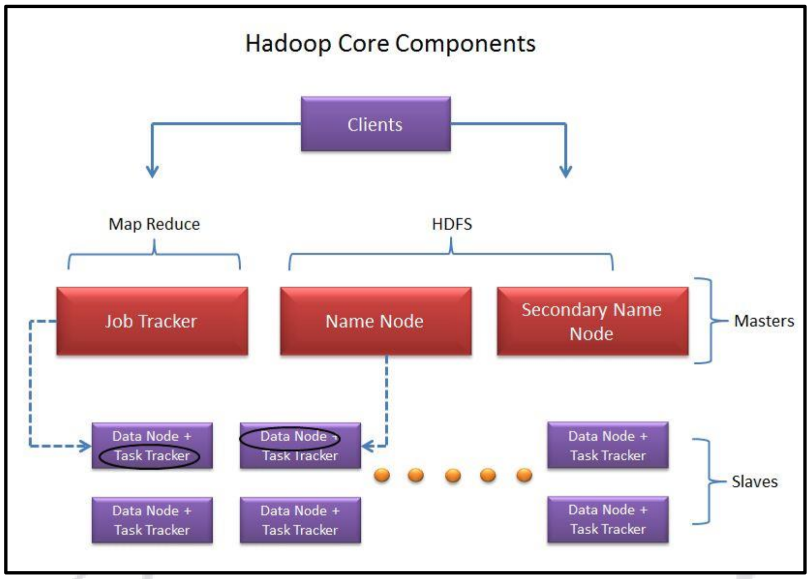


Figure 2.6: Hadoop Core Component

1. **Client:**

   i. It is neither master nor slave, rather play a role of loading the data into cluster, submit MapReduce jobs describing how the data should be processed and then retrieve the data to see the response after job completion.

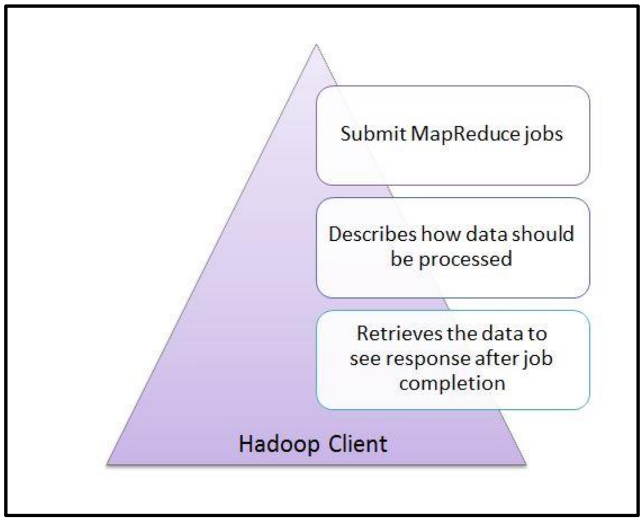

Submit MapReduce jobs
Describes how data should be processed
Retrieves the data to see response after job completion
Hadoop Client

Figure 2.6: Hadoop Client

2. Masters:

The Masters consists of 3 components NameNode, Secondary Node name and JobTracker.

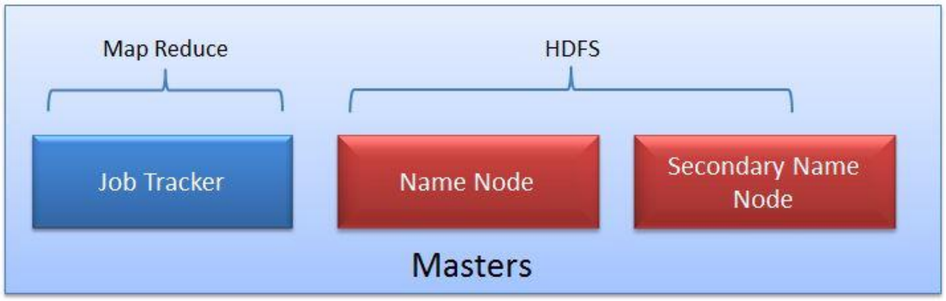


Figure 2.7: MapReduce - HDFS

i. NameNode:

- NameNode does NOT store the files but only the file's metadata. In later section we will see it is actually the DataNode which stores the files.

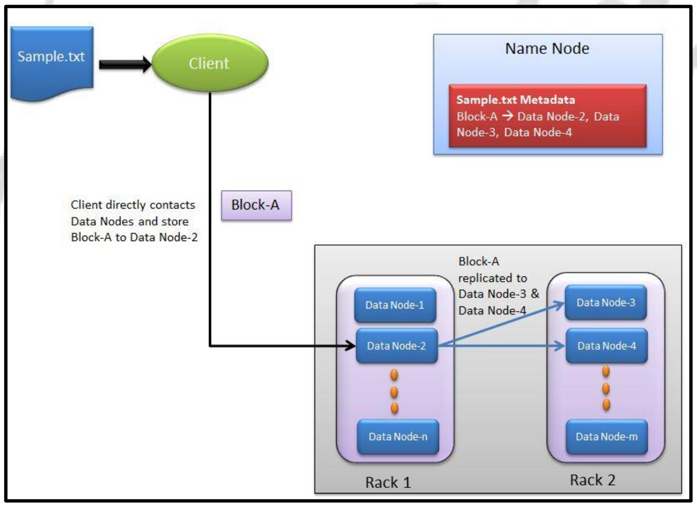

Sample.txt
Client
Name Node
Sample.txt Metadata
Block-A → Data Node-2, Data Node-3, Data Node-4
Client directly contacts Data Nodes and store Block-A to Data Node-2
Block-A
Block-A replicated to Data Node-3 & Data Node-4
Data Node-1
Data Node-2
Data Node-n
Data Node-3
Data Node-4
Data Node-m
Rack 1
Rack 2

Figure 2.8: NameNode

- NameNode oversees the health of DataNode and coordinates access to the data stored in DataNode.
- Name node keeps track of all the file system related information such as to
  - ✓ Which section of file is saved in which part of the cluster
  - ✓ Last access time for the files
  - ✓ User permissions like which user have access to the file

ii. JobTracker:

JobTracker coordinates the parallel processing of data using MapReduce.

To know more about JobTracker, please read the article All You Want to Know about MapReduce (The Heart of Hadoop)

iii. Secondary Name Node:

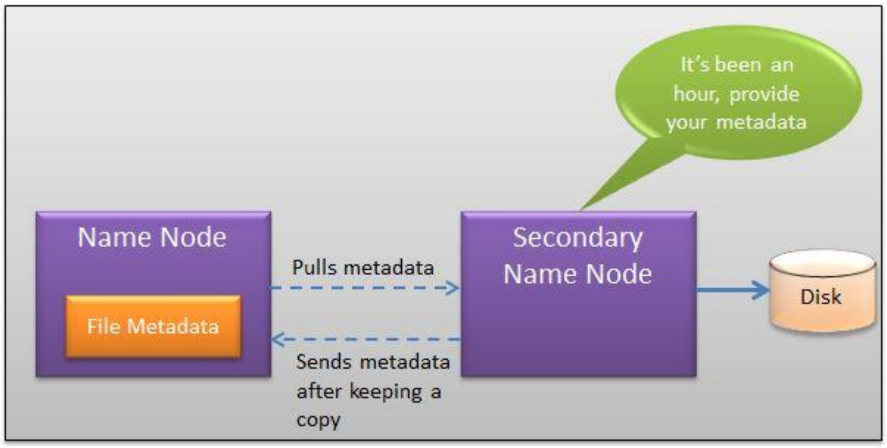

It's been an hour, provide your metadata
Name Node
File Metadata
Pulls metadata
Sends metadata after keeping a copy
Secondary Name Node
Disk

Figure 2.9: Secondary NameNode

- The job of Secondary Node is to contact NameNode in a periodic manner after certain time interval (by default 1 hour).
- NameNode which keeps all filesystem metadata in RAM has no capability to process that metadata on to disk.
- If NameNode crashes, you lose everything in RAM itself and you don't have any backup of filesystem.
- What secondary node does is it contacts NameNode in an hour and pulls copy of metadata information out of NameNode.
- It shuffle and merge this information into clean file folder and sent to back again to NameNode, while keeping a copy for itself.
- Hence Secondary Node is not the backup rather it does job of housekeeping.
- In case of NameNode failure, saved metadata can rebuild it easily.

3. Slaves:

i. Slave nodes are the majority of machines in Hadoop Cluster and are responsible to

- Store the data
- Process the computation

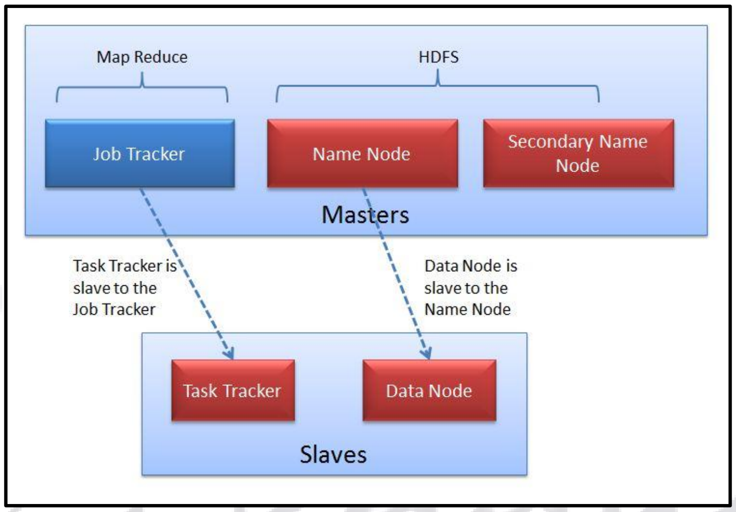


Figure 2.10: Slaves

ii. Each slave runs both a DataNode and Task Tracker daemon which communicates to their masters.

iii. The Task Tracker daemon is a slave to the Job Tracker and the DataNode daemon a slave to the NameNode

II. Hadoop- Typical Workflow in HDFS:

Take the example of input file as Sample.txt.

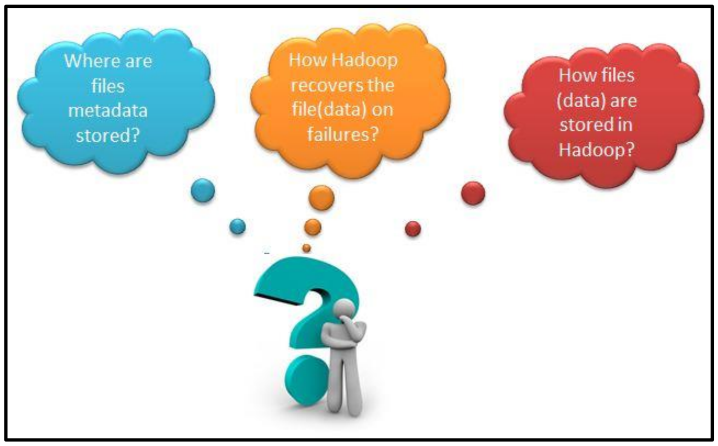

Where are files metadata stored?
How Hadoop recovers the file(data) on failures?
How files (data) are stored in Hadoop?

Figure 2.11: HDFS Workflow

1. **How TestingHadoop.txt gets loaded into the Hadoop Cluster?**

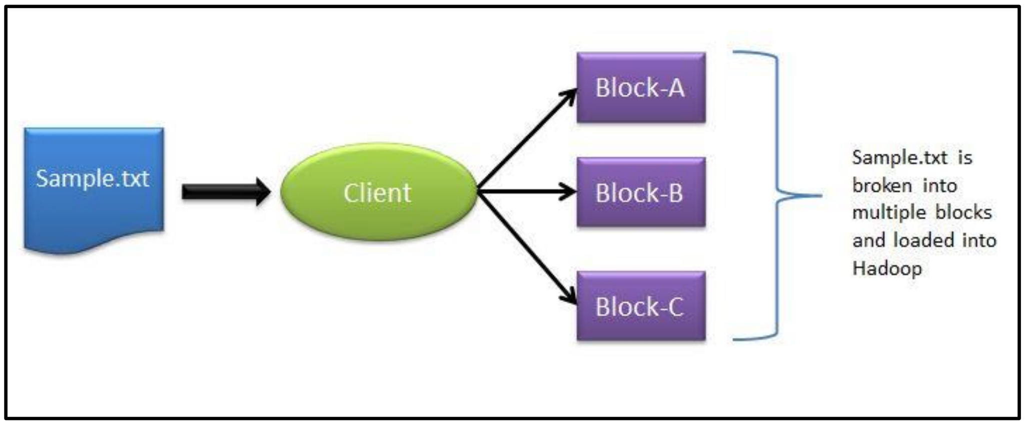

Sample.txt
Client
Block-A
Block-B
Block-C
Sample.txt is broken into multiple blocks and loaded into Hadoop

Figure 2.12: Loading file in Hadoop Cluster

- Client machine does this step and loads the Sample.txt into cluster.
- It breaks the sample.txt into smaller chunks which are known as "Blocks" in Hadoop context.
- Client put these blocks on different machines (data nodes) throughout the cluster.

2. **Next, how does the Client knows that to which data nodes load the blocks?**

- Now NameNode comes into picture.
- The NameNode used its Rack Awareness intelligence to decide on which DataNode to provide.
- For each of the data block (in this case Block-A, Block-B and Block-C), Client contacts NameNode and in response NameNode sends an ordered list of 3 DataNodes.

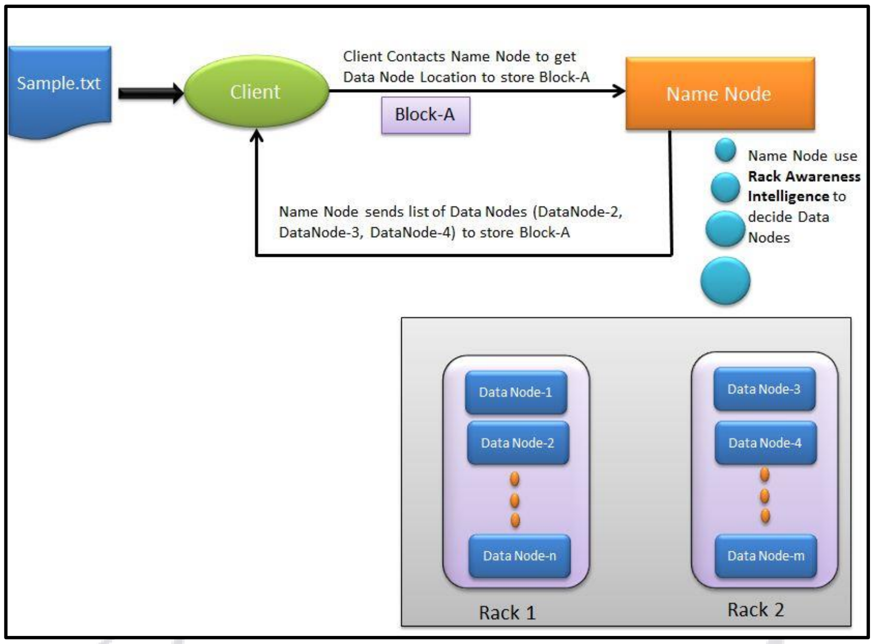

Sample.txt
Client
Client Contacts Name Node to get Data Node Location to store Block-A
Block-A
Name Node
Name Node use Rack Awareness Intelligence to decide Data Nodes
Name Node sends list of Data Nodes (DataNode-2, DataNode-3, DataNode-4) to store Block-A
Data Node-1
Data Node-2
Data Node-n
Rack 1
Data Node-3
Data Node-4
Data Node-m
Rack 2

Figure 2.13: how does the Client knows that to which data nodes load the blocks?

- ➢ For example in response to Block-A request, Node Name may send DataNode-2, DataNode-3 and DataNode-4.
  - ✓ Block-B DataNodes list DataNode-1, DataNode-3, DataNode-4 and for Block C data node list DataNode-1, DataNode-2, DataNode-3. Hence
    - ❖ Block A gets stored in DataNode-2, DataNode-3, DataNode-4
    - ❖ Block B gets stored in DataNode-1, DataNode-3, DataNode-4
    - ❖ Block C gets stored in DataNode-1, DataNode-2, DataNode-3
  - ✓ Every block is replicated to more than 1 data nodes to ensure the data recovery on the time of machine failures. That's why NameNode send 3 DataNodes list for each individual block

3. Who does the block replication?

- ➢ Client write the data block directly to one DataNode. DataNodes then replicate the block to other Data nodes.
- ➢ When one block gets written in all 3 DataNode then only cycle repeats for next block.

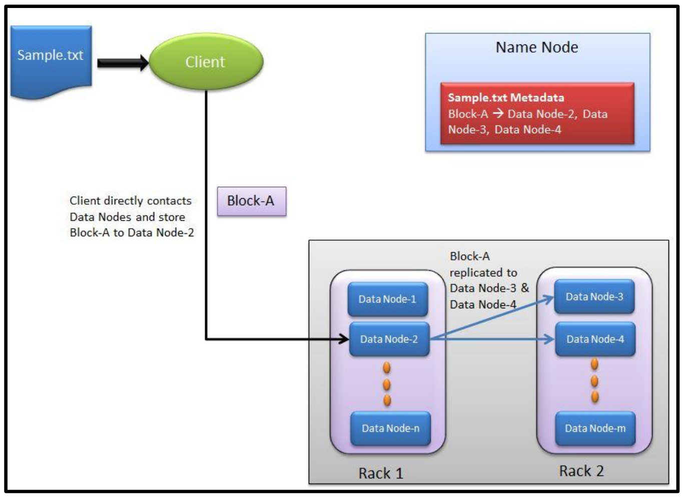

Sample.txt
Client
Name Node
Sample.txt Metadata
Block-A → Data Node-2, Data Node-3, Data Node-4
Client directly contacts Data Nodes and store Block-A to Data Node-2
Block-A
Block-A replicated to Data Node-3 & Data Node-4
Data Node-1
Data Node-2
Data Node-n
Data Node-3
Data Node-4
Data Node-m
Rack 1
Rack 2

Figure 2.14: Who does the block replication?

- In Hadoop Gen 1 there is only one NameNode wherein Gen2 there is active passive model in NameNode where one more node "Passive Node" comes in picture.
- The default setting for Hadoop is to have 3 copies of each block in the cluster. This setting can be configured with "dfs.replication" parameter of hdfs-site.xml file.
- Keep note that Client directly writes the block to the DataNode without any intervention of NameNode in this process.

2.6 Hadoop limitations

i. Network File system is the oldest and the most commonly used distributed file system and was designed for the general class of applications, Hadoop only specific kind of applications can make use of it.

ii. It is known that Hadoop has been created to address the limitations of the distributed file system, where it can store the large amount of data, offers failure protection and provides fast access, but it should be known that the benefits that come with Hadoop come at some cost.

iii. Hadoop is designed for applications that require random reads; so if a file has four parts the file would like to read all the parts one-by-one going from 1 to 4 till the end. Random seek is where you want to go to a specific location in the file; this is something that isn't possible with Hadoop. Hence, Hadoop is designed for non-real-time batch processing of data.

iv. Hadoop is designed for streaming reads caching of data isn't provided. Caching of data is provided which means that when you want to read data another time, it can be read very fast from the cache. This caching isn't possible because you get faster access to the data directly by doing the sequential read; hence caching isn't available through Hadoop.

v. It will write the data and then it will read the data several times. It will not be updating the data that it has written; hence updating data written to closed files is not available. However, you have to know that in update 0.19 appending will be supported for those files that aren't closed. But for those files that have been closed, updating isn't possible.

vi. In case of Hadoop we aren't talking about one computer; in this scenario we usually have a large number of computers and hardware failures are unavoidable; sometime one computer will fail and sometimes the entire rack can fail too. Hadoop gives excellent protection against hardware failure; however the performance will go down proportionate to the number of computers that are down. In the big picture, it doesn't really matter and it is not generally noticeable since if you have 100 computers and in them if 3 fail then 97 are still working. So the proportionate loss of performance isn't that noticeable. However, the way Hadoop works there is the loss in performance. Now this loss of performance through hardware failures is something that is managed through replication strategy.

3. NoSQL

CONTENTS

3.1 What is NoSQL? NoSQL business drivers; NoSQL case studies

1. NoSQL is a whole new way of thinking about a database.
2. Though NoSQL is not a relational database, the reality is that a relational database model may not be the best solution for all situations.
3. Example: Imagine that you have coupons that you wanted to push to mobile customers that purchase a specific item. This is a customer facing system of engagement requires location data, purchase data, wallet data, and so on. You want to engage the mobile customer in real-time.
4. NoSQL is a whole new way of thinking about a database. NoSQL is not a relational database.
5. The easiest way to think of NoSQL, is that of a database which does not adhering to the traditional relational database management system (RDMS) structure. Sometimes you will also see it revered to as 'not only SQL'.
6. It is not built on tables and does not employ SQL to manipulate data. It also may not provide full ACID (atomicity, consistency, isolation, durability) guarantees, but still has a distributed and fault tolerant architecture.
7. The NoSQL taxonomy supports key-value stores, document store, Big Table, and graph databases.
8. MongoDB, for example, uses a document model, which can be thought of as a row in a RDBMS. Documents, a set of fields (key-value pairs) map nicely to programming language data types.
9. A MongoDB database holds a collection which is a set of documents. Embedded documents and arrays reduce need for joins, which is key for high performance and speed.

I. Why NoSQL?

1. It's high performance with high availability, and offers rich query language and easy scalability.
2. NoSQL is gaining momentum, and is supported by Hadoop, MongoDB and others.

3. The NoSQL Database site is a good reference for someone looking for more information.

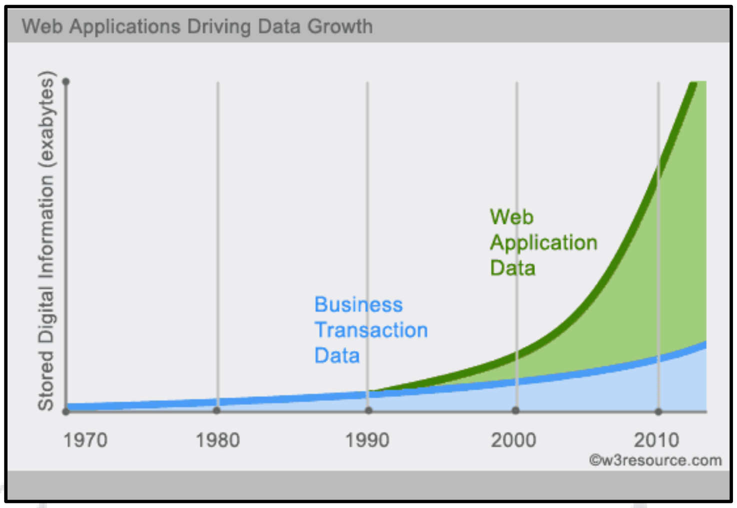


Figure 3.1: Web application Data Growth

NoSQL is:

- More than rows in tables—NoSQL systems store and retrieve data from many formats; key-value stores, graph databases, column-family (Bigtable) stores, document stores and even rows in tables.
- Free of joins—NoSQL systems allow you to extract your data using simple interfaces without joins.
- Schema free—NoSQL systems allow you to drag-and-drop your data into a folder and then query it without creating an entity-relational model.
- Compatible with many processors—NoSQL systems allow you to store your database on multiple processors and maintain high-speed performance.
- Usable on shared-nothing commodity computers—Most (but not all) NoSQL systems leverage low cost commodity processors that have separate RAM and disk.
- Supportive of linear scalability—NoSQL supports linear scalability; when you add more processors you get a consistent increase in performance.
- Innovative—NoSQL offers options to a single way of storing, retrieving and manipulating data. NoSQL supporters (also known as NoSQLers) have an inclusive

attitude about NoSQL and recognize SQL solutions as viable options. To the NoSQL community, NoSQL means not only SQL.

NoSQL is not:

- About the SQL language—the definition of NoSQL is not an application that uses a language other than SQL. SQL as well as other query languages are used with NoSQL databases.
- Not only open source—although many NoSQL systems have an open source model, commercial products use NOSQL concepts as well as open source initiatives. You can still have an innovative approach to problem solving with a commercial product.
- Not only Big Data—many, but not all NoSQL applications, are driven by the inability of a current application to efficiently scale when Big Data is an issue. While volume and velocity are important, NoSQL also focuses on variability and agility.
- About cloud computing—Many NoSQL systems reside in the cloud to take advantage of its ability to rapidly scale when the situations dictates. NoSQL systems can run in the cloud as well as in your corporate data center.
- About a clever use of RAM and SSD—Many NoSQL systems focus on the efficient use of RAM or solid-state disks to increase performance. While important, NoSQL systems can run on standard hardware.
- An elite group of products—NoSQL is not an exclusive club with a few products. There are no membership dues or tests required to join.

Table 3.1: Types of NoSQL data stores

Type	Typical usage	Examples
Key-value stor—A simple data storage system that uses a key to access a value	• Image stores • Key-based file systems • Object cache • Systems designed to scale	• Memcache • Redis • Riak • DynamoDB
Column family store—A sparse matrix system that uses a row and column as keys	• Web crawler results • Big Data problems that can relax consistency rules	• HBase • Cassandra • Hypertable
Graph store—For relationship intensive problems	• Social networks • Fraud detection • Relationship heavy data	• Neo4J • AllegroGraph • Big data (RDF Store) • InfiniteGraph (Objectivity)
Document store—Storing hierarchical data structures directly in the database	• High variability data • Document search • Integration hubs • Web content management • Publishing	• MongoDB (10Gen) • CouchDB • Couchbase • MarkLogic • eXist-db

II. RDBMS vs NoSQL

RDBMS

1. Structured and organized data
2. Structured query language (SQL)
3. Data and its relationships are stored in separate tables.
4. Data Manipulation Language, Data Definition Language
5. Tight Consistency
6. BASE Transaction

NoSQL

1. Stands for Not Only SQL
2. No declarative query language
3. No predefined schema
4. Key-Value pair storage, Column Store, Document Store, Graph databases
5. Eventual consistency rather ACID property
6. Unstructured and unpredictable data
7. CAP Theorem
8. Prioritizes high performance, high availability and scalability

CAP Theorem (Brewer's Theorem)

CAP theorem states that there are three basic requirements which exist in a special relation when designing applications for a distributed architecture.

1. **Consistency** - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.
2. **Availability** - This means that the system is always on (service guarantee availability), no downtime.
3. **Partition Tolerance** - This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

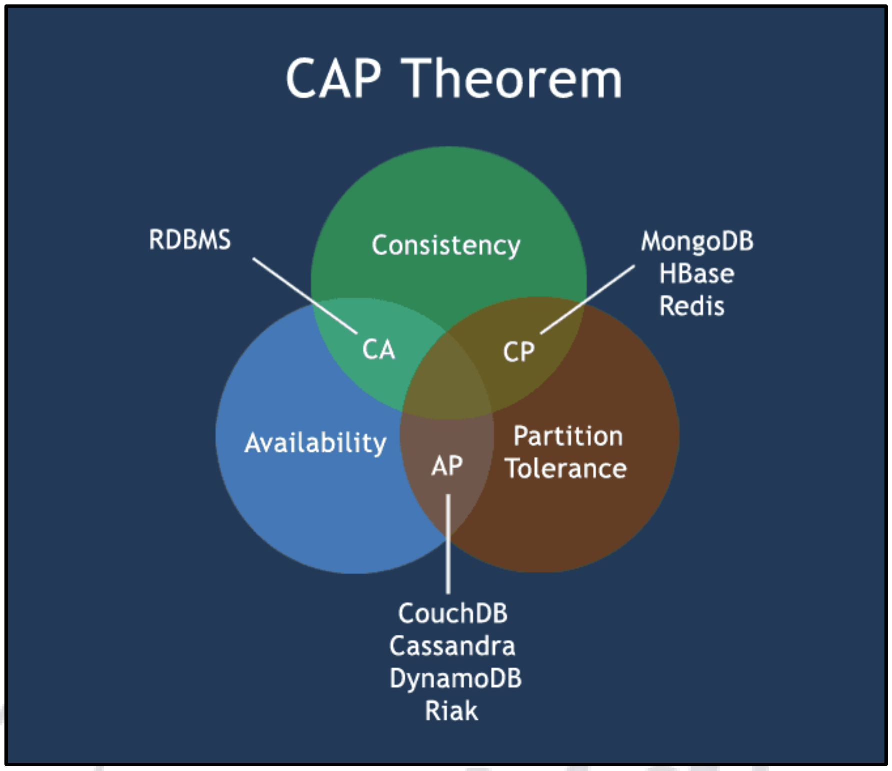


Figure 3.2: CAP Theorem

i. **CA** - Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

ii. **CP** - Some data may not be accessible, but the rest is still consistent/accurate.

iii. **AP** - System is still available under partitioning, but some of the data returned may be inaccurate.

Advantages:

1. High scalability
2. Distributed Computing
3. Lower cost
4. Schema flexibility, semi-structure data
5. No complicated Relationships

Disadvantages

1. No standardization
2. Limited query capabilities (so far)

3. Eventual consistent is not intuitive for program

III. A BASE system gives up on consistency.

i. **B**asically **A**vailable indicates that the system does guarantee availability, in terms of the CAP theorem.

ii. **S**oft state indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.

iii. **E**ventual consistency indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

ACID vs BASE

ACID	**BASE**
Atomic	**B**asically **A**vailable
Consistency	**S**oft state
Isolation	**E**ventual consistency
Durable	

IV. NoSQL business drivers

i. The scientist-philosopher Thomas Kuhn coined the term paradigm shift to identify a recurring process he observed in science, where innovative ideas came in bursts and impacted the world in non-linear ways.

ii. Kuhn's concept of the paradigm shift as a way to think about and explain the NoSQL movement and the changes in thought patterns, architectures and methods emerging today.

iii. Many organizations supporting single CPU relational systems have come to a crossroad; the needs of their organization are changing.

iv. Businesses have found value in rapidly capturing and analyzing large amounts of variable data, and making immediate changes in their business based on the information they receive.
v. Figure 1 shows how the demands of volume, velocity, variability, and agility play a key role in the emergence of NoSQL solutions.
vi. As each of these drivers apply pressure to the single processor relational model, its foundation becomes less stable and in time no longer meets the organization's needs.

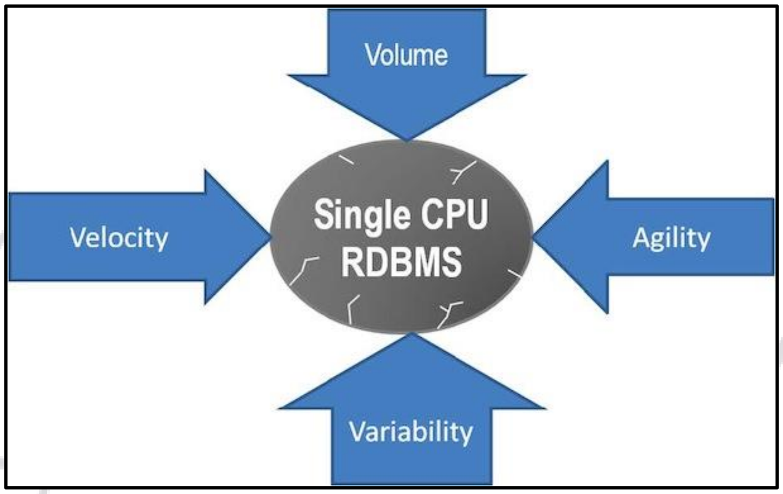


Figure 3.3: The business drivers—volume, velocity, variability, and agility—apply pressure to the single CPU system resulting in the cracks.

3.2 NoSQL data architecture patterns

I. NOSQL Patterns

1. These storage solution differs quite significantly with the RDBMS model and is also known as the NOSQL. Some of the key players include ...
   - GoogleBigTable, HBase, Hypertable
   - AmazonDynamo, Voldemort, Cassendra, Riak
   - Redis
   - CouchDB, MongoDB
2. These solutions has a number of characteristics in common

i. Key value store
ii. Run on large number of commodity machines
iii. Data are partitioned and replicated among these machines
iv. Relax the data consistency requirement. (because the CAP theorem proves that you cannot get Consistency, Availability and Partitioning at the the same time)

A. Key-value stores

1. Key-value stores are most basic types of NoSQL databases.
2. Designed to handle huge amounts of data.
3. Based on Amazon's Dynamo paper.
4. Key value stores allow developer to store schema-less data.
5. In the key-value storage, database stores data as hash table where each key is unique and the value can be string, JSON, BLOB (basic large object) etc.
6. A key may be strings, hashes, lists, sets, sorted sets and values are stored against these keys.
7. For example a key-value pair might consist of a key like "Name" that is associated with a value like "Robin".
8. Key-Value stores can be used as collections, dictionaries, associative arrays etc.
9. Key-Value stores follows the 'Availability' and 'Partition' aspects of CAP theorem.
10. Key-Values stores would work well for shopping cart contents, or individual values like color schemes, a landing page URI, or a default account number.
11. Example of Key-value store DataBase : Redis, Dynamo, Riak. etc.

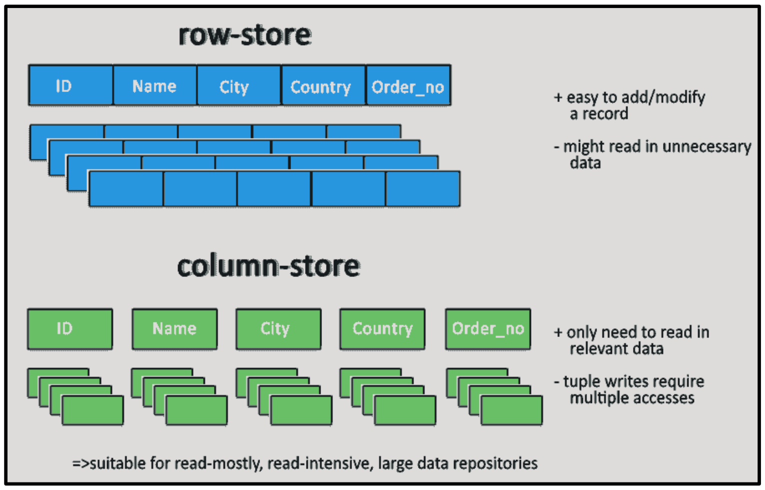

row-store
ID
Name
City
Country
Order_no
+ easy to add/modify a record
- might read in unnecessary data
column-store
ID
Name
City
Country
Order_no
+ only need to read in relevant data
- tuple writes require multiple accesses
=>suitable for read-mostly, read-intensive, large data repositories

Figure 3.4: Pictorial Representation

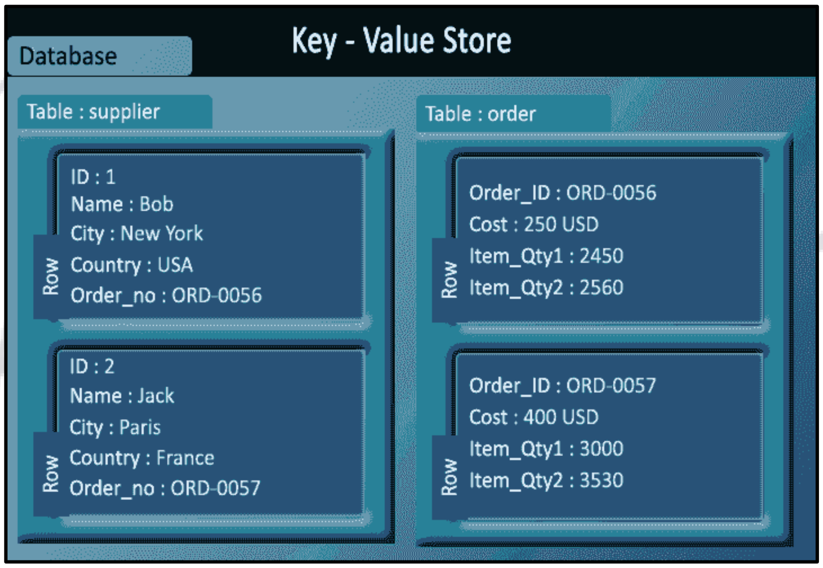


Figure 3.5: Key value store

B. Column-oriented databases

1. Column-oriented databases primarily work on columns and every column is treated individually.
2. Values of a single column are stored contiguously.
3. Column stores data in column specific files.
4. In Column stores, query processors work on columns too.
5. All data within each column data file have the same type which makes it ideal for compression.

6. Column stores can improve the performance of queries as it can access specific column data.
7. High performance on aggregation queries (e.g. COUNT, SUM, AVG, MIN, MAX).
8. Works on data warehouses and business intelligence, customer relationship management (CRM), Library card catalogs etc.
9. Example of Column-oriented databases : BigTable, Cassandra, SimpleDB etc.

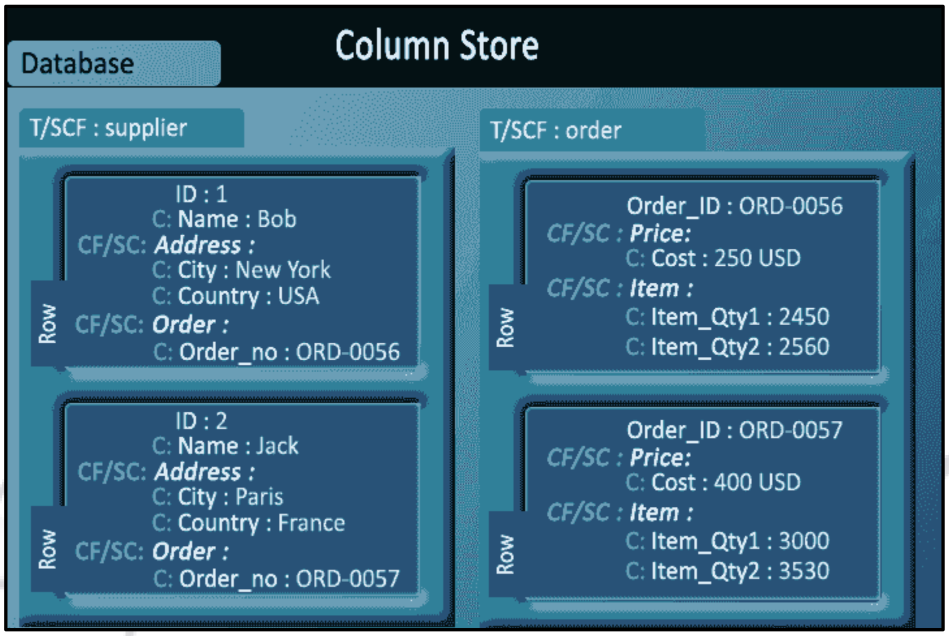


Figure 3.6: Column store

C. Graph databases

1. A graph database stores data in a graph.
2. It is capable of elegantly representing any kind of data in a highly accessible way.
3. A graph database is a collection of nodes and edges
4. Each node represents an entity (such as a student or business) and each edge represents a connection or relationship between two nodes.
5. Every node and edge is defined by a unique identifier.
6. Each node knows its adjacent nodes.
7. As the number of nodes increases, the cost of a local step (or hop) remains the same.
8. Index for lookups.
9. Example of Graph databases: OrientDB, Neo4J, Titan.etc.

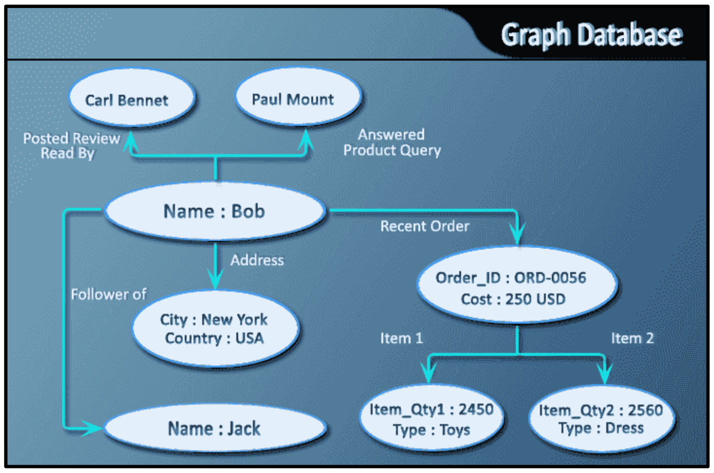


Figure 3.7: Graph Database

D. Document oriented databases

1. Document Oriented databases
2. A collection of documents
3. Data in this model is stored inside documents.
4. A document is a key value collection where the key allows access to its value.
5. Documents are not typically forced to have a schema and therefore are flexible and easy to change.
6. Documents are stored into collections in order to group different kinds of data.
7. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.
8. Example of Document Oriented databases: MongoDB, CouchDB etc.

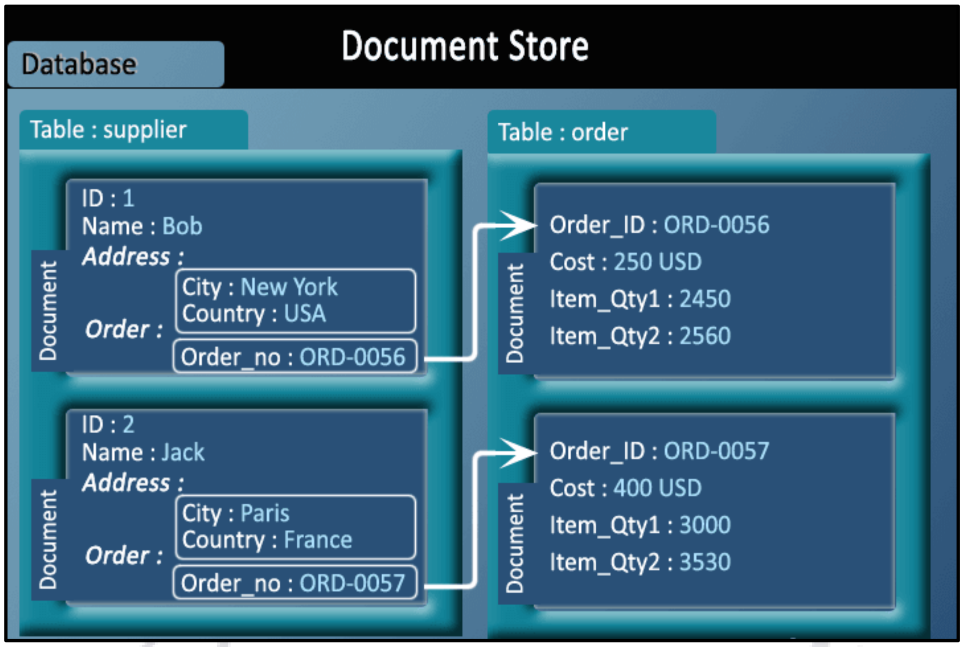


Figure 3.8: Document Store

E. API model

The underlying data model can be considered as a large Hash table (key/value store).

The basic form of API access is

i. get(key) -- Extract the value given a key

ii. put(key, value) -- Create or Update the value given its key

iii. delete(key) -- Remove the key and its associated value

More advance form of API allows to execute user defined function in the server environment

i. Execute (key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... Etc).

ii. MapReduce (keylist, mapfunc, reducefunc) -- Invoke a map/reduce function across a key range.

F. Sharding

1. Sharding means that no one machine has to handle the write workload on the entire dataset, but no one machine can answer queries about the entire dataset.
2. Most NoSQL systems are key-oriented in both their data and query models, and few queries touch the entire dataset anyway.

3. Because the primary access method for data in these systems is key-based, Sharding is typically key-based as well: some function of the key determines the machine on which a key-value pair is stored.
4. Sharding adds complexity, to avoid Sharding we do
   i. **Read Replicas**
      - Many storage systems see more read requests than write requests.
      - A simple solution in these cases is to make copies of the data on multiple machines.
      - All write requests still go to a master node. Read requests go to machines which replicate the data, and are often slightly stale with respect to the data on the write master.

   ii. **Caching**
      - Caching the most popular content in your system often works surprisingly well.
      - Memcached dedicates blocks of memory on multiple servers to cache data from your data store.
      - Memcached clients take advantage of several horizontal scalability tricks to distribute load across
      - Memcached installations on different servers. To add memory to the cache pool, just add another Memcached host.

5. Because Memcached is designed for caching, it does not have as much architectural complexity as the persistent solutions for scaling workloads. Before considering more complicated solutions, think about whether caching can solve your scalability woes.
6. Caching is not solely a temporary Band-Aid: Facebook has Memcached installations in the range of tens of terabytes of memory.
7. Read replicas and caching allow you to scale up your read-heavy workloads. When you start to increase the frequency of writes and updates to your data,

G. When a new node joins the network

1. The joining node announce its presence and its id to some well-known VNs or just broadcast)
2. All the neighbors (left and right side) will adjust the change of key ownership as well as the change of replica memberships. This is typically done synchronously.
3. The joining node starts to bulk copy data from its neighbor in parallel asynchronously.
4. The membership change is asynchronously propagate to the other nodes.

H. When an existing node leaves the network (e.g. crash)

1. The crashed node no longer respond to gossip message so its neighbors knows about it.
2. The neighbor will update the membership changes and copy data asynchronously.

3.3 Using NoSQL to manage big data

A. Client Consistency

Once we have multiple copies of the same data, we need to worry about how to synchronize them such that the client can has a consistent view of the data.

There is a number of client consistency models

1. **Strict Consistency** (one copy serializability): This provides the semantics as if there is only one copy of data. Any update is observed instantaneously.
2. **Read your write consistency**: The allows the client to see his own update immediately (and the client can switch server between requests), but not the updates made by other clients
3. **Session consistency**: Provide the read-your-write consistency only when the client is issuing the request under the same session scope (which is usually bind to the same server)
4. **Monotonic Read Consistency**: This provide the time monotonicity guarantee that the client will only see more updated version of the data in future requests.
5. **Eventual Consistency**: This provides the weakness form of guarantee. The client can see an inconsistent view as the update are in progress. This model works when

concurrent access of the same data is very unlikely, and the client need to wait for some time if he needs to see his previous update.

Depends on which consistency model to provide, 2 mechanisms need to be arranged ...

- How the client request is dispatched to a replica
- How the replicas propagate and apply the updates

There are various models how these 2 aspects can be done, with different tradeoffs.

B. Master Slave (or Single Master) Model

1. Under this model, each data partition has a single master and multiple slaves. In above model, B is the master of keyAB and C, D are the slaves.
2. All update requests has to go to the master where update is applied and then asynchronously propagated to the slaves.
3. Notice that there is a time window of data lost if the master crashes before it propagate its update to any slaves, so some system will wait synchronously for the update to be propagated to at least one slave.
4. Read requests can go to any replicas if the client can tolerate some degree of data staleness.
5. This is where the read workload is distributed among many replicas. If the client cannot tolerate staleness for certain data, it also need to go to the master.
6. When a physical node crashes, the masters of certain partitions will be lost. Usually, the most updated slave will be nominated to become the new master.
7. Master Slave model works very well in general when the application has a high read/write ratio. It also works very well when the update happens evenly in the key range. So it is the predominant model of data replication.
8. There are 2 ways how the master propagate updates to the slave; State transfer and Operation transfer.
9. In State transfer, the master passes its latest state to the slave, which then replace its current state with the latest state.
10. In operation transfer, the master propagate a sequence of operations to the slave which then apply the operations in its local state.

11. The state transfer model is more robust against message lost because as long as a latter more updated message arrives, the replica still be able to advance to the latest state.
12. Even in state transfer mode, we don't want to send the full object for updating other replicas because changes typically happens within a small portion of the object.
13. In will be a waste of network bandwidth if we send the unchanged portion of the object, so we need a mechanism to detect and send just the delta (the portion that has been changed).
14. One common approach is break the object into chunks and compute a hash tree of the object. So the replica can just compare their hash tree to figure out which chunk of the object has been changed and only send those over.

C. Multi-Master (or No Master) Model

1. If there is hot spots in certain key range, and there is intensive write request, the master slave model will be unable to spread the workload evenly. Multi-master model allows updates to happen at any replica
2. If any client can issue any update to any server, how do we synchronize the state's such that we can retain client consistency and also eventually every replica will get to the same state?

D. Gossip (State Transfer Model)

1. In a state transfer model, each replica maintain a vector clock as well as a state version tree where each state is neither > or < among each other (based on vector clock comparison).
2. In other words, the state version tree contains all the conflicting updates.
3. At query time, the client will attach its vector clock and the replica will send back a subset of the state tree which precedes the client's vector clock (this will provide monotonic read consistency).

4. The client will then advance its vector clock by merging all the versions. This means the client is responsible to resolve the conflict of all these versions because when the client sends the update later, its vector clock will precede all these versions.
5. At update, the client will send its vector clock and the replica will check whether the client state precedes any of its existing version, if so, it will throw away the client's update.
6. Replicas also gossip among each other in the background and try to merge their version tree together.

E. Gossip (Operation Transfer Model)

1. In an operation transfer approach, the sequence of applying the operations is very important.
2. At the minimum causal order need to be maintained. Because of the ordering issue, each replica has to defer executing the operation until all the preceding operations has been executed.
3. Therefore replicas save the operation request to a log file and exchange the log among each other and consolidate these operation logs to figure out the right sequence to apply the operations to their local store in an appropriate order.
4. "Causal order" means every replica will apply changes to the "causes" before apply changes to the "effect". "Total order" requires that every replica applies the operation in the same sequence.
5. In this model, each replica keeps a list of vector clock, Vi is the vector clock the replica itself and Vj is the vector clock when replica i receive replica j's gossip message. There is also a V-state that represent the vector clock of the last updated state.
6. When a query is submitted by the client, it will also send along its vector clock which reflect the client's view of the world. The replica will check if it has a view of the state that is later than the client's view.
7. When an update operation is received, the replica will buffer the update operation until it can be applied to the local state. Every submitted operation will be tag with

2 timestamp, V-client indicates the client's view when he is making the update request. V-receive is the replica's view when it receives the submission.

8. This update operation request will be sitting in the queue until the replica has received all the other updates that this one depends on. This condition is reflected in the vector clock Vi when it is larger than V-client.

4. MapReduce and the New Software Stack

CONTENTS

1. Modern data-mining applications, often called "big-data" analysis, require us to manage immense amounts of data quickly. In many of these applications, the data is extremely regular, and there is ample opportunity to exploit parallelism. Important examples are:
    i. The ranking of Web pages by importance, which involves an iterated matrix-vector multiplication where the dimension is many billions.
    ii. Searches in "friends" networks at social-networking sites, which involve graphs with hundreds of millions of nodes and many billions of edges.
2. To deal with applications such as these, a new software stack has evolved.
3. These programming systems are designed to get their parallelism not from a "supercomputer," but from "computing clusters" – large collections of commodity hardware, including conventional processors ("compute nodes") connected by Ethernet cables or inexpensive switches.
4. The software stack begins with a new form of file system, called a "distributed file system," which features much larger units than the disk blocks in a conventional operating system.
5. Distributed file systems also provide replication of data or redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes.
6. On top of these file systems, many different higher-level programming systems have been developed.
7. Central to the new software stack is a programming system called MapReduce.
8. Implementations of MapReduce enable many of the most common calculations on large-scale data to be performed on computing clusters efficiently and in a way that is tolerant of hardware failures during the computation.
9. MapReduce systems are evolving and extending rapidly.

4.1 Distributed File Systems

1. Most computing is done on a single processor, with its main memory, cache, and local disk (a compute node).
2. Applications that called for parallel processing, such as large scientific calculations, were done on special-purpose parallel computers with many processors and specialized hardware.
3. The prevalence of large-scale Web services has caused more and more computing to be done on installations with thousands of compute nodes operating more or less independently.
4. In these installations, the compute nodes are commodity hardware, which greatly reduces the cost compared with special-purpose parallel machines.

A. Physical Organization of Compute Nodes

i. The new parallel-computing architecture, sometimes called cluster computing, is organized as follows. Compute nodes are stored on racks, perhaps 8–64 on a rack.

ii. The nodes on a single rack are connected by a network, typically gigabit Ethernet.

iii. There can be many racks of compute nodes, and racks are connected by another level of network or a switch.

iv. The bandwidth of inter-rack communication is somewhat greater than the intrarack Ethernet, but given the number of pairs of nodes that might need to communicate between racks, this bandwidth may be essential. Figure 4.1 suggests the architecture of a largescale computing system. However, there may be many more racks and many more compute nodes per rack.

v. Some important calculations take minutes or even hours on thousands of compute nodes. If we had to abort and restart the computation every time one component failed, then the computation might never complete successfully.

vi. The solution to this problem takes two forms:

- Files must be stored redundantly. If we did not duplicate the file at several compute nodes, then if one node failed, all its files would be unavailable until the node is replaced. If we did not back up the files at all, and the disk crashes, the files would be lost forever.
- Computations must be divided into tasks, such that if any one task fails to execute to completion, it can be restarted without affecting other tasks.

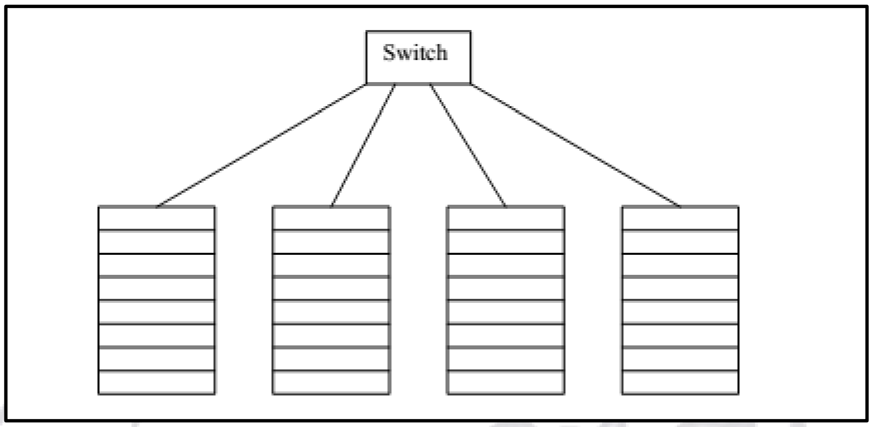


Figure 4.1: Rack of compute nodes

B. Large-Scale File-System Organization

i. To exploit cluster computing, files must look and behave somewhat differently from the conventional file systems found on single computers.

ii. This new file system, often called a distributed file system or DFS (although this term has had other meanings in the past), is typically used as follows.

iii. There are several distributed file systems of the type we have described that are used in practice. Among these:

- The Google File System (GFS), the original of the class.
- Hadoop Distributed File System (HDFS), an open-source DFS used with Hadoop, an implementation of map-reduce and distributed by the Apache Software Foundation.
- CloudStore, an open-source DFS originally developed by Kosmix.

iv. Files can be enormous, possibly a terabyte in size. If you have only small files, there is no point using a DFS for them.

v. Files are rarely updated. Rather, they are read as data for some calculation, and possibly additional data is appended to files from time to time. For example, an airline reservation system would not be suitable for a DFS, even if the data were very large, because the data is changed so frequently.

vi. Files are divided into chunks, which are typically 64 megabytes in size.

vii. Chunks are replicated, perhaps three times, at three different compute nodes.

viii. Moreover, the nodes holding copies of one chunk should be located on different racks, so we don't lose all copies due to a rack failure. Normally, both the chunk size and the degree of replication can be decided by the user.

ix. To find the chunks of a file, there is another small file called the master node or name node for that file.

x. The master node is itself replicated, and a directory for the file system as a whole knows where to find its copies.

xi. The directory itself can be replicated, and all participants using the DFS know where the directory copies are.

4.2 MapReduce

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

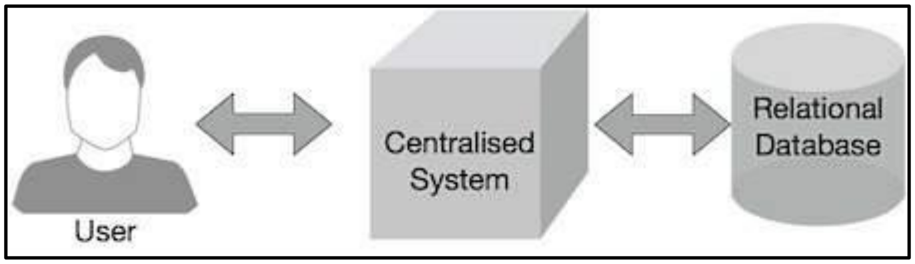


Figure 4.2: MapReduce

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

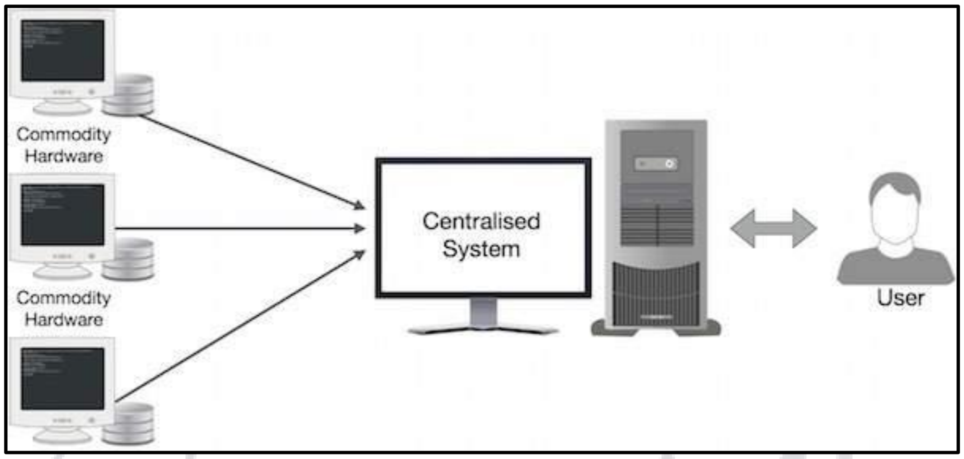


Figure 4.3: Physical structure

6. A MapReduce computation executes as follows:
   - Some number of Map tasks each are given one or more chunks from a distributed file system. These Map tasks turn the chunk into a sequence of key-value pairs. The way key-value pairs are produced from the input data is determined by the code written by the user for the Map function.
   - The key-value pairs from each Map task are collected by a master controller and sorted by key. The keys are divided among all the Reduce tasks, so all key-value pairs with the same key wind up at the same Reduce task.
   - The Reduce tasks work on one key at a time, and combine all the values associated with that key in some way. The manner of combination of values is determined by the code written by the user for the Reduce function.

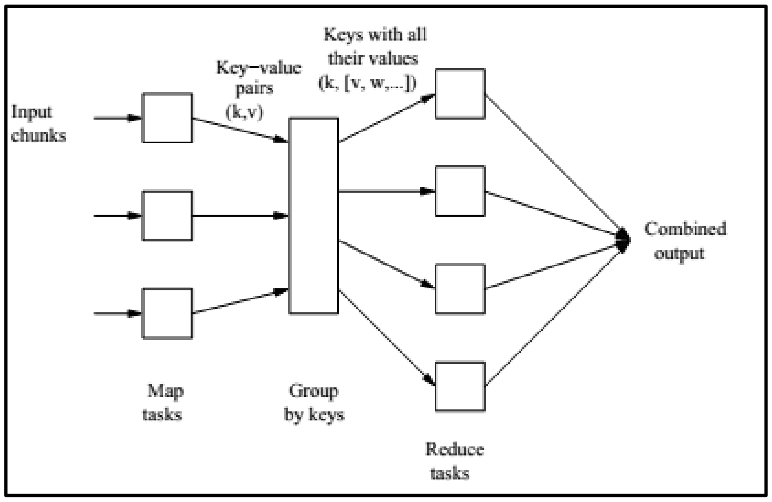


Figure 4.4: Schematic MapReduce Computation

A. The Map Task

i. We view input files for a Map task as consisting of elements, which can be any type: a tuple or a document, for example.

ii. A chunk is a collection of elements, and no element is stored across two chunks.

iii. Technically, all inputs to Map tasks and outputs from Reduce tasks are of the key-value-pair form, but normally the keys of input elements are not relevant and we shall tend to ignore them.

iv. Insisting on this form for inputs and outputs is motivated by the desire to allow composition of several MapReduce processes.

v. The Map function takes an input element as its argument and produces zero or more key-value pairs.

vi. The types of keys and values are each arbitrary.

vii. Further, keys are not “keys” in the usual sense; they do not have to be unique.

viii. Rather a Map task can produce several key-value pairs with the same key, even from the same element.

ix. **Example 1**: A MapReduce computation with what has become the standard example application: counting the number of occurrences for each word in a collection of documents. In this example, the input file is a repository of documents, and each document is an element. The Map function for this example uses keys that are of type String (the words) and values that are integers. The Map task reads a document and breaks it into its sequence of words $w_1, w_2, \ldots, w_n$. It then emits a sequence of key-value pairs where the value is always 1. That is, the output of the Map task for this document is the sequence of key-value pairs:

$$(w_1, 1), (w_2, 1), \ldots, (w_n, 1)$$

A single Map task will typically process many documents – all the documents in one or more chunks. Thus, its output will be more than the sequence for the one document suggested above. If a word *w* appears *m* times among all the documents assigned to that process, then there will be m key-value pairs (w, 1) among its output. An option, is to combine these m pairs into a single pair (w, m), but we can only do that because, the Reduce tasks apply an associative and commutative operation, addition, to the values.

B. Grouping by Key

i. As the Map tasks have all completed successfully, the key-value pairs are grouped by key, and the values associated with each key are formed into a list of values.
ii. The grouping is performed by the system, regardless of what the Map and Reduce tasks do.
iii. The master controller process knows how many Reduce tasks there will be, say r such tasks.
iv. The user typically tells the MapReduce system what r should be.
v. Then the master controller picks a hash function that applies to keys and produces a bucket number from 0 to r − 1.
vi. Each key that is output by a Map task is hashed and its key-value pair is put in one of r local files. Each file is destined for one of the Reduce tasks.1.
vii. To perform the grouping by key and distribution to the Reduce tasks, the master controller merges the files from each Map task that are destined for a

particular Reduce task and feeds the merged file to that process as a sequence of key-list-of-value pairs.

viii. That is, for each key k, the input to the Reduce task that handles key k is a pair of the form (k, [v1, v2, . . . , vn]), where (k, v1), (k, v2), . . . , (k, vn) are all the key-value pairs with key k coming from all the Map tasks.

C. The Reduce Task

i. The Reduce function's argument is a pair consisting of a key and its list of associated values.

ii. The output of the Reduce function is a sequence of zero or more key-value pairs.

iii. These key-value pairs can be of a type different from those sent from Map tasks to Reduce tasks, but often they are the same type.

iv. We shall refer to the application of the Reduce function to a single key and its associated list of values as a reducer. A Reduce task receives one or more keys and their associated value lists.

v. That is, a Reduce task executes one or more reducers. The outputs from all the Reduce tasks are merged into a single file.

vi. Reducers may be partitioned among a smaller number of Reduce tasks is by hashing the keys and associating each

vii. Reduce task with one of the buckets of the hash function.

viii. With reference to Example 1 on Page 77, The Reduce function simply adds up all the values. The output of a reducer consists of the word and the sum. Thus, the output of all the Reduce tasks is a sequence of (w, m) pairs, where w is a word that appears at least once among all the input documents and m is the total number of occurrences of w among all those documents.

D. Combiners

i. A Reduce function is associative and commutative. That is, the values to be combined can be combined in any order, with the same result.

ii. The addition performed in Example 1 is an example of an associative and commutative operation. It doesn't matter how we group a list of numbers v_1, v_2, . . . , v_n; the sum will be the same.

iii. When the Reduce function is associative and commutative, we can push some of what the reducers do to the Map tasks

iv. These key-value pairs would thus be replaced by one pair with key w and value equal to the sum of all the 1's in all those pairs.

v. That is, the pairs with key w generated by a single Map task would be replaced by a pair (w, m), where m is the number of times that w appears among the documents handled by this Map task.

E. Details of MapReduce task

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

i. The Map task takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs).

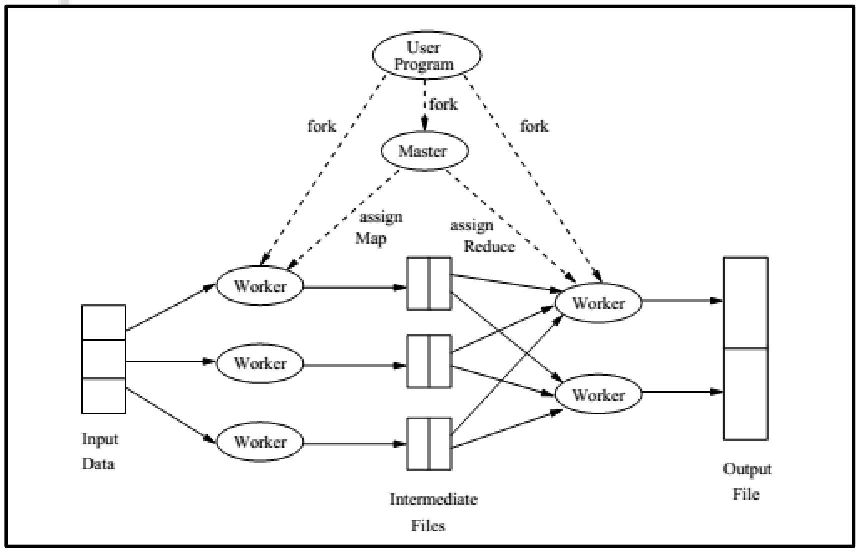


Figure 4.5: Overview of the execution of a MapReduce program

ii. The Reduce task takes the output from the Map as an input and combines those data tuples (key-value pairs) into a smaller set of tuples.

iii. The reduce task is always performed after the map job.

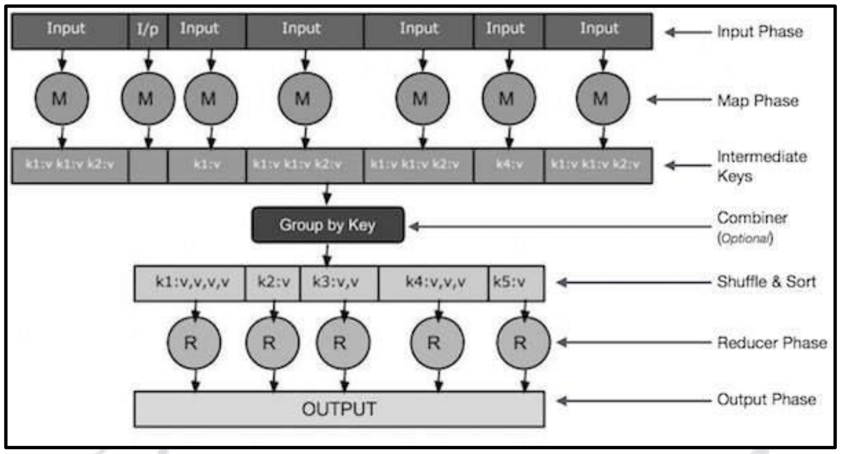


Figure 4.6: Reduce job

- **Input Phase** − Here we have a Record Reader that translates each record in an input file and sends the parsed data to the mapper in the form of key-value pairs.
- **Map** − Map is a user-defined function, which takes a series of key-value pairs and processes each one of them to generate zero or more key-value pairs.
- **Intermediate Keys** − they key-value pairs generated by the mapper are known as intermediate keys.
- **Combiner** − A combiner is a type of local Reducer that groups similar data from the map phase into identifiable sets. It takes the intermediate keys from the mapper as input and applies a user-defined code to aggregate the values in a small scope of one mapper. It is not a part of the main MapReduce algorithm; it is optional.
- **Shuffle and Sort** − The Reducer task starts with the Shuffle and Sort step. It downloads the grouped key-value pairs onto the local machine, where the Reducer is running. The individual key-value pairs are sorted by key into a larger data list. The data list groups the equivalent keys together so that their values can be iterated easily in the Reducer task.
- **Reducer** − The Reducer takes the grouped key-value paired data as input and runs a Reducer function on each one of them. Here, the data can be aggregated,

filtered, and combined in a number of ways, and it requires a wide range of processing. Once the execution is over, it gives zero or more key-value pairs to the final step.

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

iv. The MapReduce phase

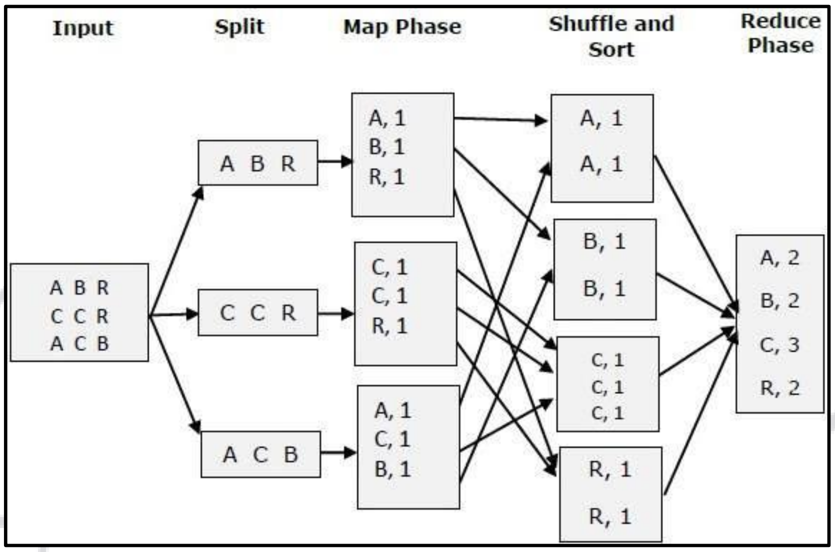


Figure 4.7: The MapReduce Phase

F. MapReduce-Example

Twitter receives around 500 million tweets per day, which is nearly 3000 tweets per second. The following illustration shows how Tweeter manages its tweets with the help of MapReduce.

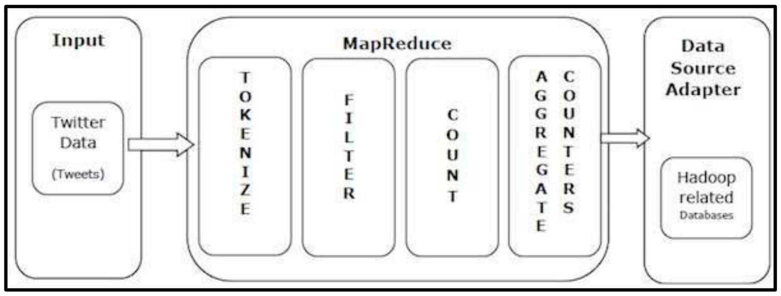


Figure4.8: Example

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

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

iii. **Count** – Generates a token counter per word.

iv. **Aggregate Counters** – Prepares an aggregate of similar counter values into small manageable units.

G. MapReduce – Algorithm

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

i. The map task is done by means of Mapper Class

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

ii. The reduce task is done by means of Reducer Class.

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

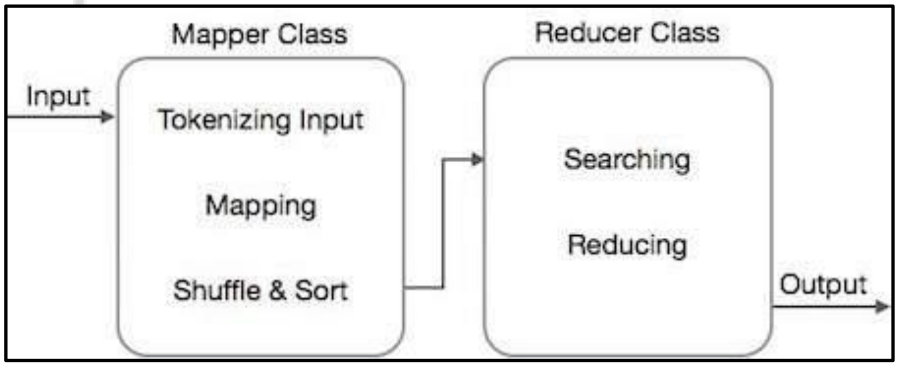


Figure 4.9: The MapReduce Class

H. Coping With Node Failures

i. The worst thing that can happen is that the compute node at which the Master is executing fails. In this case, the entire MapReduce job must be restarted.

ii. But only this one node can bring the entire process down; other failures will be managed by the Master, and the MapReduce job will complete eventually.

iii. Suppose the compute node at which a Map worker resides fails. This failure will be detected by the Master, because it periodically pings the Worker processes.

iv. All the Map tasks that were assigned to this Worker will have to be redone, even if they had completed. The reason for redoing completed Map asks is that their output destined for the Reduce tasks resides at that compute node, and is now unavailable to the Reduce tasks.

v. The Master sets the status of each of these Map tasks to idle and will schedule them on a Worker when one becomes available. The

vi. Master must also inform each Reduce task that the location of its input from that Map task has changed. Dealing with a failure at the node of a Reduce worker is simpler.

vii. The Master simply sets the status of its currently executing Reduce tasks to idle. These will be rescheduled on another reduce worker later.

4.3 Algorithms Using MapReduce

A. Matrix Vector Multiplication with MapReduce

i. Matrix-vector and matrix-matrix calculations fit nicely into the MapReduce style of computing.

ii. Suppose we have a pxq matrix M, whose element in row i and column j will be denoted $m_{ij}$and a qxr matrix N whose element in row j and column k is donated by n_{jk}then the product P = MN will be pxr matrix P whose element in row i and column k will be donated by p_{ik}, where $P(i,k)$= $m_{ij} * n_{jk}$.

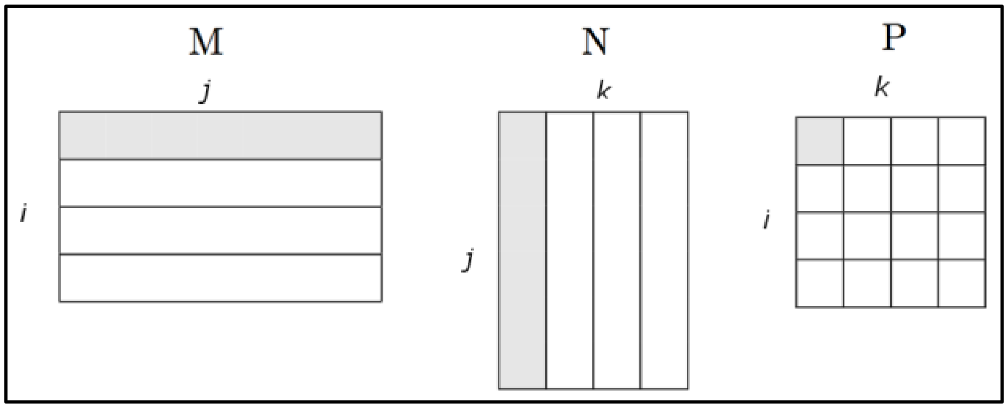
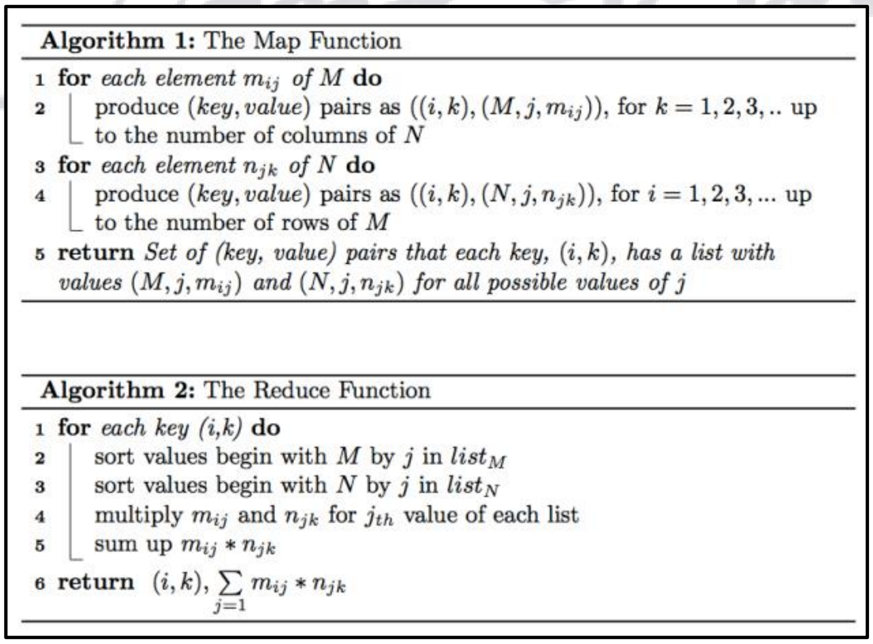


Figure 4.10: Matrix Multiplication

iii. Write Map and Reduce functions to process input files. Map function will produce $key, value$ pairs from the input data as it is described in Algorithm 1.

iv. Reduce function uses the output of the Map function and performs the calculations and produces $key, value$ pairs as described in Algorithm 2. All outputs are written to HDFS.



Algorithm 1: The Map Function

```
1 for each element $m_{ij}$ of $M$ do
2     produce $(key, value)$ pairs as $((i,k),(M,j,m_{ij}))$, for $k = 1,2,3,..$ up
      to the number of columns of $N$
3 for each element $n_{jk}$ of $N$ do
4     produce $(key, value)$ pairs as $((i,k),(N,j,n_{jk}))$, for $i = 1,2,3,...$ up
      to the number of rows of $M$
5 return Set of (key, value) pairs that each key, $(i,k)$, has a list with
  values $(M,j,m_{ij})$ and $(N,j,n_{jk})$ for all possible values of $j$
```

Algorithm 2: The Reduce Function

```
1 for each key (i,k) do
2     sort values begin with $M$ by $j$ in $list_M$
3     sort values begin with $N$ by $j$ in $list_N$
4     multiply $m_{ij}$ and $n_{jk}$ for $j_{th}$ value of each list
5     sum up $m_{ij} * n_{jk}$
6 return $(i,k), \sum_{j=1} m_{ij} * n_{jk}$
```

Figure 4.11: Algorithm

B. Relational- Algebra Operations

i. There are a number of operations on large-scale data that are used in database queries.

ii. Many traditional database applications involve retrieval of small mounts f data, even though the database itself may be large.

iii. For example, a query may ask for the bank balance of one particular account. Such queries are not useful applications of MapReduce.

iv. There are several standard operations on relations, often referred to as relational algebra, that are used to implement queries.

v. The queries themselves usually are written in SQL. The relational-algebra operations we shall discuss are:

- **Selection**: Apply a condition C to each tuple in the relation and produce as output only those tuples that satisfy C. The result of this selection is denoted σC(R).
- **Projection**: For some subset S of the attributes of the relation, produce from each tuple only the components for the attributes in S. The result of this projection is denoted πS(R).
- **Union, Intersection, and Difference**: These well-known set operations apply to the sets of tuples in two relations that have the same schema. There are also bag (multiset) versions of the operations in SQL, with somewhat unintuitive definitions, but we shall not go into the bag versions of these operations here.
- **Natural Join**: Given two relations, compare each pair of tuples, one from each relation. If the tuples agree on all the attributes that are common to the two schemas, then produce a tuple that has components for each of the attributes in either schema and agrees with the two tuples on each attribute.
- **Grouping and Aggregation**: Given a relation R, partition its tuples according to their values in one set of attributes G, called the grouping attributes. Then, for each group, aggregate the values in certain other attributes.

- **Example:** Let's try to find the paths of length two in the Web, using the relation Links as shown in the table below. That is, we want to find the triples of URL's (u, v, w) such that there is a link from u to v and a link from v to w. We essentially want to take the natural join of Links with itself, but we first need to imagine that it is two relations, with different schemas, so we can describe the desired connection as a natural join. Thus, imagine that there are two copies of Links, namely L1 (U1, U2) and L2 (U2, U3). Now, if we compute L1 ⋈ L2, we shall have exactly what we want. That is, for each tuple t1 of L1 (i.e., each tuple of Links) and each tuple t2 of L2 (another tuple of Links, possibly even the same tuple), see if their U2 components are the same. Note that these components are the second component of t1 and the first component of t2. If these two components agree, then produce a tuple for the result, with schema (U1, U2, U3). This tuple consists of the first component of t1, the second of t1 (which must equal the first component of t2), and the second component of t2. We may not want the entire path of length two, but only want the pairs (u,w) of URL's such that there is at least one path from u to w of length two. If so, we can project out the middle components by computing πU1, U3(L1 ⋈ L2).'

From	To
url1	url12
url1	url13
url2	url13
url2	url14

- **Example 2**: Imagine that a social-networking site has a relation

  Friends (User, Friend)

  This relation has tuples that are pairs (a,b) such that b is a friend of a. The site might want to develop statistics about the number of friends members have.

Their first step would be to compute a count of the number of friends of each user. This operation can be done by grouping and aggregation, specifically

$\Upsilon_{User,COUNT(Friend)}$ (Friends)

This operation groups all the tuples by the value in their first component, so there is one group for each user. Then, for each group the count of the number of friends of that user is made. The result will be one tuple for each group, and a typical tuple would look like (Sally, 300), if user “Sally” has 300 friends.

C. Computing Selections by MapReduce

i. Selections really do not need the full power of MapReduce.

ii. They can be done most conveniently in the map portion alone, although they could also be done in the reduce portion alone.

iii. Here is a MapReduce implementation of selection σC(R).

- **The Map Function**: For each tuple t in R, test if it satisfies C. If so, produce the key-value pair (t,t). That is, both the key and value are t.
- **The Reduce Function**: The Reduce function is the identity. It simply passes each key-value pair to the output.

D. Computing Projections by MapReduce

i. Projection is performed similarly to selection, because projection may cause the same tuple to appear several times, the Reduce function must eliminate duplicates.

ii. We may compute πS(R) as follows.

- **The Map Function**: For each tuple t in R, construct a tuple t’ by eliminating from t those components whose attributes are not in S. Output the key-value pair (t’, t’).
- **The Reduce Function**: For each key t’ produced by any of the Map tasks, there will be one or more key-value pairs (t’, t’). The Reduce function turns (t’, [t’, t’, . . . , t’]) into (t’, t’), so it produces exactly one pair (t’, t’) for this key

E. Union, Intersection, and Difference by MapReduce

i. First, consider the union of two relations. Suppose relations R and S have the same schema.

ii. Map tasks will be assigned chunks from either R or S; it doesn't matter which. The Map tasks don't really do anything except pass their input tuples as key-value pairs to the Reduce tasks.

iii. The latter need only eliminate duplicates as for projection.

- **The Map Function**: Turn each input tuple t into a key-value pair (t, t).
- **The Reduce Function**: Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

iv. To compute the intersection, we can use the same Map function.

v. However, the Reduce function must produce a tuple only if both relations have the tuple. If the key t has a list of two values [t, t] associated with it, then the Reduce task for t should produce (t, t).

vi. However, if the value-list associated with key t is just [t], then one of R and S is missing t, so we don't want to produce a tuple for the intersection.

- **The Map Function**: Turn each tuple t into a key-value pair (t, t).
- **The Reduce Function:** If key t has value list [t, t], then produce (t, t). Otherwise, produce nothing.

vii. The Difference R − S requires a bit more thought. The only way a tuple t can appear in the output is if it is in R but not in S.

viii. The Map function can pass tuples from R and S through, but must inform the Reduce function whether the tuple came from R or S.

ix. The relation as the value associated with the key t. Here is a specification for the two functions.

- **The Map Function**: For a tuple t in R, produce key-value pair (t,R), and for a tuple t in S, produce key-value pair (t,S). Note that the intent is that the value is the name of R or S (or better, a single bit indicating whether the relation is R or S), not the entire relation.

- **The Reduce Function:** For each key t, if the associated value list is [R], then produce (t,t). Otherwise, produce nothing.

F. Computing Natural Join by MapReduce

i. The idea behind implementing natural join via MapReduce can be seen if we look at the specific case of joining R(A,B) with S(B,C).
ii. Find tuples that agree on their B components, that is the second component from tuples of R and the first component of tuples of S.
iii. Use the B-value of tuples from either relation as the key.
iv. The value will be the other component and the name of the relation, so the Reduce function can know where each tuple came from.
   - **The Map Function**: For each tuple (a,b) of R, produce the key-value pair (b, (R,a)). For each tuple (b,c) of S, produce the key-value pair (b, (S,c)).
   - **The Reduce Function**: Each key value b will be associated with a list of pairs that are either of the form (R,a) or (S,c). Construct all pairs consisting of one with first component R and the other with first component S, say (R,a) and (S,c). The output from this key and value list is a sequence of key-value pairs. The key is irrelevant. Each value is one of the triples (a,b,c) such that (R,a) and (S,c) are on the input list of values.

G. Grouping and Aggregation by MapReduce

i. As with the join, we shall discuss the minimal example of grouping and aggregation, where there is one grouping attribute and one aggregation.
ii. Let R(A,B,C) be a relation to which we apply the operator γA,θ(B)(R).
iii. Map will perform the grouping, while Reduce does the aggregation.
   - The Map Function: For each tuple (a,b,c) produce the key-value pair (a,b).
   - The Reduce Function: Each key a represents a group. Apply the aggregation operator θ to the list [b1, b2, . . . , bn] of B-values associated with key a. The output is the pair (a, x), where x is the result of applying θ to the list. For example, if θ is SUM, then x = b1 + b2 + · · · + bn, and if θ is MAX, then x is the largest of b1, b2, . . . , bn.

H. Matrix Multiplication

i. If M is a matrix with element m_{ij} in row i and column j, and N is a matrix with element n jk in row j and column k, then the product P = MN is the matrix P with element p_ik in row i and column k, where

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

ii. It is required that the number of columns of M equals the number of rows of N, so the sum over j makes sense.

- **The Map Function**: For each matrix element m_{ij}, produce the key value pair j, (M, i, m_{ij}) . Likewise, for each matrix element n_jk, produce the key value pair j, (N, k, n_jk). Note that M and N in the values are not the matrices themselves. Rather they are names of the matrices or (as we mentioned for the similar Map function used for natural join) better, a bit indicating whether the element comes from M or N.
- **The Reduce Function**: For each key j, examine its list of associated values. For each value that comes from M, say (M, i, m_{ij}), and each value that comes from N, say (N, k, n_jk), produce a key-value pair with key equal to (i, k) and value equal to the product of these elements, $m_{ij}n_j$k.

iii. Grouping and aggregation by another MapReduce operation.

- **The Map Function**: This function is just the identity. That is, for every input element with key (i, k) and value v, produce exactly this key-value pair.
- **The Reduce Function**: For each key (i, k), produce the sum of the list of values associated with this key. The result is a pair (i, k), v, where v is the value of the element in row i and column k of the matrix P = M N.

I. Matrix Multiplication with One MapReduce Step

i. To use only a single MapReduce pass to perform matrix multiplication P = M N. 5 It is possible to do so if we put more work into the two functions. Start by using the Map function to create the sets of matrix elements that are needed to compute each element of the answer P. Notice that an element of M or N contributes to many elements of the result, so one input element will be turned into many key-value pairs. The keys will be pairs (i, k), where i is a row of M and k is a column of N. Here is a synopsis of the Map and Reduce functions.

- **The Map Function**: For each element m_{ij} of M, produce all the key-value pairs (i, k), (M, j, m_{ij}) for k = 1, 2, . . ., up to the number of columns of N. Similarly, for each element njk of N, produce all the key-value pairs (i, k), (N, j, n_jk) for i = 1, 2, . . ., up to the number of rows of M. As before, M and N are really bits to tell which of the two matrices a value comes from.
- **The Reduce Function**: Each key (i, k) will have an associated list with all the values (M, j, m_{ij}) and (N, j, n_jk), for all possible values of j. The Reduce function needs to connect the two values on the list that have the same value of j, for each j. An easy way to do this step is to sort by j the values that begin with M and sort by j the values that begin with N, in separate lists. The jth values on each list must have their third components, m_{ij} and n_jk extracted and multiplied. Then, these products are summed and the result is paired with (i, k) in the output of the Reduce function.

5. Finding Similar Items

CONTENTS

1. A fundamental data-mining problem is to examine data for “similar” items.
2. As example we would be considering at a collection of Web pages and finding near-duplicate pages.
3. These pages could be plagiarisms, or they could be mirrors that have almost the same content but differ in information about the host and about other mirrors.
4. We first do phrasing the problem of similarity as one of finding sets with a relatively large intersection.
5. Then we resolve how the problem of finding textually similar documents can be turned into such a set problem by the technique known as “shingling.”
6. Then, we introduce a technique called “minhashing,” which compresses large sets in such a way that we can still deduce the similarity of the underlying sets from their compressed versions.
7. Another important problem that arises when we search for similar items of any kind is that there may be far too many pairs of items to test each pair for their degree of similarity, even if computing the similarity of any one pair can be made very easy.
8. That concern motivates a technique called “locality-sensitive hashing,” for focusing our search on pairs that are most likely to be similar.

5.1 Applications of Near-Neighbor Search

1. Initially on a particular notion of “similarity”: the similarity of sets by looking at the relative size of their intersection. This notion of similarity is called “Jaccard similarity,”
2. Examine some of the uses of finding similar sets.
3. These include finding textually similar documents and collaborative filtering by finding similar customers and similar products. In order to turn the problem of textual similarity of documents into one of set intersection, we use a technique called “shingling,”

i. **Jaccard Similarity of Sets:** The Jaccard similarity of sets S and T is |S ∩ T |/|S ∪ T |, that is, the ratio of the size of the intersection of S and T to the size of their union. We shall denote the Jaccard similarity of S and T by SIM(S, T).

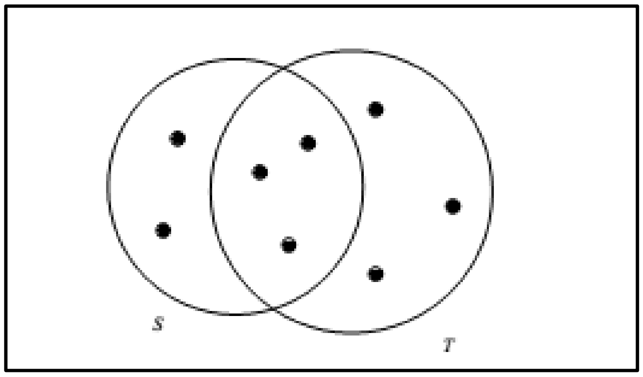


Figure 5.1: Two sets with Jaccard similarity 3/8

ii. **Similarity of Documents:** An important class of problems that Jaccard similarity addresses well is that of finding textually similar documents in a large corpus such as the Web or a collection of news articles. We should understand that the aspect of similarity we are looking at here is character-level similarity, not "similar meaning," which requires us to examine the words in the documents and their uses. That problem is also interesting. Textual similarity also has important uses. Many of these involve finding duplicates or near duplicates. First, observe that testing whether two documents are exact duplicates is easy; just compare the two documents character-by-character, and if they ever differ then they are not the same. However, in many applications, the documents are not identical, yet they share large portions of their text. Examples are

- **Plagiarism:** Finding plagiarized documents tests our ability to find textual similarity. The plagiarizer may extract only some parts of a document for his own. He may alter a few words and may alter the order in which sentences of the original appear. Yet the resulting document may still contain 50% or more of the original. No simple process of comparing documents character by character will detect a sophisticated plagiarism.

- **Mirror Pages**: It is common for important or popular Web sites to be duplicated at a number of hosts, in order to share the load. The pages of these mirror sites will be quite similar, but are rarely identical. For instance, they might each contain information associated with their particular host, and they might each have links to the other mirror sites but not to themselves. A related phenomenon is the appropriation of pages from one class to another. These pages might include class notes, assignments, and lecture slides. Similar pages might change the name of the course, year, and make small changes from year to year. It is important to be able to detect similar pages of these kinds, because search engines produce better results if they avoid showing two pages that are nearly identical within the first page of results.
- **Articles from the Same Source:** It is common for one reporter to write a news article that gets distributed, say through the Associated Press, to many newspapers, which then publish the article on their Web sites. Each newspaper changes the article somewhat. They may cut out paragraphs, or even add material of their own. They most likely will surround the article by their own logo, ads, and links to other articles at their site. However, the core of each newspaper's page will be the original article. News aggregators, such as Google News, try to find all versions of such an article, in order to show only one, and that task requires finding when two Web pages are textually similar, although not identical.

iii. **Collaborative Filtering as a Similar-Sets Problem**: Another class of applications where similarity of sets is very important is called collaborative filtering, a process whereby we recommend to users items that were liked by other users who have exhibited similar tastes. We shall investigate collaborative filtering in detail in Section 9.3, but for the moment let us see some common examples.

- **On-Line Purchases:** Amazon.com has millions of customers and sells millions of items. Its database records which items have been bought by which customers. We can say two customers are similar if their sets of purchased items have a high Jaccard similarity. Likewise, two items that have sets of purchasers with high Jaccard similarity will be deemed similar. Note that, while we might expect mirror sites to have Jaccard similarity above 90%, it is unlikely that any two customers have Jaccard similarity that high (unless they have purchased only one item). Even a Jaccard similarity like 20% might be unusual enough to identify customers with similar tastes. The same observation holds for items; Jaccard similarities need not be very high to be significant.

- **Movie Ratings**: Netflix records which movies each of its customers rented, and also the ratings assigned to those movies by the customers. We can see movies as similar if they were rented or rated highly by many of the same customers, and see customers as similar if they rented or rated highly many of the same movies. The same observations that we made for Amazon above apply in this situation: similarities need not be high to be significant, and clustering movies by genre will make things easier.
- When our data consists of ratings rather than binary decisions (bought/did not buy or liked/disliked), we cannot rely simply on sets as representations of customers or items. Some options are:
  a. Ignore low-rated customer/movie pairs; that is, treat these events as if the customer never watched the movie.
  b. When comparing customers, imagine two set elements for each movie,“liked” and “hated.”
  c. If a customer rated a movie highly, put the “liked” for that movie in the customer’s set. If they gave a low rating to a movie, put “hated” for that movie in their set.
  d. Then, we can look for high Jaccard similarity among these sets. We can do a similar trick when comparing movies.
  e. If ratings are 1-to-5-stars, put a movie in a customer’s set n times if they rated the movie n-stars.

f. Then, use Jaccard similarity for bags when measuring the similarity of customers. The Jaccard similarity for bags.

Example 1

Compute the Jaccard similarity of each pair of the following sets: {1, 2, 3, 4, 5}, {1, 6, 7}, {2, 4, 6, 8} .

Answer:

Recall the general formula for the Jaccard similarity of two sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

For the three combinations of pairs above, we have

J({1, 2, 3, 4, 5}, {1, 6, 7}) = 1/7

J({1, 2, 3, 4, 5}, {2, 4, 6, 8}) = 2/7

J({1, 6, 7}, {2, 4, 6, 8}) = 1/6

5.2 Distance Measures

The closer sets are, the higher the Jaccard similarity. Rather, 1 minus the Jaccard similarity is a distance measure, as we shall see; it is called the Jaccard distance.

1. **Definition of a Distance Measure**
   i. Suppose we have a set of points, called a space. A distance measure on this space is a function d(x, y) that takes two points in the space as arguments and produces a real number, and satisfies the following axioms:
      - $d(x, y) \geq 0$ (no negative distances).
      - $d(x, y) = 0$ if and only if $x = y$ (distances are positive, except for the
      - distance from a point to itself).
      - $d(x, y) = d(y, x)$ (distance is symmetric).
2. $d(x, y) \leq d(x, z) + d(z, y)$ (the triangle inequality).

3. **Euclidean Distances**
   i. The most familiar distance measure is the one we normally think of as “distance.”
   ii. An n-dimensional Euclidean space is one where points are vectors of n real numbers.
   iii. The conventional distance measure in this space, which we shall refer to as the L2-norm, is defined:

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

   iv. That is, we square the distance in each dimension, sum the squares, and take the positive square root.
4. **Jaccard Distance**
   i. As mentioned at the beginning of the section, we define the Jaccard distance of sets by d(x, y) = 1 − SIM(x, y). That is, the Jaccard distance is 1 minus the ratio of the sizes of the intersection and union of sets x and y. We must verify that this function is a distance measure.
      - d(x, y) is nonnegative because the size of the intersection cannot exceed the size of the union.
      - d(x, y) = 0 if x = y, because x ∪ x = x ∩ x = x. However, if x 6= y, then the size of x ∩ y is strictly less than the size of x ∪ y, so d(x, y) is strictly positive.
      - d(x, y) = d(y, x) because both union and intersection are symmetric; i.e., x ∪ y = y ∪ x and x ∩ y = y ∩ x.
      - For the triangle inequality, recall from Section 3.3.3 that SIM(x, y) is the probability a random minhash function maps x and y to the same value. Thus, the Jaccard distance d(x, y) is the probability that a random minhash function does not send x and y to the same value. We can therefore translate the condition d(x, y) ≤ d(x, z) + d(z, y) to the statement that if h is a random minhash function, then the probability

that h(x) 6= h(y) is no greater than the sum of the probability that h(x) 6= h(z) and the probability that h(z) 6= h(y). However, this statement is true because whenever h(x) 6= h(y), at least one of h(x) and h(y) must be different from h(z). They could not both be h(z), because then h(x) and h(y) would be the same.

5. **Cosine Distance**
   i. The cosine distance makes sense in spaces that have dimensions, including Euclidean spaces and discrete versions of Euclidean spaces, such as spaces where points are vectors with integer components or boolean (0 or 1) components.
   ii. In such a space, points may be thought of as directions. We do not distinguish between a vector and a multiple of that vector.
   iii. Then the cosine distance between two points is the angle that the vectors to those points make.
   iv. This angle will be in the range 0 to 180 degrees, regardless of how many dimensions the space has.
6. **Edit Distance**
   i. This distance makes sense when points are strings. The distance between two strings $x = x_1x_2 \cdots x_n$ and $y = y1y2 \cdots y_m$ is the smallest number of insertions and deletions of single characters that will convert x to y.
   ii. Another way to define and calculate the edit distance d(x, y) is to compute a longest common subsequence (LCS) of x and y. An LCS of x and y is a string that is constructed by deleting positions from x and y, and that is as long as any string that can be constructed that way.
   iii. The edit distance d(x, y) can be calculated as the length of x plus the length of y minus twice the length of their LCS.

Example 3

The edit distance between the strings x = abcde and y = acfdeg is 3.

To convert x to y:

1. Delete b.
2. Insert f after c.
3. Insert g after e.

No sequence of fewer than three insertions and/or deletions will convert x to y.

Thus, d(x, y) = 3.

Another way to define and calculate the edit distance d(x, y) is to compute a longest common subsequence (LCS) of x and y. An LCS of x and y is a string that is constructed by deleting positions from x and y, and that is as long as any string that can be constructed that way. The edit distance d(x, y) can be calculated as the length of x plus the length of y minus twice the length of their LCS.

Example 4:

The strings x = abcde and y = acfdeg from Example 3 have a unique LCS, which is acde. We can be sure it is the longest possible, because it contains every symbol appearing in both x and y. fortunately, these common symbols appear in the same order in both strings, so we are able to use them all in an LCS. Note that the length of x is 5, the length of y is 6, and the length of their LCS is 4. The edit distance is thus $5 + 6 - 2 \times 4 = 3$, which agrees with the direct calculation

7. Hamming Distance

i. Given a space of vectors, we define the Hamming distance between two vectors to be the number of components in which they differ.

ii. It should be obvious that Hamming distance is a distance measure. Clearly the Hamming distance cannot be negative, and if it is zero, then the vectors are identical.

iii. The distance does not depend on which of two vectors we consider first.

iv. The triangle inequality should also be evident. If x and z differ in m components, and z and y differ in n components, then x and y cannot differ in more than m+n components.

v. Most commonly, hamming distance is used when the vectors are boolean; they consist of 0's and 1's only.

vi. In principle, the vectors can have components from any set.

6. Mining Data Streams

Contents

1. The algorithms for processing streams each involve summarization of the stream in some way.
2. Start by considering how to make a useful sample of a stream and how to filter a stream to eliminate most of the "undesirable" elements.
3. Then show how to estimate the number of different elements in a stream using much less storage than would be required if we listed all the elements we have seen.
4. Another approach to summarizing a stream is to look at only a fixed-length "window" consisting of the last n elements for some (typically large) n.
5. We then query the window as if it were a relation in a database. If there are many streams and/or n is large, we may not be able to store the entire window for every stream, so we need to summarize even the windows.
6. Address the fundamental problem of maintaining an approximate count on the number of 1's in the window of a bit stream, while using much less space than would be needed to store the entire window itself.
7. This technique generalizes to approximating various kinds of sums.

6.1 The Stream Data Model

i. As data arrives in various streams, it's important to stream the data as the data which arrives cannot be stored immediately and the probability of losing the data is more.
ii. To mine the data we need to create the sample of stream data and to filter the stream to eliminate most of the "undesirable" data elements.
iii. To summarize the large stream of data, consider the fixed length window consisting of last n elements.
iv. The data stream model is considered for the management of data, along with the stream queries and issues in stream processing.

A. A Data-Stream-Management System

i. To a database-management system, we can view a stream processor as a kind of data-management system, the high-level organization of which is shown below.

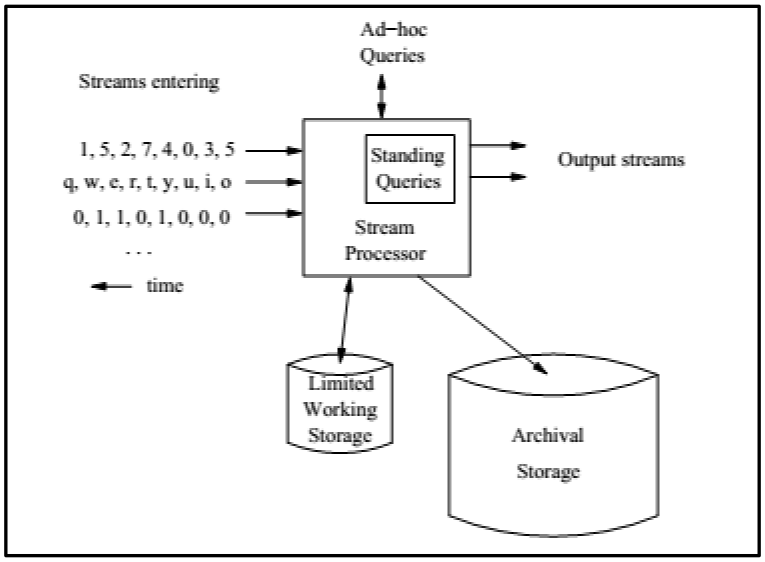


Figure 6.1: A data-stream-management system

ii. Any number of streams can enter the system.

iii. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform.

iv. The fact that the rate of arrival of stream elements is not under the control of the system distinguishes stream processing from the processing of data that goes on within a database-management system.

v. The latter system controls the rate at which data is read from the disk, and therefore never has to worry about data getting lost as it attempts to execute queries.

B. Examples of Stream Sources

i. **Sensor Data**: Imagine a temperature sensor bobbing about in the ocean, sending back to a base station a reading of the surface temperature each hour. The data produced by this sensor is a stream of real numbers. It is not a very interesting stream, since the data rate is so low. It would not stress modern technology, and the entire stream could be kept in main memory, essentially forever.

ii. **Image Data:** Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second. London is said to have six million such cameras, each producing a stream.

iii. **Internet and Web Traffic:** A switching node in the middle of the Internet receives streams of IP packets from many inputs and routes them to its outputs. Normally, the job of the switch is to transmit data and not to retain it or query it. But there is a tendency to put more capability into the switch, e.g., the ability to detect denial-of-service attacks or the ability to reroute packets based on information about congestion in the network.

C. Stream Queries

i. There are two ways that queries get asked about streams.

ii. A place within the processor where standing queries are stored. As shown below.

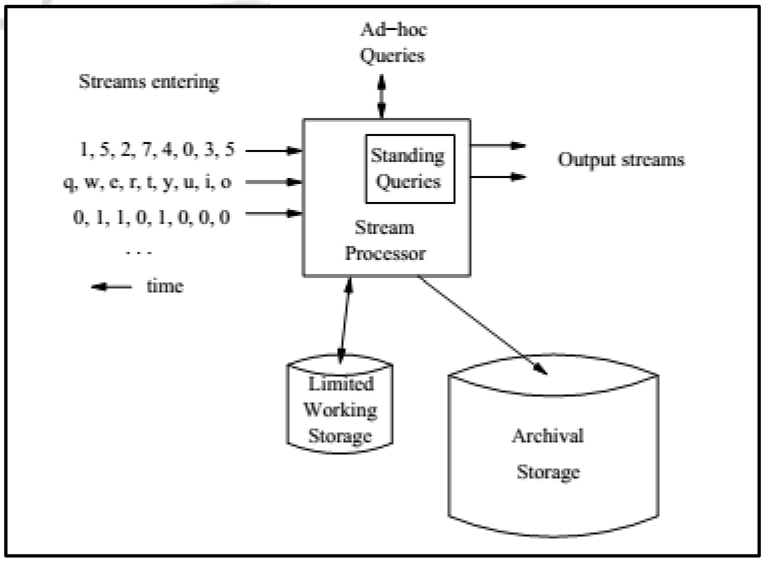


Figure 6.2: Streaming Queries

iii. These queries are, in a sense, permanently executing, and produce outputs at appropriate times.

D. Issues in Stream Processing

i. First, streams often deliver elements very rapidly.
ii. We must process elements in real time, or we lose the opportunity to process them at all, without accessing the archival storage.
iii. Thus, it often is important that the stream-processing algorithm is executed in main memory, without access to secondary storage or with only rare accesses to secondary storage.
iv. Thus, many problems about streaming data would be easy to solve if we had enough memory, but become rather hard and require the invention of new techniques in order to execute them at a realistic rate on a machine of realistic size.

6.2 Sampling Data in a Stream

It refers to as managing and extracting the relevant data from the reliable sample of the stream.

A. Obtaining a Representative Sample

i. Like many queries about the statistics of typical users, cannot be answered by taking a sample of each user's search queries.
ii. Thus, we must strive to pick 1/10th of the users, and take all their searches for the sample, while taking none of the searches from other users.
iii. If we can store a list of all users, and whether or not they are in the sample, then we could do the following. Each time a search query arrives in the stream, we look up the user to see whether or not they are in the sample.
iv. If so, we add this search query to the sample, and if not, then not. However, if we have no record of ever having seen this user before, then we generate a random integer between 0 and 9.
v. If the number is 0, we add this user to our list with value "in," and if the number is other than 0, we add the user with the value "out."

vi. That method works as long as we can afford to keep the list of all users and their in/out decision in main memory, because there isn't time to go to disk for every search that arrives.

vii. By using a hash function, one can avoid keeping the list of users. That is, we hash each user name to one of ten buckets, 0 through 9.

viii. If the user hashes to bucket 0, then accept this search query for the sample, and if not, then not.

B. The General Sampling Problem

i. The running example is typical of the following general problem. Our stream consists of tuples with n components.

ii. A subset of the components are the key components, on which the selection of the sample will be based. In our running example, there are three components – user, query, and time – of which only user is in the key. However, we could also take a sample of queries by making query be the key, or even take a sample of user-query pairs by making both those components form the key.

iii. To take a sample of size a/b, we hash the key value for each tuple to b buckets, and accept the tuple for the sample if the hash value is less than a.

iv. If the key consists of more than one component, the hash function needs to combine the values for those components to make a single hash-value.

v. The result will be a sample consisting of all tuples with certain key values. The selected key values will be approximately a/b of all the key values appearing in the stream.

C. Varying the Sample Size

i. If we have a budget for how many tuples from the stream can be stored as the sample, then the fraction of key values must vary, lowering as time goes on.

ii. In order to assure that at all times, the sample consists of all tuples from a subset of the key values, we choose a hash function h from key values to a very large number of values 0, 1, . . . , B−1.

iii. We maintain a threshold t, which initially can be the largest bucket number, B − 1.
iv. At all times, the sample consists of those tuples whose key K satisfies h(K) ≤ t. New tuples from the stream are added to the sample if and only if they satisfy the same condition.
v. If the number of stored tuples of the sample exceeds the allotted space, we lower t to t−1 and remove from the sample all those tuples whose key K hashes to t.
vi. For efficiency, we can lower t by more than 1, and remove the tuples with several of the highest hash values, whenever we need to throw some key values out of the sample.
vii. Further efficiency is obtained by maintaining an index on the hash value, so we can find all those tuples whose keys hash to a particular value quickly.

6.3 Filtering Streams

i. Another common process on streams is selection, or filtering.
ii. We want to accept those tuples in the stream that meet a criterion.
iii. Accepted tuples are passed to another process as a stream, while other tuples are dropped.
iv. If the selection criterion is a property of the tuple that can be calculated (e.g., the first component is less than 10), then the selection is easy to do.

A. The Bloom Filter

i. A Bloom filter consists of:
   - ➢ An array of n bits, initially all 0’s.
   - ➢ A collection of hash functions h1, h2, . . . , hk. Each hash function maps “key” values to n buckets, corresponding to the n bits of the bit-array.
   - ➢ A set S of m key values.

ii. The purpose of the Bloom filter is to allow through all stream elements whose keys are in S, while rejecting most of the stream elements whose keys are not in S.

iii. To initialize the bit array, begin with all bits 0. Take each key value in S and hash it using each of the k hash functions. Set to 1 each bit that is hi(K) for some hash function hi and some key value K in S.

iv. To test a key K that arrives in the stream, check that all of are 1's in the bit-array. If all are 1's, then let the stream element through. If one or more of these bits are 0, then K could not be in S, so reject the stream element.

$$h_1(K), h_2(K), \ldots, h_k(K)$$

B. Analysis of Bloom Filtering

i. If a key value is in S, then the element will surely pass through the Bloom filter.

ii. However, if the key value is not in S, it might still pass.

iii. We need to understand how to calculate the probability of a false positive, as a function of n, the bit-array length, m the number of members of S, and k, the number of hash functions.

6.4 Counting Distinct Elements in a Stream

As filtering and sampling the distinctive data plays an important role for removal of data from the stream of data. Various distinctive criteria's are used in this section.

A. The Count-Distinct Problem

i. Stream elements are chosen from some universal set. We would like to know how many different elements have appeared in the stream, counting either from the beginning of the stream or from some known time in the past.

ii. **Consider a Web site gathering statistics** on how many unique users it has seen in each given month. The universal set is the set of logins for that site, and a stream element is generated each time someone logs in. This measure is appropriate for a site like Amazon, where the typical user logs in with their unique login name.

iii. **Solution:** The obvious way to solve the problem is to keep in main memory a list of all the elements seen so far in the stream. Keep them in an efficient search structure such as a hash table or search tree, so one can quickly add new elements and check whether or not the element that just arrived on the stream was already seen. As long as the number of distinct elements is not too great, this structure can fit in main memory and there is little problem obtaining an exact answer to the question how many distinct elements appear in the stream.

B. The Flajolet-Martin Algorithm

i. The idea behind the Flajolet-Martin Algorithm is that the more different elements we see in the stream, the more different hash-values we shall see.

ii. We get more different hash-values, it becomes more likely that one of these values will be "unusual." The particular unusual property we shall exploit is that the value ends in many 0's, although many other options exist.

iii. Whenever we apply a hash function h to a stream element a, the bit string h(a) will end in some number of 0's, possibly none. Call this number the tail length for a and h. Let R be the maximum tail length of any a seen so far in the stream. Then we shall use estimate 2R for the number of distinct elements seen in the stream.

iv. **Conclusion:**

- If m is much larger than 2r, then the probability that we shall find a tail of length at least r approaches 1.
- If m is much less than 2r, then the probability of finding a tail length at least r approaches 0.

C. Combining Estimates

i. There is a trap regarding the strategy for combining the estimates of m, the number of distinct elements that we obtain by using many different hash functions.

ii. Our first assumption would be that if we take the average of the values 2R that we get from each hash function, we shall get a value that approaches the true m, the more hash functions we use.

iii. However, that is not the case, and the reason has to do with the influence an overestimate has on the average.

iv. Another way to combine estimates is to take the median of all estimates. The median is not affected by the occasional outsized value of 2R, so the worry described above for the average should not carry over to the median. Unfortunately, the median suffers from another defect: it is always a power of 2.

v. Thus, no matter how many hash functions we use, should the correct value of m be between two powers of 2, say 400, then it will be impossible to obtain a close estimate.

D. Space Requirements

i. The only thing we need to keep in main memory is one integer per hash function; this integer records the largest tail length seen so far for that hash function and any stream element.

ii. If we are processing only one stream, we could use millions of hash functions, which is far more than we need to get a close estimate.

iii. Only if we are trying to process many streams at the same time would main memory constrain the number of hash functions we could associate with any one stream.

iv. In practice, the time it takes to compute hash values for each stream element would be the more significant limitation on the number of hash functions we use.

6.5 Counting Ones in a Window:

To solve the problem of the window of the length N on a binary stream, to count how many 1's are there in the last k bits? We focus on the situation where we cannot effort to store the entire window. This section deals with the binary case of summing up numbers.

A. The Cost of Exact Counts

i. To begin, suppose we want to be able to count exactly the number of 1's in the last k bits for any k ≤ N.

ii. Then we claim it is necessary to store all N bits of the window, as any representation that used fewer than N bits could not work. In proof, suppose we have a representation that uses fewer than N bits to represent the N bits in the window.

iii. Since there are 2N sequences of N bits, but fewer than 2N representations, there must be two different bit strings w and x that have the same representation.

iv. Since w 6= x, they must differ in at least one bit. Let the last k − 1 bits of w and x agree, but let them differ on the kth bit from the right end.

B. The Datar-Gionis-Indyk-Motwani Algorithm

i. The simplest case of an algorithm called DGIM.

ii. This version of the algorithm uses O(log2 N) bits to represent a window of N bits, and allows us to estimate the number of 1's in the window with an error of no more than 50%.

iii. An improvement of the method that limits the error to any fraction ϙ > 0, and still uses only O(log2 N) bits (although with a constant factor that grows as ϙ shrinks).

iv. To begin, each bit of the stream has a timestamp, the position in which it arrives. The first bit has timestamp 1, the second has timestamp 2, and so on.

v. Since we only need to distinguish positions within the window of length N, we shall represent timestamps modulo N, so they can be represented by log2 N bits.

vi. If we also store the total number of bits ever seen in the stream (i.e., the most recent timestamp) modulo N, then we can determine from a timestamp modulo N where in the current window the bit with that timestamp is.

vii. We divide the window into buckets,5 consisting of:

- The timestamp of its right (most recent) end.
- The number of 1's in the bucket. This number must be a power of 2, and we refer to the number of 1's as the size of the bucket.

viii. There are six rules that must be followed when representing a stream by buckets.

- The right end of a bucket is always a position with a 1.
- Every position with a 1 is in some bucket.
- No position is in more than one bucket.
- There are one or two buckets of any given size, up to some maximum size.
- All sizes must be a power of 2.
- Buckets cannot decrease in size as we move to the left (back in time).

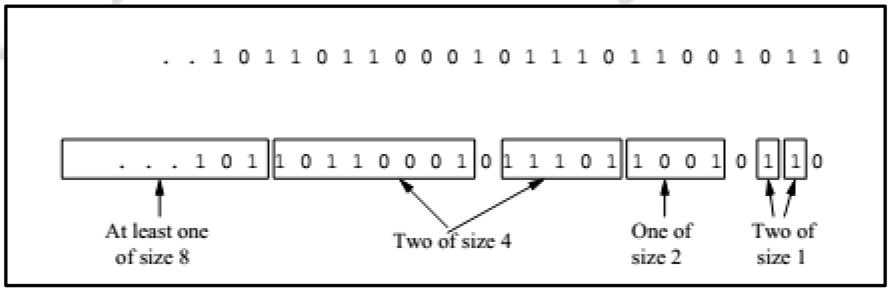


Figure 6.3: A bit-stream divided into buckets following the DGIM rules

C. Query Answering in the DGIM Algorithm

i. To calculate how many 1's there are in the last k bits of the window, for some $1 \le k \le N$.

ii. Find the bucket b with the earliest timestamp that includes at least some of the k most recent bits.

iii. Estimate the number of 1's to be the sum of the sizes of all the buckets to the right (more recent) than bucket b, plus half the size of b itself.

D. Decaying Windows.

i. A sliding window held a certain tail of the stream, either the most recent N elements for fixed N, or all the elements that arrived after some time in the past.

ii. Sometimes we do not want to make a sharp distinction between recent elements and those in the distant past, but want to weight the recent elements more heavily.

iii. Consider "exponentially decaying windows," and an application where they are quite useful: finding the most common "recent" elements.

Example 1

If $n \epsilon$ N, then $1+3+5+7+\cdots+(2n-1) = n^2$.

Proof. We will prove this with mathematical induction.

a. Observe that if $n = 1$, this statement is $1 = 1^2$, which is obviously true.

b. We must now prove S_k ⍰ S_{k+1} for any $k \geq 1$. That is, we must show that if $1+3+5+7+\cdots+(2k-1) = k^2$,

then $1+3+5+7+\cdots+(2(k+1)-1) = (k+1)^2$.

We use direct proof. Suppose $1+3+5+7+\cdots+(2k-1) = k^2$. Then

$$\begin{aligned}
1+3+5+7+\cdots\cdots\cdots+(2(k+1)-1) &= \\
1+3+5+7+\cdots+(2k-1)+(2(k+1)-1) &= \\
(1+3+5+7+\cdots+(2k-1))+(2(k+1)-1) &= \\
k^2+(2(k+1)-1) &= k^2+2k+1 \\
&= (k+1)^2.
\end{aligned}$$

Thus $1+3+5+7+\cdots+(2(k+1)-1) = (k+1)^2$.

This proves that $S_k = S_{k+1}$.

It follows by induction that $1+3+5+7+\cdots+(2n-1) = n2$ for every $n \epsilon$ N.

7. Link Analysis

CONTENTS

i. When search engines were designed the page ranking algorithms are the base line for implementation.
ii. PageRank was established as the essential technique for search engine, and spammers invented ways to manipulate the PageRank on a webpage called as link spam.
iii. The link analysis plays an important role in the analysis of web pages and linking.

7.1 Page Rank

A. Page Rank Definition

i. PageRank is a function that assigns a real number to each page in the Web (or at least to that portion of the Web that has been crawled and its links discovered).
ii. The intent is that the higher the PageRank of a page, the more "important" it is.
iii. There is not one fixed algorithm for assignment of PageRank, and in fact variations on the basic idea can alter the relative PageRank of any two pages.
iv. We begin by defining the basic, idealized PageRank, and follow it by modifications that are necessary for dealing with some real-world problems concerning the structure of the Web.
v. Think of the Web as a directed graph, where pages are the nodes, and there is an arc from page p1 to page p2 if there are one or more links from p1 to p2.
vi. Figure below is an example of a tiny version of the Web, where there are only four pages.

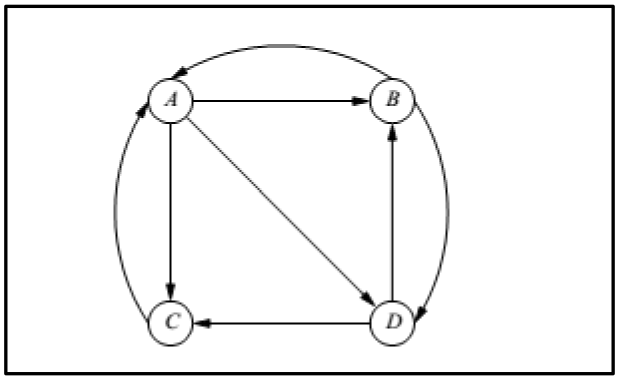


Figure 7.1: A hypothetical example of the Web

vii. Page A has links to each of the other three pages; page B has links to A and D only; page C has a link only to A, and page D has links to B and C only.

viii. Suppose a random surfer starts at page A in Fig. 5.1. There are links to B, C, and D, so this surfer will next be at each of those pages with probability 1/3, and has zero probability of being at A. A random surfer at B has, at the next step, probability 1/2 of being at A, 1/2 of being at D, and 0 of being at B or C.

ix. The transition matrix for the above hypothetical web is

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

x. In this matrix, the order of the pages is the natural one, A, B, C, and D. Thus, the first column expresses the fact, already discussed, that a surfer at A has a 1/3 probability of next being at each of the other pages. The second column expresses the fact that a surfer at B has a 1/2 probability of being next at A and the same of being at D. The third column says a surfer at C is certain to be at A next. The last column says a surfer at D has a 1/2 probability of being next at B and the same at C.

B. Structure of Web

i. The structure below shows a large strongly connected component (SCC), but there were several other portions that were almost as large.

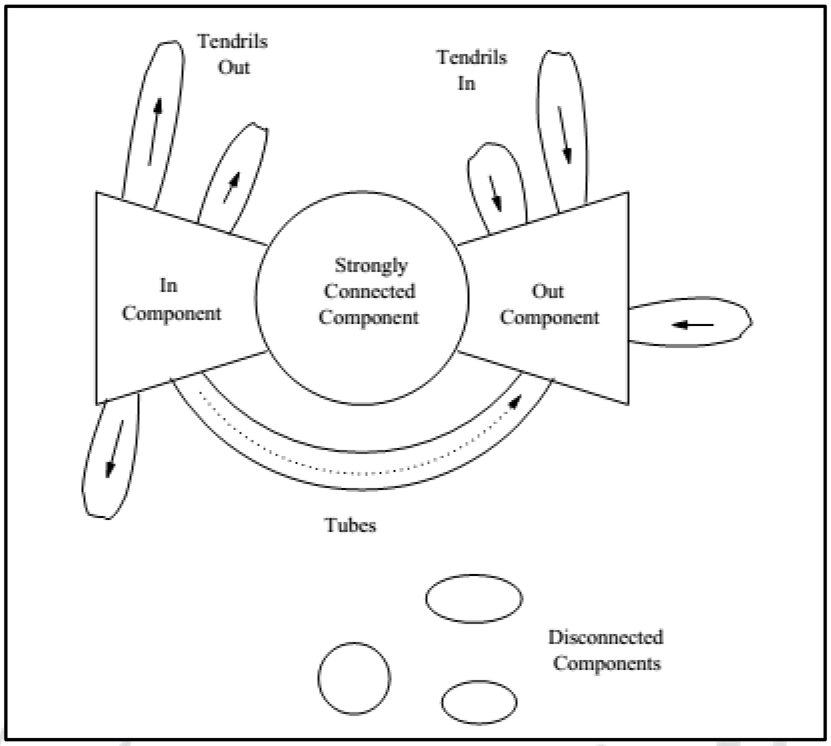


Figure 7.2: The "bowtie" picture of the Web

- The in-component, consisting of pages that could reach the SCC by following links, but were not reachable from the SCC.
- The out-component, consisting of pages reachable from the SCC but unable to reach the SCC.
- Tendrils, which are of two types. Some tendrils consist of pages reachable from the in-component but not able to reach the in-component. The other tendrils can reach the out-component, but are not reachable from the out-component.

ii. there were small numbers of pages found either in

  - Tubes, which are pages reachable from the in-component and able to reach the out-component, but unable to reach the SCC or be reached from the SCC.
  - Isolated components that are unreachable from the large components (the SCC, in- and out-components) and unable to reach those components.

iii. PageRank is usually modified to prevent such anomalies. There are really two problems we need to avoid.
iv. First is the dead end, a page that has no links out. Surfers reaching such a page disappear, and the result is that in the limit no page that can reach a dead end can have any PageRank at all.
v. The second problem is groups of pages that all have outlinks but they never link to any other pages. These structures are called spider traps.
vi. Both these problems are solved by a method called “taxation,” where we assume a random surfer has a finite probability of leaving the Web at any step, and new surfers are started at each page.

C. Avoiding Dead Ends

i. A page with no link out is called a dead end.
ii. If we allow dead ends, the transition matrix of the Web is no longer stochastic, since some of the columns will sum to 0 rather than 1.
iii. A matrix whose column sums are at most 1 is called substochastic.
iv. If we compute Miv for increasing powers of a substochastic matrix M, then some or all of the components of the vector go to 0.
v. That is, importance “drains out” of the Web, and we get no information about the relative importance of pages.

D. Using PageRank in a Search Engine

i. Order to be considered for the ranking at all, a page has to have at least one of the search terms in the query.
ii. The weighting of properties is such that unless all the search terms are present, a page has very little chance of being in the top ten that are normally shown first to the user.
iii. The qualified pages, a score is computed for each, and an important component of this score is the PageRank of the page. Other components include the presence or absence of search terms in prominent places, such as headers or the links to the page itself.

E. Efficient Computation of PageRank

i. To compute the PageRank for a large graph representing the Web, we have to perform a matrix–vector multiplication on the order of 50 times, until the vector is close to unchanged at one iteration.

ii. To a first approximation, the MapReduce method is suitable. However, we must deal with two issues.

- The transition matrix of the Web M is very sparse.
- Representing it by all its elements is highly inefficient. Rather, we want to represent the matrix by its nonzero elements.
- We may not be using MapReduce, or for efficiency reasons we may wish to use a combiner with the Map tasks to reduce the amount of data that must be passed from Map tasks to Reduce tasks.
- The striping approach is not sufficient to avoid heavy use of disk (thrashing).
- **Representing Transition Matrices:** The transition matrix is very sparse, since the average Web page has about 10 out-links. If, say, we are analyzing a graph of ten billion pages, then only one in a billion entries is not 0. The proper way to represent any sparse matrix is to list the locations of the nonzero entries and their values. If we use 4-byte integers for coordinates of an element and an 8-byte double-precision number for the value, then we need 16 bytes per nonzero entry. That is, the space needed is linear in the number of nonzero entries, rather than quadratic in the side of the matrix.

F. PageRank Iteration Using MapReduce

i. One iteration of the PageRank algorithm involves taking an estimated PageRank vector v and computing the next estimate v′ by

$$\mathbf{v}' = \beta M\mathbf{v} + (1-\beta)\mathbf{e}/n$$

ii. β is a constant slightly less than 1, e is a vector of all 1's, and n is the number of nodes in the graph that transition matrix M represents.
iii. If n is small enough that each Map task can store the full vector v in main memory and also have room in main memory for the result vector v′, then there is little more here than a matrix–vector multiplication.
iv. The additional steps are to multiply each component of M_v by constant β and to add (1 − β)/n to each component.

G. Use of Combiners to Consolidate the Result Vector

The combiner method is not adequate because of the following reasons

i. We might wish to add terms for v_i ′, the ith component of the result vector v, at the Map tasks. This improvement is the same as using a combiner, since the Reduce function simply adds terms with a common key.
ii. For a MapReduce implementation of matrix–vector multiplication, the key is the value of i for which a term $m_{ij}v_j$ is intended.
iii. Using MapReduce will not be adequate, but rather executing the iteration step at a single machine or a collection of machines.
iv. An alternative strategy is based on partitioning the matrix into k^2 blocks, while the vectors are still partitioned into k stripes. A picture, showing the division for k = 4, is in Figure 7.4. Note that we have not shown the multiplication of the matrix by β or the addition of (1 − β) e/n, because these steps are straightforward, regardless of the strategy we use.
v. In this method, we use k^2 Map tasks. Each task gets one square of the matrix M, say M_{ij}, and one stripe of the vector v, which must be v_j. Notice that each stripe of the vector is sent to k different Map tasks; v_j is sent to the task handling M_{ij} for each of the k possible values of i.
vi. Thus, v is transmitted over the network k times. However, each piece of the matrix is sent only once. Since the size of the matrix can be expected to be several times the size of the vector, the transmission cost is not too much greater than the minimum possible.

vii. And because we are doing considerable combining at the Map tasks, we save as data is passed from the Map tasks to the Reduce tasks. The advantage of this approach is that we can keep both the j^{th} stripe of v and the i^{th} stripe of v′ in main memory as we process M_{ij}. Note that all terms generated from M_{ij} and v_j contribute to v_i ′ and no other stripe of v′.

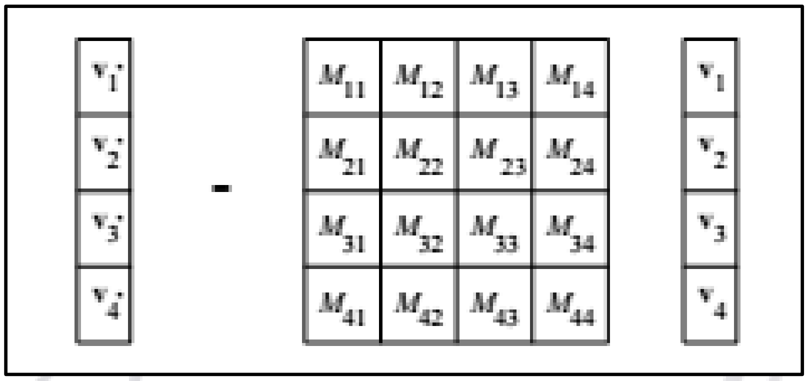


Figure 7.4: Partitioning a matrix into square blocks

H. Representing Blocks of the Transition Matrix

i. For each block, we need data about all those columns that have at least one nonzero entry within the block.

ii. If k, the number of stripes in each dimension, is large, then most columns will have nothing in most blocks of its stripe.

iii. For a given block, we not only have to list those rows that have a nonzero entry for that column, but we must repeat the out-degree for the node represented by the column.

iv. Consequently, it is possible that the out-degree will be repeated as many times as the out-degree itself.

v. That observation bounds from above the space needed to store the blocks of a stripe at twice the space needed to store the stripe as a whole.

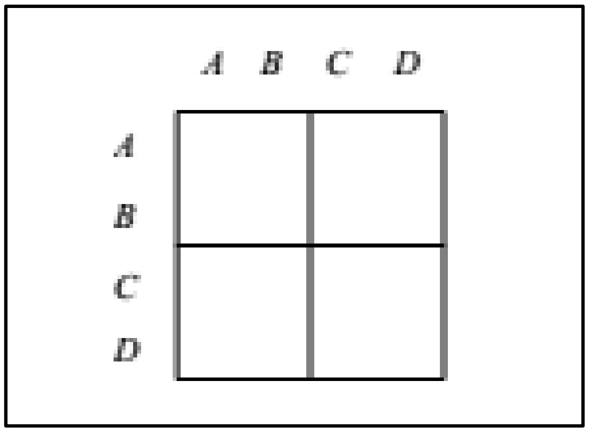


Figure 7.5: A four-node graph is divided into four 2-by-2 blocks

7.2 Topic sensitive Page Rank

i. The mechanism for enforcing this weighting is to alter the way random surfers behave, having them prefer to land on a page that is known to cover the chosen topic.

ii. The topic-sensitive idea can also be applied to negate the effects of a new kind of spam, called "'link spam," that has developed to try to avoid the PageRank algorithm.

A. Motivation for Topic-Sensitive Page Rank

- Different people have different interests, and sometimes distinct interests are expressed using the same term in a query.
- The canonical example is the search query jaguar, which might refer to the animal, the automobile, a version of the MAC operating system, or even an ancient game console.
- If a search engine can deduce that the user is interested in automobiles, for example, then it can do a better job of returning relevant pages to the user.

B. Biased Random Walks

- To create a topic-sensitive PageRank for sports, we can arrange that the random surfers are introduced only to a random sports page, rather than to a random page of any kind.

- The consequence of this choice is that random surfers are likely to be at an identified sports page, or a page reachable along a short path from one of these known sports pages.
- The pages linked to by sports pages are themselves likely to be about sports.
- The pages they link to are also likely to be about sports, although the probability of being about sports surely decreases as the distance from an identified sports page increases.

C. Using Topic-Sensitive PageRank

To integrate and incorporate the topic-sensitive PageRank into search engine:

- Decide on the topics for which we shall create specialized PageRank vectors.
- Pick a teleport set for each of these topics, and use that set to compute the topic-sensitive PageRank vector for that topic.
- Find a way of determining the topic or set of topics that are most relevant for a particular search query.
- Use the PageRank vectors for that topic or topics in the ordering of the responses to the search query.

7.3 Link Spam

i. When it became apparent that PageRank and other techniques used by Google made term spam ineffective, spammers turned to methods designed to fool the PageRank algorithm into overvaluing certain pages.
ii. The techniques for artificially increasing the PageRank of a page are collectively called link spam.
iii. To understand how spammers create link spam, and then see several methods for decreasing the effectiveness of these spamming techniques, including Trust Rank and measurement of spam mass, the architecture of the Spam Farm is practiced.

A. Architecture of a Spam Farm

A collection of pages whose purpose is to increase the PageRank of a certain page or pages is called a spam farm.

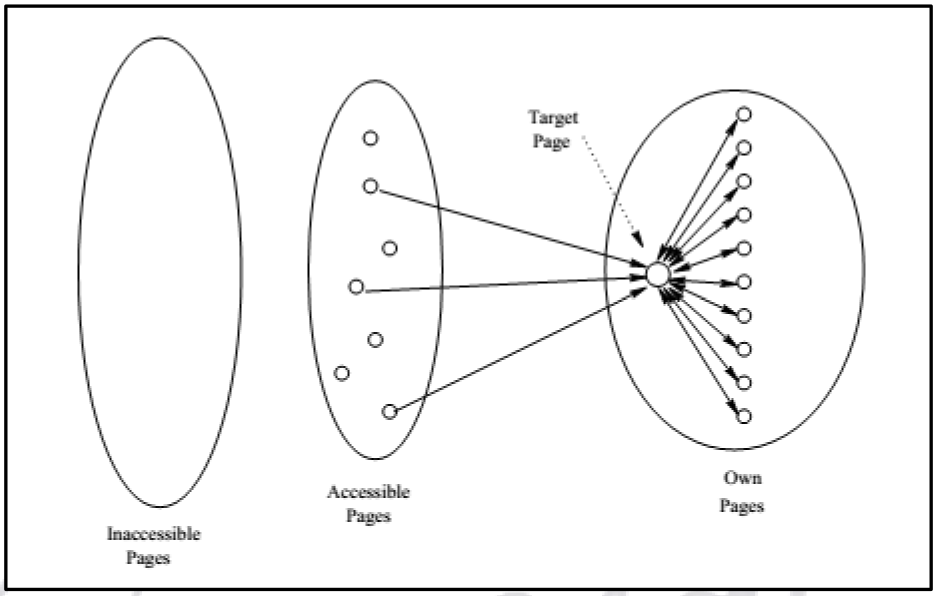


Figure 7.6: The Web from the point of view of the link spammer

The figure above shows the simplest form of spam farm. From the point of view of the spammer, the Web is divided into three parts:

- Inaccessible pages: the pages that the spammer cannot affect. Most of the Web is in this part.
- Accessible pages: those pages that, while they are not controlled by the spammer, can be affected by the spammer.
- Own pages: the pages that the spammer owns and controls.

B. Analysis of a Spam Farm

- PageRank is computed using a taxation parameter β, typically around 0.85.
- That is, β is the fraction of a page's PageRank that gets distributed to its successors at the next round.
- There be n pages on the Web in total, and let some of them be a spam farm of the form, with a target page t and m supporting pages.
- Let x be the amount of PageRank contributed by the accessible pages.

- That is, x is the sum, over all accessible pages p with a link to t, of the PageRank of p time's β, divided by the number of successors of p.
- Finally, let y be the unknown PageRank of t.

C. Combating Link Spam

- It has become essential for search engines to detect and eliminate link spam, just as it was necessary in the previous decade to eliminate term spam.
- There are approaches to link spam. One is to look for structures such as the spam farm as shown below, where one page links to a very large number of pages, each of which links back to it.
- Search engines surely search for such structures and eliminate those pages from their index.
- That causes spammers to develop different structures that have essentially the same effect of capturing PageRank for a target page or pages.
- There is essentially no end to variations of above figure, so this war between the spammers and the search engines will likely go on for a long time.

D. Trust Rank

Trust Rank is topic-sensitive PageRank, where the "topic" is a set of pages believed to be trustworthy (not spam). The theory is that while a spam page might easily be made to link to a trustworthy page, it is unlikely that a trustworthy page would link to a spam page. The approaches are

- Let humans examine a set of pages and decide which of them are trustworthy. For example, we might pick the pages of highest PageRank to examine, on the theory that, while link spam can raise a page's rank from the bottom to the middle of the pack, it is essentially impossible to give a spam page a PageRank near the top of the list.
- Pick a domain whose membership is controlled, on the assumption that it is hard for a spammer to get their pages into these domains. For example,

we could pick the .edu domain, since university pages are unlikely to be spam farms. We could likewise pick .mil, or .gov.

7.4 Hubs and Authorities

i. An idea called “hubs and authorities’ was proposed shortly after PageRank was first implemented.
ii. The algorithm for computing hubs and authorities bears some resemblance to the computation of PageRank, since it also deals with the iterative computation of a fixed point involving repeated matrix–vector multiplication.
iii. There are also significant differences between the two ideas, and neither can substitute for the other.
iv. This hubs-and-authorities algorithm, sometimes called HITS (hyperlink induced topic search), was originally intended not as a preprocessing step before handling search queries, as PageRank is, but as a step to be done along with the processing of a search query, to rank only the responses to that query.

A. The Intuition Behind HITS

- While PageRank assumes a one-dimensional notion of importance for pages, HITS views important pages as having two flavors of importance.
- Certain pages are valuable because they provide information about a topic. These pages are called authorities.
- Other pages are valuable not because they provide information about any topic, but because they tell you where to go to find out about that topic. These pages are called hubs.

B. Formalizing Hubbiness and Authority

- To formalize the above intuition, we shall assign two scores to each Web page.
- One score represents the hubbiness of a page – that is, the degree to which it is a good hub, and the second score represents the degree to which the page is a good authority.

- The pages are enumerated, we represent these scores by vectors h and a. The ith component of h gives the hubbiness of the ith page, and the ith component of a gives the authority of the same page.

8. Frequent Itemsets

CONTENTS

1. The major techniques for characterizing data is the discovery of frequent itemsets.
2. This problem is often viewed as the discovery of "association rules," although the latter is a more complex characterization of data, whose discovery depends fundamentally on the discovery of frequent itemsets.
3. To begin, we introduce the "market-basket" model of data, which is essentially a many-many relationship between two kinds of elements, called "items" and "baskets," but with some assumptions about the shape of the data.
4. The frequent-itemsets problem is that of finding sets of items that appear in many of the same baskets.
5. The difference leads to a new class of algorithms for finding frequent itemsets.
6. We begin with the A-Priori Algorithm, which works by eliminating most large sets as candidates by looking first at smaller sets and recognizing that a large set cannot be frequent unless all its subsets are.
7. We then consider various improvements to the basic A-Priori idea, concentrating on very large data sets that stress the available main memory.

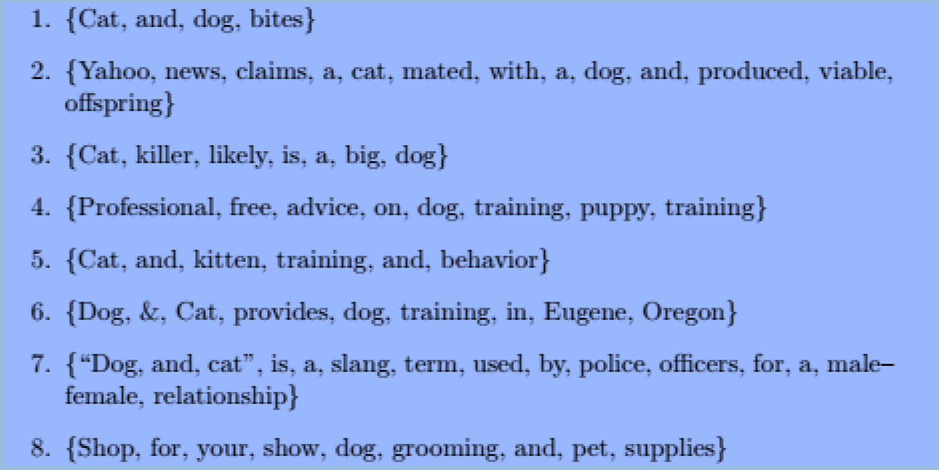
8.1 The Market Basket Model

i. The market-basket model of data is used to describe a common form of many relationship between two kinds of objects.
ii. On the one hand, we have items, and on the other we have baskets, sometimes called "transactions."
iii. Each basket consists of a set of items (an itemset), and usually we assume that the number of items in a basket is small – much smaller than the total number of items.
iv. The number of baskets is usually assumed to be very large, bigger than what can fit in main memory.
v. The data is assumed to be represented in a file consisting of a sequence of baskets.

A. **Definition of Frequent Itemsets:** Intuitively, a set of items that appears in many baskets is said to be "frequent." To be formal, we assume there is a number s, called the support threshold. I I is a set of items, the support for I is the number of baskets for which I is a subset.

Example

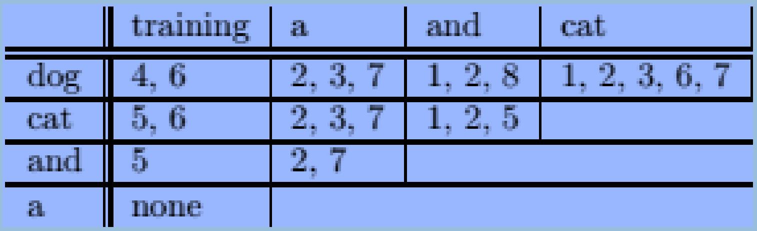
In Figure 8.1 are sets of words. Each set is a basket, and the words are items. We took these sets by Googling cat dog and taking snippets from the highest-ranked pages. Do not be concerned if a word appears twice in a basket, as baskets are sets, and in principle items can appear only once. Also, ignore capitalization.



1. {Cat, and, dog, bites}
2. {Yahoo, news, claims, a, cat, mated, with, a, dog, and, produced, viable, offspring}
3. {Cat, killer, likely, is, a, big, dog}
4. {Professional, free, advice, on, dog, training, puppy, training}
5. {Cat, and, kitten, training, and, behavior}
6. {Dog, &, Cat, provides, dog, training, in, Eugene, Oregon}
7. {"Dog, and, cat", is, a, slang, term, used, by, police, officers, for, a, male–female, relationship}
8. {Shop, for, your, show, dog, grooming, and, pet, supplies}

Figure 8.1: Here are eight baskets, each consisting of items that are words

Among the singleton sets, obviously {cat} and {dog} are quite frequent. "Dog" appears in all but basket (5), so its support is 7, while "cat" appears in all but (4) and (8), so its support is 6. The word "and" is also quite frequent; it appears in (1), (2), (5), (7), and (8), so its support is 5. The words "a" and "training" appear in three sets, while "for" and "is" appear in two each. No other word appears more than once. Suppose that we set our threshold at s = 3. Then there are five frequent singleton itemsets: {dog}, {cat}, {and}, {a}, and {training}. Now, let us look at the doubletons. A doubleton cannot be frequent unless both items in the set are frequent by themselves. Thus, there are only ten possible frequent doubletons. Figure 8.2 is a table indicating which baskets contain which doubletons.



	training	a	and	cat
dog	4, 6	2, 3, 7	1, 2, 8	1, 2, 3, 6, 7
cat	5, 6	2, 3, 7	1, 2, 5	
and	5	2, 7		
a	none			

Figure 8.2: Occurrences of doubletons

Next, let us see if there are frequent triples. In order to be a frequent triple, each pair of elements in the set must be a frequent doubleton. For example, {dog, a, and} cannot be a frequent itemset, because if it were, then surely {a, and} would be frequent, but it is not. The triple {dog, cat, and} might be frequent, because each of its doubleton subsets is frequent. Unfortunately, the three words appear together only in baskets (1) and (2), so there are in fact no frequent triples. The triple {dog, cat, a} might be frequent, since its doubletons are all frequent. In fact, all three words do appear in baskets (2), (3), and (7), so it is a frequent triple. No other triple of words is even a candidate for being a frequent triple, since for no other triple of words are its three doubleton subsets frequent. As there is only one frequent triple, there can be no frequent quadruples or larger sets.

B. **Applications of Frequent Itemsets:** The original application of the market-basket model was in the analysis of true market baskets. That is, supermarkets and chain stores record the contents of every market basket (physical shopping cart) brought to the register for checkout. Here the “items” are the different products that the store sells, and the “baskets” are the sets of items in a single market basket. A major chain might sell 100,000 different items and collect data about millions of market baskets. By finding frequent itemsets, a retailer can learn what is commonly bought together. Especially important are pairs or larger sets of items that occur much more frequently than would be expected were the items bought independently. We will discover by this analysis that many people buy bread and milk together, but that is of little interest, since we already knew that these were popular items individually. We might discover that many people buy hot dogs and mustard together. That, again, should be no surprise to people who like hot dogs, but it offers the supermarket an opportunity to do some clever marketing. They can advertise a sale on hot dogs and raise the price of mustard. When people come to the store for the cheap hot dogs, they often will remember that they need mustard, and buy that too. Either they will not notice the price is high, or they reason that it is not worth the trouble to go somewhere else for cheaper mustard.

C. **Applications of frequent-itemset analysis** is not limited to market baskets. The same model can be used to mine many other kinds of data. Some examples are:

- ***Related concepts***: Let items be words, and let baskets be documents (e.g., Web pages, blogs, and tweets). A basket/document contains those items/words that are present in the document. If we look for sets of words that appear together in many documents, the sets will be above. There, even though the intent was to find snippets that talked about cats and dogs, the stop words “and” and “a” were prominent among the frequent itemsets. However, if we ignore all the most common words, then we would hope to find among the frequent pairs some pairs of words that represent a joint concept. For example, we would expect a pair like {Brad, Angelina} to appear with surprising frequency.
- ***Plagiarism***: Let the items be documents and the baskets be sentences. An item/document is “in” a basket/sentence if the sentence is in the document. This arrangement appears backwards, but it is exactly what we need, and we should remember that the relationship between items and baskets is an arbitrary many-many relationship. That is, “in” need not have its conventional meaning: “part of.” In this application, we look for pairs of items that appear together in several baskets. If we find such a pair, then

we have two documents that share several sentences in common. In practice, even one or two sentences in common is a good indicator of plagiarism.

- ***Biomarkers***: Let the items be of two types – biomarkers such as genes or blood proteins, and diseases. Each basket is the set of data about a patient: their genome and blood-chemistry analysis, as well as their medical history of disease. A frequent itemset that consists of one disease and one or more biomarkers suggests a test for the disease.

D. Association Rules: While the subject of this chapter is extracting frequent sets of items from data, this information is often presented as a collection of if–then rules, called association rules. The form of an association rule is I → j, where I is a set of items and j is an item. The implication of this association rule is that if all of the items in I appear in some basket, then j is "likely" to appear in that basket as well.

8.2 Handling Larger Datasets in Main Memory

i. The PCY Algorithm, which takes advantage of the fact that in the first pass of A-Priori there is typically lots of main memory not needed for the counting of single items.

ii. The Multistage Algorithm, which uses the PCY trick and also inserts extra passes to further reduce the size of C2.

A. The Algorithm of Park, Chen, and Yu

- The PCY Algorithm uses that space for an array of integers that generalizes the idea of a Bloom filter.

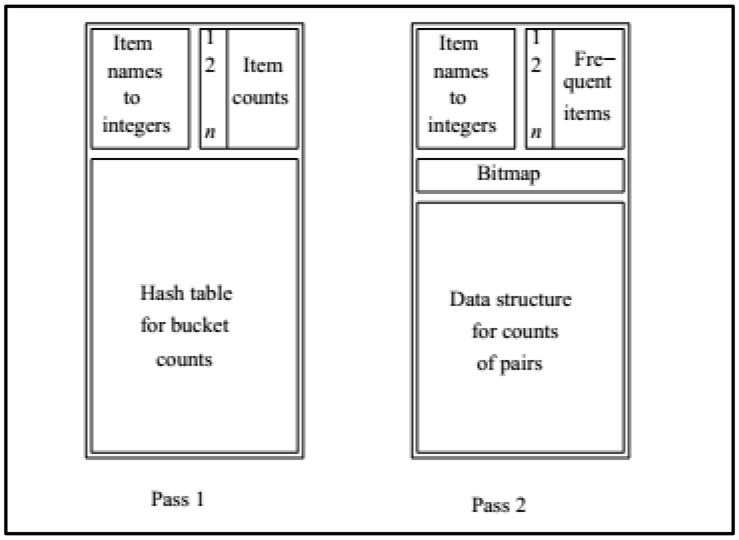


Figure 8.3: Algorithm PCY

- The algorithm, PCY after its authors, works on the observation that there may be much unused space in main memory on the first pass.
- An array is considered as an hash table, whose buckets hold integers rather than sets of keys (as in an ordinary hash table) or bits (as in a Bloom filter).
- Pairs of items are hashed to buckets of this hash table.
- As we examine a basket during the first pass, we not only add 1 to the count for each item in the basket, but we generate all the pairs, using a double loop.
- We hash each pair, and we add 1 to the bucket into which that pair hashes. Note that the pair itself doesn't go into the bucket; the pair only affects the single integer in the bucket.
- At the end of the first pass, each bucket has a count, which is the sum of the counts of all the pairs that hash to that bucket.
- If the count of a bucket is at least as great as the support threshold s, it is called a frequent bucket.
- We can say nothing about the pairs that hash to a frequent bucket; they could all be frequent pairs from the information available to us.

- But if the count of the bucket is less than s (an infrequent bucket), we know no pair that hashes to this bucket can be frequent, even if the pair consists of two frequent items.

B. The Multistage Algorithm

- The Multistage Algorithm improves upon PCY by using several successive hash tables to reduce further the number of candidate pairs.
- The tradeoff is that Multistage takes more than two passes to find the frequent pairs.
- An outline of the Multistage Algorithm is shown below

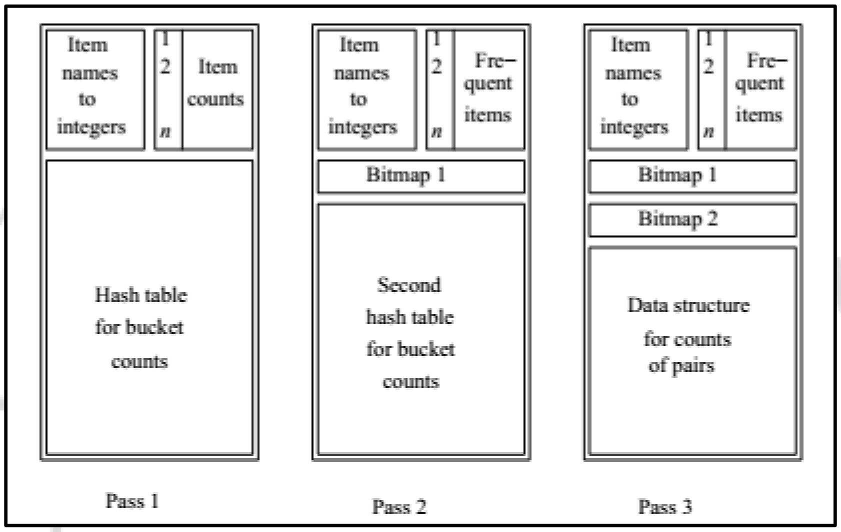


Figure 8.4: Multistage Algorithm

- The **first pass of Multistage** is the same as the first pass of PCY.
- After that pass, the frequent buckets are identified and summarized by a bitmap, again the same as in PCY.
- The **second pass of Multistage** does not count the candidate pairs. Rather, it uses the available main memory for another hash table, using another hash function.
- Since the bitmap from the first hash table takes up 1/32 of the available main memory, the second hash table has almost as many buckets as the first.

- After the second pass, the second hash table is also summarized as a bitmap, and that bitmap is stored in main memory.
- The two bitmaps together take up slightly less than 1/16th of the available main memory, so there is still plenty of space to count the candidate pairs on the third pass.

C. The Multihash Algorithm

- The benefit of the extra passes of the Multistage Algorithm in a single pass.
- This variation of PCY is called the Multihash Algorithm. Instead of using two different hash tables on two successive passes, use two hash functions and two separate hash tables that share main memory on the first pass, as shown in figure below.

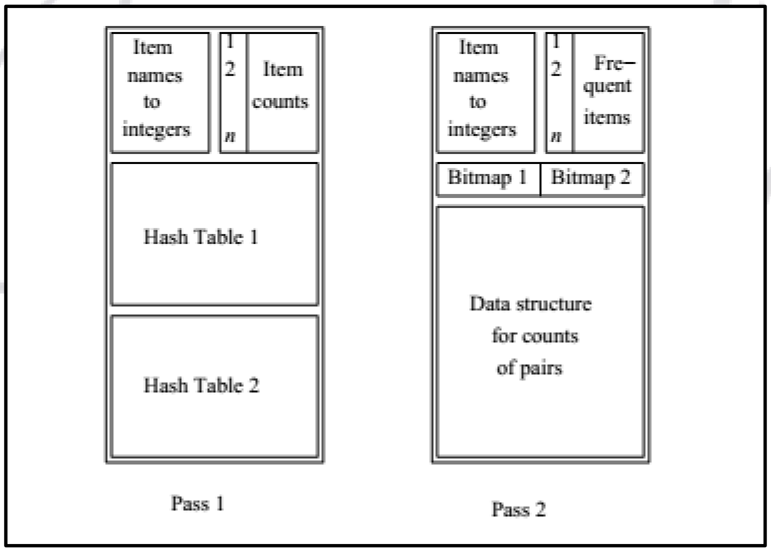


Figure 8.3: Multihash Algorithm

- The danger of using two hash tables on one pass is that each hash table has half as many buckets as the one large hash table of PCY.
- As long as the average count of a bucket for PCY is much lower than the support threshold, we can operate two half-sized hash tables and still expect most of the buckets of both hash tables to be infrequent.
- Thus, in this situation we might well choose the Multihash approach.

8.3 The SON Algorithm and MapReduce

i. The SON algorithm lends itself well to a parallel-computing environment.
ii. Each of the chunks can be processed in parallel, and the frequent itemsets from each chunk combined to form the candidates.
iii. Distribute the candidates to many processors, have each processor count the support for each candidate in a subset of the baskets, and finally sum those supports to get the support for each candidate itemset in the whole dataset.
iv. This process does not have to be implemented in MapReduce, but there is a natural way of expressing each of the two passes as a MapReduce operation.
v. The MapReduce sequence is:
   - **First Map Function**: Take the assigned subset of the baskets and find the itemsets frequent in the subset using the algorithm. As described there, lower the support threshold from s to ps if each Map task gets fraction p of the total input file. The output is a set of key-value pairs (F, 1), where F is a frequent itemset from the sample. The value is always 1 and is irrelevant.
   - **First Reduce Function**: Each Reduce task is assigned a set of keys, which are itemsets. The value is ignored, and the Reduce task simply produces those keys (itemsets) that appear one or more times. Thus, the output of the first Reduce function is the candidate itemsets.
   - **Second Map Function**: The Map tasks for the second Map function take all the output from the first Reduce Function (the candidate itemsets) and a portion of the input data file. Each Map task counts the number of occurrences of each of the candidate itemsets among the baskets in the portion of the dataset that it was assigned. The output is a set of key-value pairs (C, v), where C is one of the candidate sets and v is the support for that itemset among the baskets that were input to this Map task.
   - **Second Reduce Function**: The Reduce tasks take the itemsets they are given as keys and sum the associated values. The result is the total support

for each of the itemsets that the Reduce task was assigned to handle. Those itemsets whose sum of values is at least s are frequent in the whole dataset, so the Reduce task outputs these itemsets with their counts. Itemsets that do not have total support at least s are not transmitted to the output of the Reduce task.

8.4 Counting Frequent Items in a Stream

i. A clear distinction between streams and files, when frequent itemsets are considered, is that there is no end to a stream, so eventually an itemset is going to exceed the support threshold, as long as it appears repeatedly in the stream.
ii. As a result, for streams, we must think of the support threshold s as a fraction of the baskets in which an itemset must appear in order to be considered frequent.
iii. Even with this adjustment, we still have several options regarding the portion of the stream over which that fraction is measured.

A. Sampling Methods for Streams

- Assume that stream elements are baskets of items.
- Perhaps the simplest approach to maintaining a current estimate of the frequent itemsets in a stream is to collect some number of baskets and store it as a file.
- Run one of the frequent-itemset algorithms discussed in this chapter, meanwhile ignoring the stream elements that arrive, or storing them as another file to be analyzed later.
- When the frequent-itemsets algorithm finishes, we have an estimate of the frequent itemsets in the stream. We then have several options.
- We can use this collection of frequent itemsets for whatever application is at hand, but start running another iteration of the chosen frequent-itemset algorithm immediately.
- This algorithm can either:

    - ✓ Use the file that was collected while the first iteration of the algorithm was running. At the same time, collect yet another file to be used at another iteration of the algorithm, when this current iteration finishes.
    - ✓ Start collecting another file of baskets now, and run the algorithm when an adequate number of baskets has been collected.

- ➢ Continue to count the numbers of occurrences of each of these frequent itemsets, along with the total number of baskets seen in the stream, since the counting started.
- ➢ If any itemset is discovered to occur in a fraction of the baskets that is significantly below the threshold fraction s, then this set can be dropped from the collection of frequent itemsets.
- ➢ When computing the fraction, it is important to include the occurrences from the original file of baskets, from which the frequent itemsets were derived. If not, we run the risk that we shall encounter a short period in which a truly frequent itemset does not appear sufficiently frequently and throw it out.
- ➢ We should also allow some way for new frequent itemsets to be added to the current collection.
- ➢ Possibilities include:
    - ✓ Periodically gather a new segment of the baskets in the stream and use it as the data file for another iteration of the chosen frequent-itemsets algorithm. The new collection of frequent items is formed from the result of this iteration and the frequent itemsets from the previous collection that have survived the possibility of having been deleted for becoming infrequent.
    - ✓ Add some random itemsets to the current collection, and count their fraction of occurrences for a while, until one has a good idea of whether or not they are currently frequent. Rather than

choosing new itemsets completely at random, one might focus on sets with items that appear in many itemsets already known to be frequent.

B. Frequent Itemsets in Decaying Windows

- The first is simple. Stream elements are baskets rather than individual items, so many items may appear at a given stream element.
- Treat each of those items as if they were the "current" item and add 1 to their score after multiplying all current scores by 1 – c.
- If some items in a basket have no current score, initialize the scores of those items to 1.
- The second modification is trickier.
- We want to find all frequent itemsets, not just singleton itemsets. If we were to initialize a count for an itemset whenever we saw it, we would have too many counts. For example, one basket of 20 items has over a million subsets, and all of these would have to be initiated for one basket.
- On the other hand, as we mentioned, if we use a requirement above 1 for initiating the scoring of an itemset, then we would never get any itemsets started, and the method would not work.

9. Clustering

CONTENTS

9.1 Introduction to Clustering

1. Clustering is the process of examining a collection of "points," and grouping the points into "clusters" according to some distance measure.
2. The goal is that points in the same cluster have a small distance from one another, while points in different clusters are at a large distance from one another.
3. A suggestion of what clusters might look like was seen in Figure 9.1.
4. However, there the intent was that there were three clusters around three different road intersections, but two of the clusters blended into one another because they were not sufficiently separated.

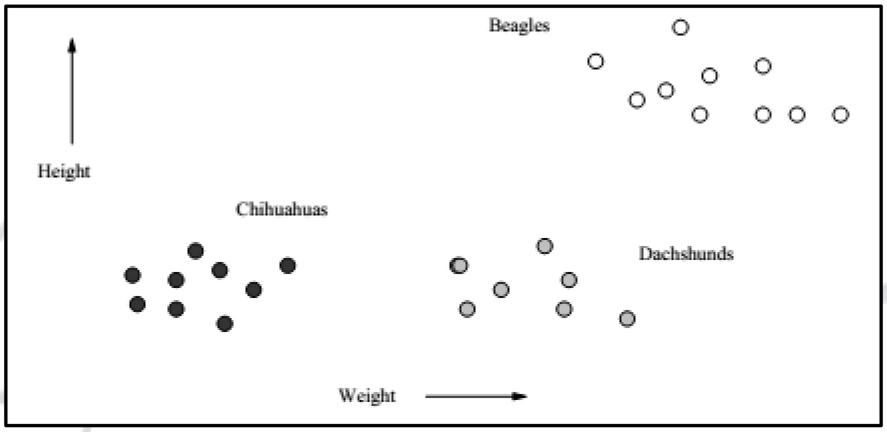


Figure 9.1: Heights and weights of dogs taken from three varieties

9.2 Clustering Strategies

1. **Hierarchical or agglomerative algorithms** start with each point in its own cluster. Clusters are combined based on their "closeness," using one of many possible definitions of "close." Combination stops when further combination leads to clusters that are undesirable for one of several reasons. For example, we may stop when we have a predetermined number of clusters, or we may use a measure of compactness for clusters, and refuse to construct a cluster by combining two smaller clusters if the resulting cluster has points that are spread out over too large a region.
2. The other class of algorithms involve **point assignment**. Points are considered in some order, and each one is assigned to the cluster into which it best fits. This

process is normally preceded by a short phase in which initial clusters are estimated. Variations allow occasional combining or splitting of clusters, or may allow points to be unassigned if they are outliers (points too far from any of the current clusters).

3. Algorithms for clustering can also be distinguished by:
   i. Whether the algorithm assumes a Euclidean space, or whether the algorithm works for an arbitrary distance measure.
   ii. A key distinction is that in a Euclidean space it is possible to summarize a collection of points by their centroid – the average of the points. In a non-Euclidean space, there is no notion of a centroid, and we are forced to develop another way to summarize clusters.
   iii. Whether the algorithm assumes that the data is small enough to fit in main memory, or whether data must reside in secondary memory, primarily. Algorithms for large amounts of data often must take shortcuts, since it is infeasible to look at all pairs of points, for example. It is also necessary to summarize clusters in main memory, since we cannot hold all the points of all the clusters in main memory at the same time.

9.3 Hierarchical Clustering

1. Considering hierarchical clustering in a Euclidean space.
   - We begin with every point in its own cluster. As time goes on, larger clusters will be constructed by combining two smaller clusters, and we have to decide in advance:
     1. How will clusters be represented?
     2. 2. How will we choose which two clusters to merge?
     3. 3. When will we stop combining clusters?
2. This algorithm can only be used for relatively small datasets, but even so, there are some efficiencies we can make by careful implementation.

3. When the space is non-Euclidean, there are additional problems associated with hierarchical clustering.
4. Consider “clustroid” and the way we can represent a cluster when there is no centroid or average point in a cluster.

9.4 CURE Algorithm

i. CURE (Clustering Using REpresentatives), assumes a Euclidean space.

ii. This does not assume anything about the shape of clusters; they need not be normally distributed, and can even have strange bends, S-shapes, or even rings.

iii. Instead of representing clusters by their centroid, it uses a collection of representative points, as the name implies.

iv. Representation of two clusters. The inner cluster is an ordinary circle, while the second is a ring around the circle.

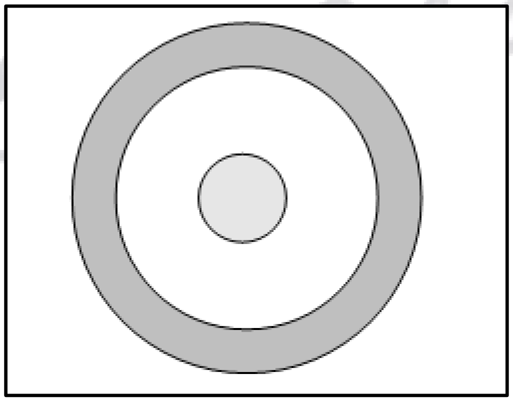


Figure 9.2: Two clusters, one surrounding the other

v. This arrangement is not completely pathological.

vi. A creature from another galaxy might look at our solar system and observe that the objects cluster into an inner circle (the planets) and an outer ring (the Kuyper belt), with little in between.

A. Initialization in CURE

- Take a small sample of the data and cluster it in main memory.
- In principle, any clustering method could be used, but as CURE is designed to handle oddly shaped clusters, it is often advisable to use a hierarchical

method in which clusters are merged when they have a close pair of points, as shown below.

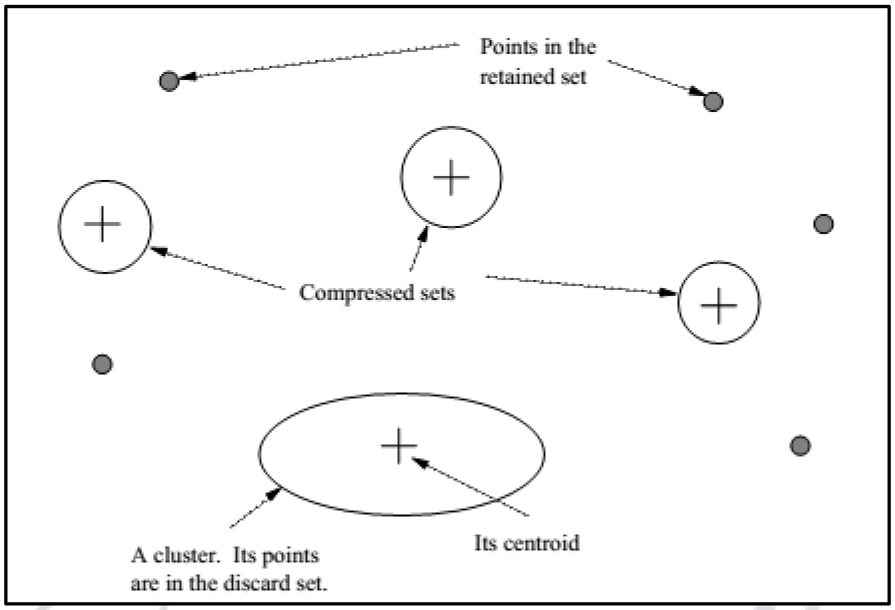

Points in the retained set
Compressed sets
A cluster. Its points are in the discard set.
Its centroid

Figure 9.3: Clustering

- Select a small set of points from each cluster to be representative points.
- These points should be chosen to be as far from one another as possible.
- Move each of the representative points a fixed fraction of the distance between its location and the centroid of its cluster.
- Perhaps 20% is a good fraction to choose.
- Each step requires a Euclidean space, since otherwise, there might not be any notion of a line between two points.

B. Completion of the CURE Algorithm

- The next phase of CURE is to merge two clusters if they have a pair of representative points, one from each cluster, that are sufficiently close.
- The user may pick the distance that defines “close.”
- This merging step can repeat, until there are no more sufficiently close clusters.

9.5 Stream Computing

i. Assuming that each stream element is a point in some space.

ii. The sliding window consists of the most recent N points.

iii. Our goal is to prelisted subsets of the points in the stream, so that we may quickly answer queries of the form "what are the clusters of the last m points?" for any m ≤ N.

iv. There are many variants of this query, depending on what we assume about what constitutes a cluster.

v. For instance, we may use a k-means approach, where we are really asking that the last m points be partitioned into exactly k clusters.

vi. The problem is considerably easier if we assume that all stream elements are chosen with statistics that do not vary along the stream.

vii. Then, a sample of the stream is good enough to estimate the clusters, and we can in effect ignore the stream after a while.

viii. However, the stream model normally assumes that the statistics of the stream elements varies with time.

ix. For example, the centroids of the clusters may migrate slowly as time goes on, or clusters may expand, contract, divide, or merge.

A. A Stream-Clustering Algorithm

- Simplified version of an algorithm referred to as BDMO (for the authors, B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan).
- The true version of the algorithm involves more complex structures, which are designed to provide performance guarantees in the worst case.
- The BDMO Algorithm builds on the methodology for counting ones stream.
- Here are the key similarities and differences:
    - ✓ Like that algorithm, the points of the stream are partitioned into, and summarized by, buckets whose sizes are a power of two.
    - ✓ Here, the size of a bucket is the number of points it represents, rather than the number of stream elements that are 1.

  - ✓ As before, the sizes of buckets obey the restriction that there are one or two of each size, up to some limit.
  - ✓ wever, we do not assume that the sequence of allowable bucket sizes starts with 1. Rather, they are required only to form a sequence where each size is twice the previous size, e.g., 3, 6, 12, 24, ...

- ➢ Bucket sizes are again restrained to be non-decreasing as we go back in time.
- ➢ The contents of a bucket consists of:
  - ✓ The size of the bucket.
  - ✓ The timestamp of the bucket, that is, the most recent point that contributes to the bucket, timestamp can be recorded for module N.
  - ✓ A collection of records that represent the clusters into which the points of that bucket have been partitioned.
  - ✓ These records contain:
    a. The number of points in the cluster.
    b. The centroid or clustroid of the cluster.
    c. Any other parameters necessary to enable us to merge clusters and maintain approximations to the full set of parameters for the merged cluster.

B. Initializing Buckets

- ➢ Our smallest bucket size will be p, a power of 2.
- ➢ Thus, every p stream elements, we create a new bucket, with the most recent p points.
- ➢ The timestamp for this bucket is the timestamp of the most recent point in the bucket.
- ➢ Leave each point in a cluster by itself, or we may perform a clustering of these points according to whatever clustering strategy we have chosen.

- For instance, if we choose a k-means algorithm, then (assuming $k < p$) we cluster the points into k clusters by some algorithm.
- Whatever method we use to cluster initially, we assume it is possible to compute the centroids or clustroid for the clusters and count the points in each cluster.
- This information becomes part of the record for each cluster.
- Compute whatever other parameters for the clusters will be needed in the merging process.

C. Merging Buckets

- We create a new bucket, we need to review the sequence of buckets.
- First, if some bucket has a timestamp that is more than N time units prior to the current time, then nothing of that bucket is in the window, and we may drop it from the list.
- Second, we may have created three buckets of size p, in which case we must merge the oldest two of the three.
- The merger may create two buckets of size 2p, in which case we may have to merge buckets of increasing sizes, recursively.
- To merge two consecutive buckets, we need to do several things:
    - ✓ The size of the bucket is twice the sizes of the two buckets being merged.
    - ✓ The timestamp for the merged bucket is the timestamp of the more recent of the two consecutive buckets.
    - ✓ Consider whether to merge clusters, and if so, we need to compute the parameters of the merged clusters.
    - ✓ Elaborate on this part of the algorithm by considering several examples of criteria for merging and ways to estimate the needed parameters.

D. Answering Queries

- Assuming a query is a request for the clusters of the most recent m points in the stream, where m ≤ N.
- Because of the strategy we have adopted of combining buckets as we go back in time, we may not be able to find a set of buckets that covers exactly the last m points.
- If we choose the smallest set of buckets that cover the last m points, we shall include in these buckets no more than the last 2m points.
- We shall produce, as answer to the query, the centroids or clustroid of all the points in the selected buckets.
- In order for the result to be a good approximation to the clusters for exactly the last m points, we must assume that the points between 2m and m + 1 will not have radically different statistics from the most recent m points.
- Reducing the error that a more complex bucketing scheme can guarantee that we can find buckets to cover at most the last $m(1 + \epsilon)$ points, for any $\epsilon > 0$.

10. Recommendation Systems

CONTENTS

i. The various implementation of Web applications that involve predicting user responses to options. This system is called a recommendation system.

ii. To bring the problem into focus, two good examples of recommendation systems are:

- Offering news articles to on-line newspaper readers, based on a prediction of reader interests.
- Offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases and/or product searches.

iii. Recommendation systems use a number of different technologies, which can be classified into two broad groups.

- Content-based systems examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.
- Collaborative filtering systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users.

10.1 A Model for Recommendation Systems

i. A model for recommendation systems, based on a utility matrix of preferences.

ii. The concept of a "long-tail," which explains the advantage of on-line vendors over conventional, brick-and mortar vendors.

A. The Utility Matrix

- A recommendation-system application there are two classes of entities, which we shall refer to as users and items.
- Users have preferences for certain items, and these preferences must be teased out of the data.

- The data itself is represented as a utility matrix, giving for each user-item pair, a value that represents what is known about the degree of preference of that user for that item.
- Values come from an ordered set, e.g., integers 1–5 representing the number of stars that the user gave as a rating for that item.
- Assuming the matrix is sparse, meaning that most entries are “unknown.” An unknown rating implies that we have no explicit information about the user’s preference for the item.

B. The Long Tail

- The long tail phenomenon that makes recommendation systems necessary.
- Physical delivery systems are characterized by a scarcity of resources.
- Brick-and-mortar stores have limited shelf space, and can show the customer only a small fraction of all the choices that exist.
- On the other hand, on-line stores can make anything that exists available to the customer.
- Thus, a physical bookstore may have several thousand books on its shelves, but Amazon offers millions of books.
- A physical newspaper can print several dozen articles per day, while on-line news services offer thousands per day.
- Recommendation in the physical world is fairly simple.
- First, it is not possible to tailor the store to each individual customer. Thus, the choice of what is made available is governed only by the aggregate numbers.
- Typically, a bookstore will display only the books that are most popular, and a newspaper will print only the articles it believes the most people will be interested in.
- In the first case, sales figures govern the choices, in the second case, editorial judgement serves.

C. Applications of Recommendation Systems

- **Product Recommendations**: Perhaps the most important use of recommendation systems is at on-line retailers. We have noted how Amazon or similar on-line vendors strive to present each returning user with some suggestions of products that they might like to buy.

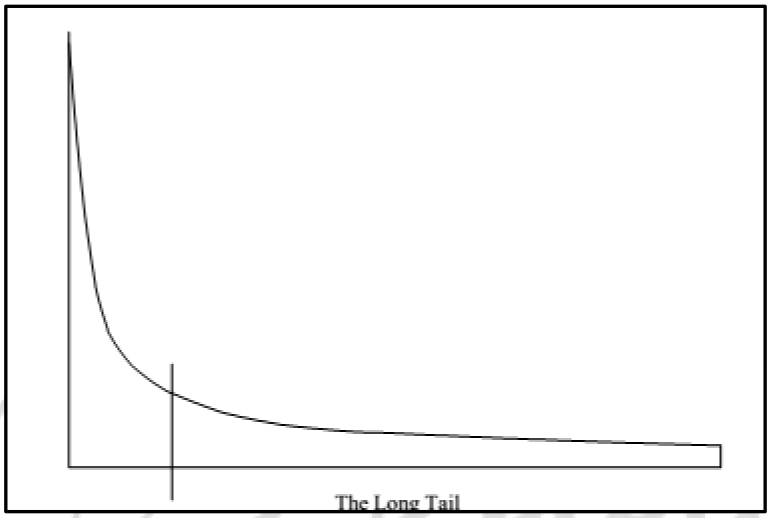


Figure 10.1: The long tail: physical institutions can only provide what is popular, while on-line institutions can make everything available

- **Movie Recommendations**: Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users, much like the ratings suggested in the example utility matrix. The importance of predicting ratings accurately is so high, that Netflix offered a prize of one million dollars for the first algorithm that could beat its own recommendation system by 10%.
- **News Articles**: News services have attempted to identify articles of interest to readers, based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents, or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the

millions of blogs available, videos on YouTube, or other sites where content is provided regularly.

10.2 Content-Based Recommendations

i. The two basic architectures for a recommendation system:

- ➢ Content-Based systems focus on properties of items. Similarity of items is determined by measuring the similarity in their properties.
- ➢ Collaborative-Filtering systems focus on the relationship between users and items. Similarity of items is determined by the similarity of the ratings of those items by the users who have rated both items.

A. Item Profiles

- ➢ In a content-based system, we must construct for each item a profile, which is a record or collection of records representing important characteristics of that item.
- ➢ The profile consists of some characteristics of the item that are easily discovered.
- ➢ For example, consider the features of a movie that might be relevant to a recommendation system.
  - ✓ The set of actors of the movie. Some viewers prefer movies with their favorite actors.
  - ✓ The director. Some viewers have a preference for the work of certain directors.
  - ✓ The year in which the movie was made. Some viewers prefer old movies; others watch only the latest releases.
  - ✓ The genre or general type of movie. Some viewers like only comedies, others dramas or romances.

B. Discovering Features of Documents

- ➢ There are many kinds of documents for which a recommendation system can be useful.

- For example, there are many news articles published each day, and we cannot read all of them.
- A recommendation system can suggest articles on topics a user is interested in, but how can we distinguish among topics? Web pages are also a collection of documents.
- These classes of documents do not tend to have readily available information giving features.
- A substitute that has been useful in practice is the identification of words that characterize the topic of a document.

C. Obtaining Item Features From Tags

- There have been a number of attempts to obtain information about features of items by inviting users to tag the items by entering words or phrases that describe the item.
- Thus, one picture with a lot of red might be tagged "Tiananmen Square," while another is tagged "sunset at Malibu."
- The distinction is not something that could be discovered by existing image-analysis programs.

D. Representing Item Profiles

- Ultimate goal for content-based recommendation is to create both an item profile consisting of feature-value pairs and a user profile summarizing the preferences of the user, based on their row of the utility matrix.
- To generalize this vector approach to all sorts of features. It is easy to do so for features that are sets of discrete values.
- For example, if one feature of movies is the set of actors, then imagine that there is a component for each actor, with 1 if the actor is in the movie, and 0 if not.
- We can have a component for each possible director, and each possible genre.

E. User Profiles

- We not only need to create vectors describing items; we need to create vectors with the same components that describe the user's preferences. We have the utility matrix representing the connection between users and items. Recall the nonblank matrix entries could be just 1's representing user purchases or a similar connection, or they could be arbitrary numbers representing a rating or degree of affection that the the user has for the item.
- With this information, the best estimate we can make regarding which items the user likes is some aggregation of the profiles of those items.
- If the utility matrix has only 1's, then the natural aggregate is the average of the components of the vectors representing the item profiles for the items in which the utility matrix has 1 for that user.

F. Recommending Items to Users Based on Content

- With profile vectors for both users and items, we can estimate the degree to which a user would prefer an item by computing the cosine distance between the user's and item's vectors.
- The random-hyperplane and locality-sensitive-hashing techniques can be used to place (just) item profiles in buckets.
- Given a user to whom we want to recommend some items, we can apply the same two techniques – random hyperplanes and LSH – to determine in which buckets we must look for items that might have a small cosine distance from the user.

10.3 Collaborative Filtering

i. A significantly different approach to recommendation, using features of items to determine their similarity, focusing on the similarity of the user ratings for two items.
ii. Place of the item-profile vector for an item, we use its column in the utility matrix.

iii. Contriving a profile vector for users, we represent them by their rows in the utility matrix.

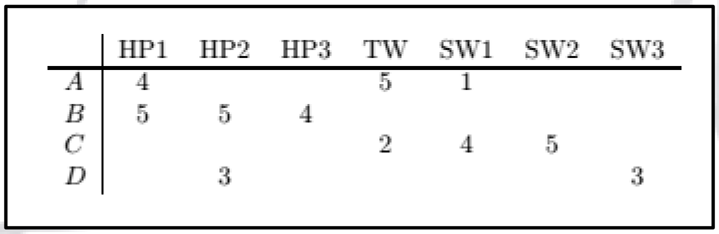
iv. Users are similar if their vectors are close according to some distance measure such as Jaccard or cosine distance.

v. Recommendation for a user U is then made by looking at the users that are most similar to U in this sense, and recommending items that these users like.

vi. The process of identifying similar users and recommending what similar users like is called collaborative filtering.

A. Measuring Similarity

- The first question we must deal with is how to measure similarity of users or items from their rows or columns in the utility matrix.
- The utility matrix is as shown below



	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Figure 10.2: The utility matrix

- The above data is too small to draw any reliable conclusions, but its small size will make clear some of the pitfalls in picking a distance measure.
- Specifically the users A and C. They rated two movies in common, but they appear to have almost diametrically opposite opinions of these movies.
- Expect that a good distance measure would make them rather far apart.
- The alternative measures to consider are:

a. Jaccard Distance

- ✓ Ignore values in the matrix and focus only on the sets of items rated.
- ✓ If the utility matrix only reflected purchases, this measure would be a good one to choose.

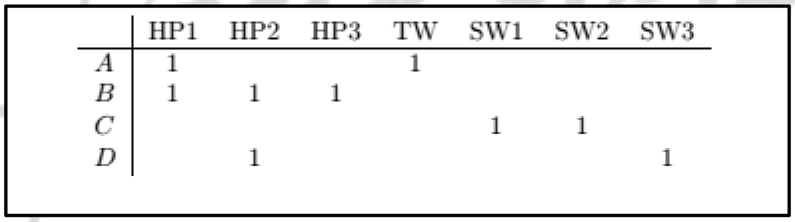
- ✓ When utilities are more detailed ratings, the Jaccard distance loses important information.

b. Cosine Distance

- ✓ Treat blanks as a 0 value.
- ✓ This choice is questionable, since it has the effect of treating the lack of a rating as more similar to disliking the movie than liking it.

c. Rounding the Data

- ✓ Try to eliminate the apparent similarity between movies a user rates highly and those with low scores by rounding the ratings.
- ✓ For instance, we could consider ratings of 3, 4, and 5 as a "1" and consider ratings 1 and 2 as unrated. The utility matrix would then look as shown below.



	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	1			1			
B	1	1	1				
C					1	1	
D		1					1

Figure 10.3: Utilities of 3, 4, and 5 have been replaced by 1, while ratings of 1 and 2 are omitted

- ✓ Now, the Jaccard distance between A and B is 3/4, while between A and C it is 1; i.e., C appears further from A than B does, which is intuitively correct.
- ✓ Applying cosine distance to Figure above allows us to draw the same conclusion.

d. Normalizing Ratings

- ✓ If we normalize ratings, by subtracting from each rating the average rating of that user, we turn low ratings into negative numbers and high ratings into positive numbers.

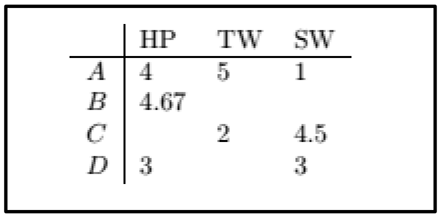
- ✓ If we then take the cosine distance, we find that users with opposite views of the movies they viewed in common will have vectors in almost opposite directions, and can be considered as far apart as possible.
- ✓ Users with similar opinions about the movies rated in common will have a relatively small angle between them.

B. The Duality of Similarity

- ➢ The utility matrix can be viewed as telling us about users or about items, or both.
- ➢ There are two ways in which the symmetry is broken in practice.
  - ✓ We can use information about users to recommend items. We can base our recommendation on the decisions made by these similar users, e.g., recommend the items that the greatest number of them have purchased or rated highly. There is no symmetry. Even if we find pairs of similar items, we need to take an additional step in order to recommend items to users. This point is explored further at the end of this subsection.
  - ✓ There is a difference in the typical behavior of users and items, as it pertains to similarity. Intuitively, items tend to be classifiable in simple terms. For example, music tends to belong to a single genre.

C. Clustering Users and Items

- ➢ It is hard to detect similarity among either items or users, because we have little information about user-item pairs in the sparse utility matrix.
- ➢ Even if two users both like a genre or genres, they may not have bought any items in common.
- ➢ One way of dealing with this pitfall is to cluster items and/or users.
- ➢ There may be little reason to try to cluster into a small number of clusters immediately.



	HP	TW	SW
A	4	5	1
B	4.67		
C		2	4.5
D	3		3

Figure 10.4: Utility matrix for users and clusters of items

- A hierarchical approach, where we leave many clusters unmerged may suffice as a first step.
- For example, we might leave half as many clusters as there are items.

11. Mining Social-Network Graphs

CONTENTS

i. Information to be gained by analyzing the large-scale data that is derived from social networks.
ii. The best-known example of a social network is the “friends” relation found on sites like Facebook. However, as we shall see there are many other sources of data that connect people or other entities.
iii. Communities almost never partition the set of nodes in a network. Rather, communities usually overlap. For example, you may belong to several communities of friends or classmates.
iv. The people from one community tend to know each other, but people from two different communities rarely know each other.
v. You would not want to be assigned to only one of the communities, nor would it make sense to cluster all the people from all your communities into one cluster.

11.1 Social Networks as Graphs

i. Every graph is a suitable representation of what we intuitively regard as a social network.
ii. The property of social networks that says nodes and edges of the graph tend to cluster in communities.

A. What is a Social Network?

- A social network, we think of Facebook, Twitter, Google+, or another website that is called a “social network,” and indeed this kind of network is representative of the broader class of networks called “social.”
- The essential characteristics of a social network are:
  - ✓ There is a collection of entities that participate in the network. These entities are people, but they could be something else entirely.
  - ✓ There is at least one relationship between entities of the network. On Facebook or its ilk, this relationship is called friends. Sometimes the relationship is all-or-nothing; two people are either friends or

they are not. However, in other examples of social networks, the relationship has a degree. This degree could be discrete; e.g., friends, family, acquaintances, or none as in Google+. It could be a real number; an example would be the fraction of the average day that two people spend talking to each other.

- ✓ There is an assumption of no randomness or locality. This condition is the hardest to formalize, but the intuition is that relationships tend to cluster. That is, if entity A is related to both B and C, then there is a higher probability than average that B and C are related.

B. Social Networks as Graphs

- Social networks are naturally modeled as graphs, which we sometimes refer to as a social graph.
- The entities are the nodes, and an edge connects two nodes if the nodes are related by the relationship that characterizes the network.
- If there is a degree associated with the relationship, this degree is represented by labeling the edges.
- Social graphs are undirected, as for the Facebook friend's graph. But they can be directed graphs, as for example the graphs of followers on Twitter or Google+.

C. Varieties of Social Networks

- **Telephone Networks:** Here the nodes represent phone numbers, which are really individuals. There is an edge between two nodes if a call has been placed between those phones in some fixed period of time, such as last month, or "ever." The edges could be weighted by the number of calls made between these phones during the period. Communities in a telephone network will form from groups of people that communicate frequently: groups of friends, members of a club, or people working at the same company, for example.

- **Email Networks:** The nodes represent email addresses, which are again individuals. An edge represents the fact that there was at least one email in at least one direction between the two addresses. Alternatively, we may only place an edge if there were emails in both directions. In that way, we avoid viewing spammers as "friends" with all their victims. Another approach is to label edges as weak or strong. Strong edges represent communication in both directions, while weak edges indicate that the communication was in one direction only. The communities seen in email networks come from the same sorts of groupings we mentioned in connection with telephone networks. A similar sort of network involves people who text other people through their cell phones.
- **Collaboration Networks**: Nodes represent individuals who have published research papers. There is an edge between two individuals who published one or more papers jointly. Optionally, we can label edges by the number of joint publications. The communities in this network are authors working on a particular topic.

D. Graphs With Several Node Types

- There are other social phenomena that involve entities of different types.
- Several kinds of graphs that are really formed from two types of nodes. Authorship networks can be seen to have author nodes and paper nodes.
- A tripartite graph representing users, tags, and Web pages.

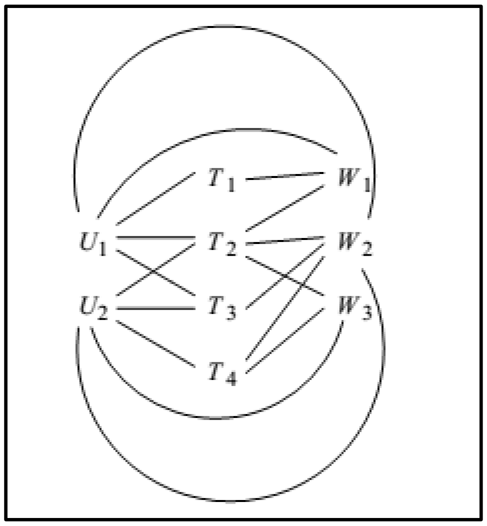


Figure 11.1 Graph of users, tags and web pages

11.2 Clustering of Social-Network Graphs

An important aspect of social networks is that they contain communities of entities that are connected by many edges.

A. Distance Measures for Social-Network Graphs

- Apply standard clustering techniques to a social-network graph, our first step would be to define a distance measure.
- When the edges of the graph have labels, these labels might be usable as a distance measure, depending on what they represented.
- But when the edges are unlabeled, as in a "friends" graph, there is not much we can do to define a suitable distance.

B. Applying Standard Clustering Methods

- Hierarchical clustering of a social-network graph starts by combining some two nodes that are connected by an edge.
- Successively, edges that are not between two nodes of the same cluster would be chosen randomly to combine the clusters to which their two nodes belong.
- The choices would be random, because all distances represented by an edge are the same.

C. Betweenness

- There are problems with standard clustering methods, several specialized clustering techniques have been developed to find communities in social networks.
- Define the betweenness of an edge (a, b) to be the number of pairs of nodes x and y such that the edge (a, b) lies on the shortest path between x and y.
- To be more precise, since there can be several shortest paths between x and y, edge (a, b) is credited with the fraction of those shortest paths that include the edge (a, b).
- As in golf, a high score is bad. It suggests that the edge (a, b) runs between two different communities; that is, a and b do not belong to the same community.

D. The Girvan-Newman Algorithm

- To exploit the Betweenness of edges, we need to calculate the number of shortest paths going through each edge.
- A method called the Girvan-Newman (GN) Algorithm, which visits each node X once and computes the number of shortest paths from X to each of the other nodes that go through each of the edges.
- The algorithm begins by performing a breadth-first search (BFS) of the graph, starting at the node X.
- The level of each node in the BFS presentation is the length of the shortest path from X to that node.
- The edges that go between nodes at the same level can never be part of a shortest path from X.

11.3 Direct Discovery of Communities

It is not possible to place an individual in two different communities, and everyone is assigned to a community. A technique for discovering communities directly by looking for subsets of the nodes that have a relatively large number of edges among them.

A. Finding Cliques

- To find sets of nodes with many edges between them is to start by finding a large clique (a set of nodes with edges between any two of them).
- For finding maximal cliques NP-complete, but it is among the hardest of the NP-complete problems in the sense that even approximating the maximal clique is hard.
- It is possible to have a set of nodes with almost all edges between them, and yet have only relatively small cliques.

B. Complete Bipartite Graphs

- A complete bipartite graph consists of s nodes on one side and t nodes on the other side, with all st possible edges between the nodes of one side and the other present. We denote this graph by $K_{s,t}$.
- Draw an analogy between complete bipartite graphs as subgraphs of general bipartite graphs and cliques as subgraphs of general graphs.
- A clique of s nodes is often referred to as a complete graph and denoted K_s, while a complete bipartite subgraph is sometimes called a bi-clique.

C. Finding Complete Bipartite Subgraphs

- Taking a large bipartite graph G , and to find instances of $K_{s,t}$ within it. It is possible to view the problem of finding instances of $K_{s,t}$ within G as one of finding frequent itemsets.
- For this purpose, let the “items” be the nodes on one side of G, which we shall call the left side. We assume that the instance of $K_{s,t}$ we are looking for has t nodes on the left side, and to assume for efficiency that $t \le s$.
- The “baskets” correspond to the nodes on the other side of G (the right side).

- The members of the basket for node v are the nodes of the left side to which v is connected.

11.4 SimRank

i. "SimRank," applies best to graphs with several types of nodes, although it can in principle be applied to any graph.

ii. The purpose of SimRank is to measure the similarity between nodes of the same type, and it does so by seeing where random walkers on the graph wind up when starting at a particular node.

iii. Because calculation must be carried out once for each starting node, it is limited in the sizes of graphs that can be analyzed completely

A. Random Walkers on a Social Graph

- The graph of a social network is generally undirected, while the Web graph is directed. However, the difference is unimportant.
- A walker at a node N of an undirected graph will move with equal probability to any of the neighbors of N (those nodes with which N shares an edge).
- Example, that such a walker starts out at node T1 as shown below.

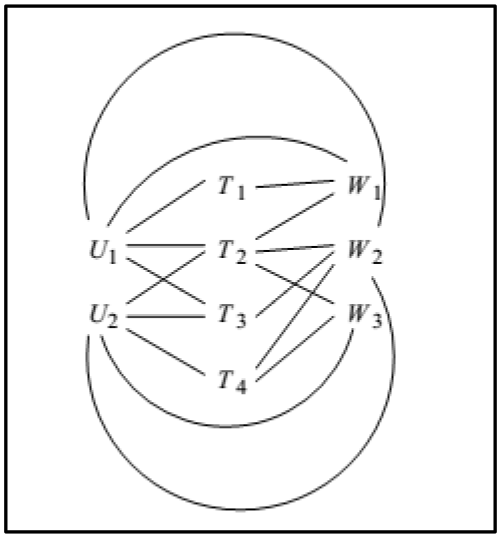


Figure 11.2: Walker

- At the first step, it would go either to U1 or W1. If to W1, then it would next either come back to T1 or go to T2.
- If the walker first moved to U1, it would wind up at either T1, T2, or T3 next.
- Starting at T1, there is a good chance the walker would visit T2, at least initially, and that chance is better than the chance it would visit T3 or T4.
- It would be interesting if we could infer that tags T1 and T2 are therefore related or similar in some way. The evidence is that they have both been placed on a common Web page, W1, and they have also been used by a common tagger, U1.

B. Random Walks with Restart

- To be focused on one particular node N of a social network, and want to see where the random walker winds up on short walks from that node, we modify the matrix of transition probabilities to have a small additional probability of transitioning to N from any node.
- Formally, let M be the transition matrix of the graph G.
- That is, the entry in row i and column j of M is 1/k if node j of G has degree k, and one of the adjacent nodes is i. Otherwise, this entry is 0.

11.5 Counting triangles using MapReduce

i. For a very large graph, we want to use parallelism to speed the computation.
ii. To optimize the use of a single MapReduce job to count triangles.
iii. It turns out that this use is one where the multiway join technique of that section is generally much more efficient than taking two two-way joins.
iv. To begin, assume that the nodes of a graph are numbered 1, 2, . . . , n.
v. Use a relation E to represent edges. To avoid representing each edge twice, we assume that if E(A, B) is a tuple of this relation, then not only is there an edge between nodes A and B, but also, as integers, we have A < B.

vi. This requirement also eliminates loops (edges from a node to itself), which we generally assume do not exist in social-network graphs anyway, but which could lead to “triangles” that really do not involve three different nodes.