BITS Pilani
Pilani | Dubai | Goa | Hyderabad

DSECL ZG 522: Big Data Systems
Session 3 – Fault Tolerance, BDA and Systems

Janardhanan PS
janardhanan.ps@wilp.bits-pilani.ac.in

Topics for today

- **Distributed Computing – Availability and Reliability**
- Analytics
  - Definitions
  - Maturity model
  - Types
- Big Data Analytics
  - Characterization
  - Adoption challenges
  - Requirements
  - Technology challenges
- Popular technologies - Hadoop, Spark

Will help you to pitch a Big Data project proposal

Self Reading

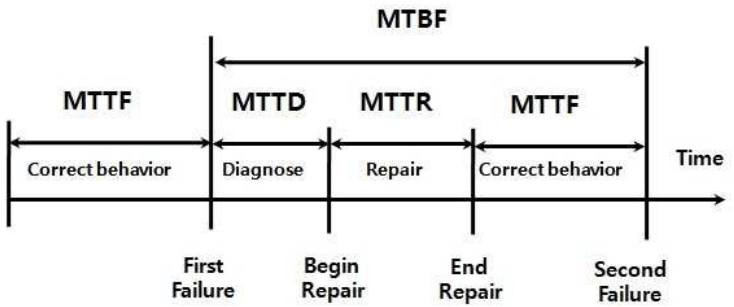
Will help you to build the system

Distributed computing – Living with failures

- Failures of nodes and links are common concern in Distributed Systems
  - Essential to incorporate fault tolerance aspects in design of distributed systems
- Fault Tolerance or Graceful Degradation (in performance) is the property that enables a system to continue operating properly in the event of failure of some of its components
- Fault-Tolerant Systems - Ideally systems capable of executing their tasks correctly regardless of either HW failures or SW errors
- Fault tolerance of a distributed system is a measure of
  - How a distributed system functions in the presence of failures of its system components
  - Tolerance of component faults is measured by 2 parameters
    - Reliability - An inverse indicator of failure rate
      - How soon a system will fail
    - Availability - An indicator of fraction of time a system is available for use
      - System is not available during failure

Metrics

- MTTF - Mean Time To Failure. Is an averaged value
- MTTR - Mean Time to Recovery / Repair
- MTTR = Total #hours for maintenance / Total #repairs
- MTTD - Mean Time to Diagnose
- MTBF - Mean Time Between Failures
  - MTBF = MTTD + MTTR + MTTF
- Failure rate = 1 / MTBF (assuming average value over time)



https://www.epsilonengineer.com/mtbf-mttr-and-reliability.html

Availability

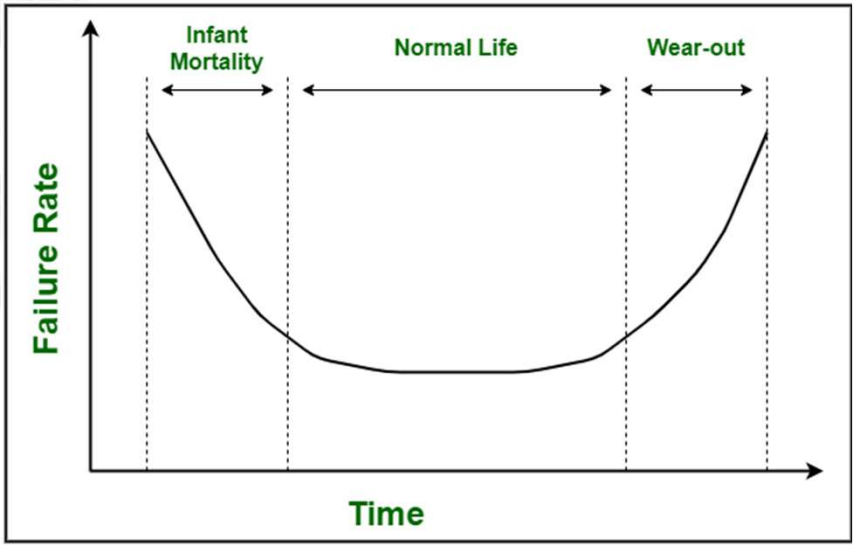
- In reality, failure rate changes over life-time of a system because it may depend on age of component.
- Availability = Time system is UP and accessible / Total time observed
- Availability = MTTF / (MTTD* + MTTR + MTTF)

  But MTBF = MTTD + MTTR + MTTF

  or

- Availability = MTTF / MTBF
- A system is highly available when
  - MTTF is high
  - MTTR is low

* Unless specified one can assume MTTD = 0

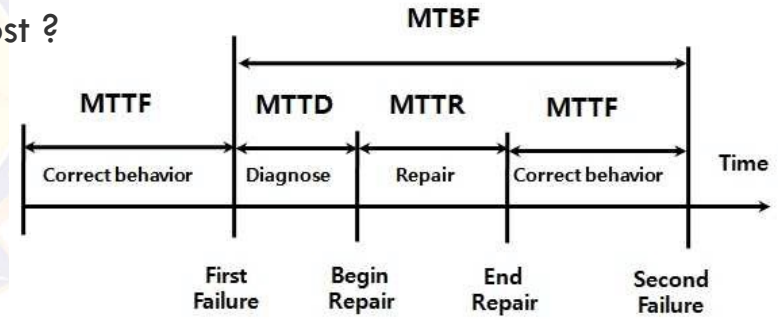


Bathtub Curve

http://www.eventhelix.com/RealtimeMantra/FaultHandling/reliability_availability_basics.htm

Example

- A node in a cluster fails every 100 hours while other parts never fail.
- On failure of the node the whole system needs to be shutdown, faulty node replaced and system has to be restarted. This takes 2 hours.
- The application needs to be restarted, which takes 2 hours.
- What is the availability of the cluster ?
- If downtime is $80k per hour, then what is the yearly cost ?
- Solution
  - MTTF = 100 hours
  - MTTR = 2 + 2 = 4 hours
  - MTBF = MTTR+MTTF = 104 hours
  - Availability = MTTF/MTBF = 100/104 = 96.15%
  - Cost of downtime per year = 80000 x (100-96.15)* 365 * 24 / 100 = USD 27 million



Little Maths

- **Availability**
  - Availability of the module is the percentage of time when system is operational.

$$A = (1 - (MTTR/MTBF)) \times 100\%.$$

  - Availability is typically specified in nines notation. For example 3-nines availability corresponds to 99.9% availability. A 5-nines availability corresponds to 99.999% availability.
- **Downtime**
  - Downtime per year is a more intuitive way of understanding the availability.

    The table below compares the availability and the corresponding downtime.

Availability	**Downtime**	
90% (1-nine)	36.5 days/year	
99% (2-nines)	3.65 days/year	
99.9% (3-nines)	8.76 hours/year	
99.99% (4-nines)	52 minutes/year	Business Critical
99.999% (5-nines)	5 minutes/year	Mission Critical (Considered as HA)
99.9999% (6-nines)	31 seconds/year	Fault Tolerant (no restart)

(availabilitydigest.com)

The Nines of Availability

bmc

The Nines of Availability

Availability percentages vs service downtime

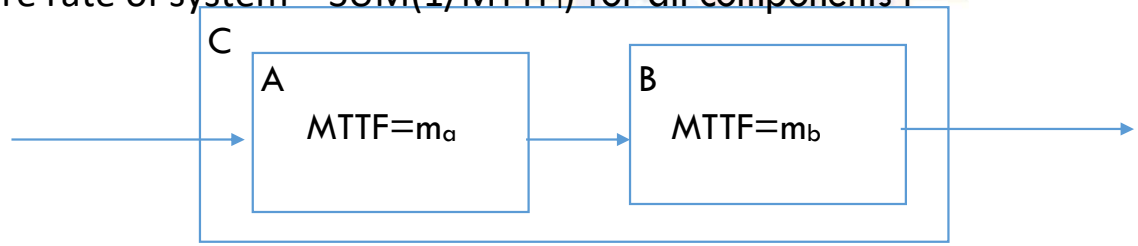
Availability %	Downtime per year	Downtime per month	Downtime per week
90% (one nine)	36.5 days	72 hours	16.8 hours
99% (two nines)	3.65 days	7.20 hours	1.68 hours
99.5%	1.83 days	3.60 hours	50.4 minutes
99.9% (three nines)	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% (four nines)	52.56 minutes	4.32 minutes	1.01 minutes
99.999% (five nines)	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% (six nines)	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% (seven nines)	3.15 seconds	0.259 seconds	0.0605 seconds

The 9s Game – Different platforms

Amazon S3	99.999999999 ? (11 nines)
NonStop Integrity	99.9999
Mainframe	99.999
OpenVMS	99.998
AS400	99.998
HPUX	99.996
Tru64	99.996
Solaris	99.995
NT Cluster	99.992 - 99.995

Reliability - Serial assembly

- MTTF of a system as a function of MTTF of components connected serially
- Failure rate = 1 / MTBF (MTBF = MTTD + MTTR + **MTTF**)
- Serial assembly of components
  - Failure of any component results in system failure
  - m_a = MTTF of system A, m_b = MTTF of system B
  - Failure rate of C = Failure rate of A + Failure rate of B = $1/m_a + 1/m_b$
  - MTTF of system = 1 / SUM ($1/MTTF_i$) for all components i
  - Failure rate of system = SUM($1/MTTF_i$) for all components i



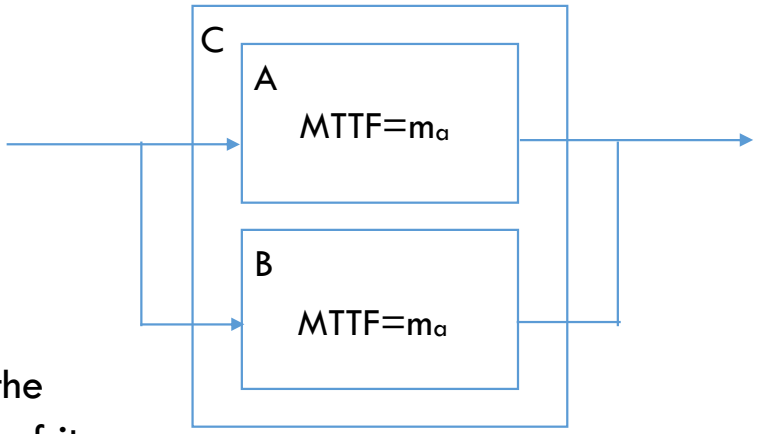
Failure rate of C $m_c = 1/(1/m_a + 1/m_b)$

- The combined availability of system C is:

$A_c = A_a A_b$, Where A_a is the availability of component A and A_b is the availability of component B

Reliability - Parallel assembly

- MTTF of a system as a function of MTTF of components connected in parallel
- In a parallel assembly, e.g. a cluster of nodes
  - MTTF of C = MTTF A + MTTF B because both A and B have to fail for C to fail
- MTTF of system = SUM($MTTF_i$) for all components i



MTTF $m_c = m_a + m_b$

Availability - If two components are connected in parallel, then the availability of the whole will always be higher than the availability of its individual components

Combined Availability $A_c = 1 - ((1-A_a) \times (1-A_b))$

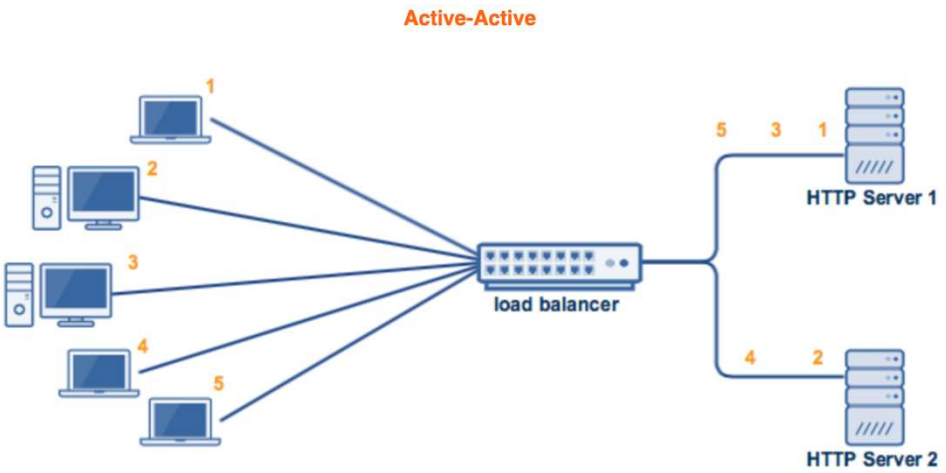
Example:

If components A and B each has an availability of 99.75%

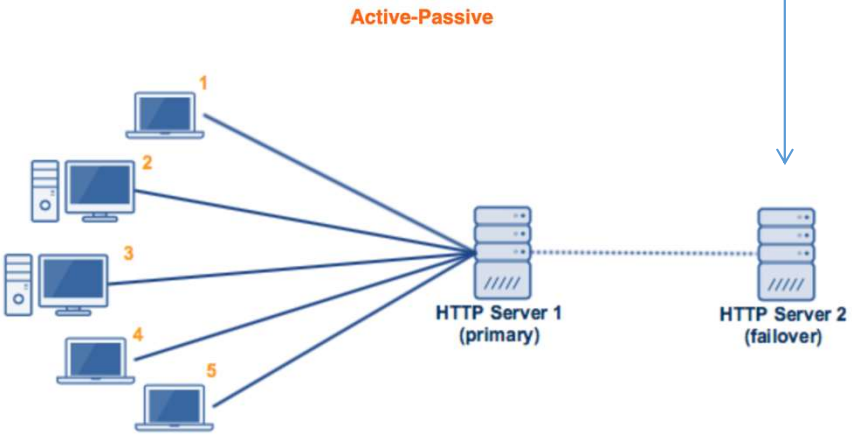
The Parallel combination of both A and B will have an availability of 99.999375%

This value is calculated by multiplying the unavailability of both components.

Fault tolerance configurations - standby options



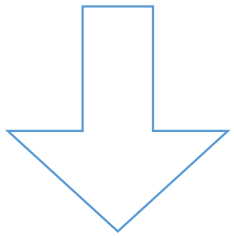
Warm and cold depends on how the passive / failover node is kept updated



Fault tolerance configurations - standby options

High availability model	Secondary node behavior	Data protection	Failover time
Load-balanced	Both the primary node and the secondary node are active and they process system requests in parallel. aka active-active	Data replication is bidirectional and is performed based on the software capabilities.	Zero failover time
Hot standby	The software component is installed and available on both the primary node and the secondary node. The secondary system is up and running, but it does not process data until the primary node fails. aka active-passive	Data is replicated and both systems contain identical data. Data replication is performed based on the software capabilities.	A few seconds
Warm standby	The software component is installed and available on the secondary server, which is up and running. If a failure occurs on the primary node, the software components are started on the secondary node. This process is automated by using a cluster manager.	Data is regularly replicated to the secondary system or stored on a shared disk.	A few minutes
Cold standby	A secondary node acts as the backup for an identical primary system. The secondary node is installed and configured only when the primary node breaks down for the first time. Later, in the event of a primary node failure, the secondary node is powered on and the data is restored while the failed component is restarted.	Data from a primary system can be backed up on a storage system and restored on a secondary system when it is required.	A few hours

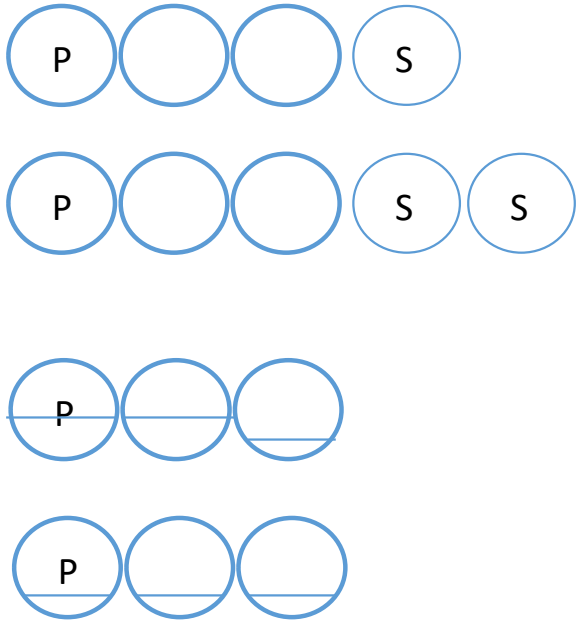
Decreasing cost
vs
Increasing MTTR



Assumption:
Same software bug or runtime fault will not recur in the standby

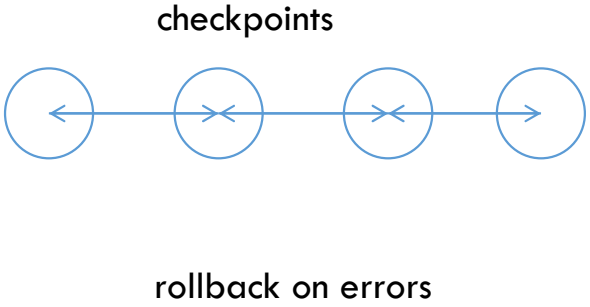
Fault tolerance configurations - cluster topologies

- N+**1**
  - One node is configured to take the role of the primary
- N+**M**
  - M standby nodes if multiple failures are anticipated especially when running multiple services in the cluster
- N to 1
  - The secondary failover node is a temporary replacement and once primary is restored, services must be reactivated on it
- N to **N**
  - When any node fails, the workload is redistributed to the remaining active nodes. So, there is no special standby node.



Fault Tolerant Clusters – Recovery

- Diagnosis
  - Detection of failure and location of the failed component, e.g. using heartbeat messages between nodes
- Backward (Rollback) recovery - Checkpoints
  - Fault tolerance is achieved by periodically saving the processes' consistent states during the failure-free execution using stable storage
  - Each of the saved state is called a checkpoint
  - On failure, isolate the failed component, rollback to last checkpoint and resume normal operation
  - Ease to implement, independent of application, but leads to wastage of execution time on rollback besides unused checkpointing work
- Forward recovery
  - Finding a new state from which the system can continue operation
  - In real-time systems or time-critical systems cannot rollback. So state is reconstructed on the fly from diagnosis data.
  - Application specific and may need additional hardware



Topics for today

- Distributed Computing – Reliability and Availability
- **Analytics**
  - **Definitions**
  - **Maturity model**
  - **Types**
- Big Data Analytics
  - Characterization
  - Adoption challenges
  - Requirements
  - Technology challenges
- Popular technologies - Hadoop, Spark

Will help you to pitch a Big Data project proposal

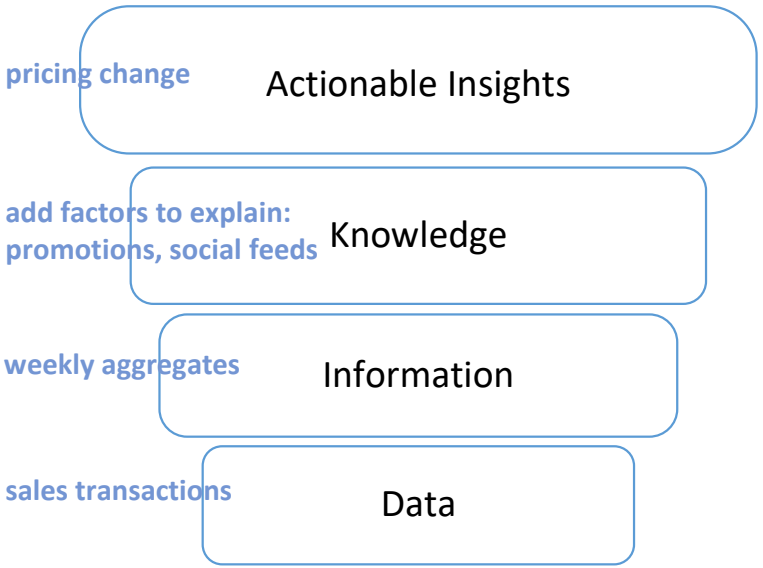
Will help you to build the system

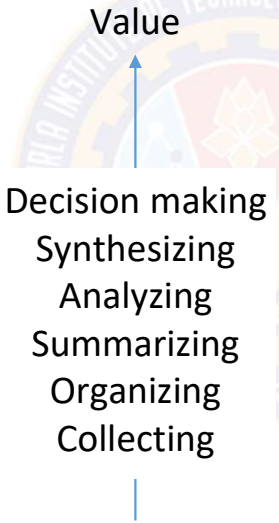
Analytics - Definitions

- Extensive use of data, statistical and quantitative analysis, explanatory and predictive models, and fact based management to drive decisions and actions
- Purpose
  - ✓ To unearth hidden patterns
    - ✓ 2 / 100 stores had no sales for a promotion item because it was not in the right shelf
  - ✓ To decipher unknown correlations
    - ✓ The famous "Beer and diapers" story
  - ✓ Understand the rationale behind trends
    - ✓ What do users like about a popular product that has growing sales
  - ✓ Mine useful business information
    - ✓ Popular items during specific holiday sales
- Helps in
  - ✓ Effective marketing
  - ✓ Better customer service and satisfaction
  - ✓ Improved operational efficiency
  - ✓ Competitive advantage over rivals

Process of Analysis

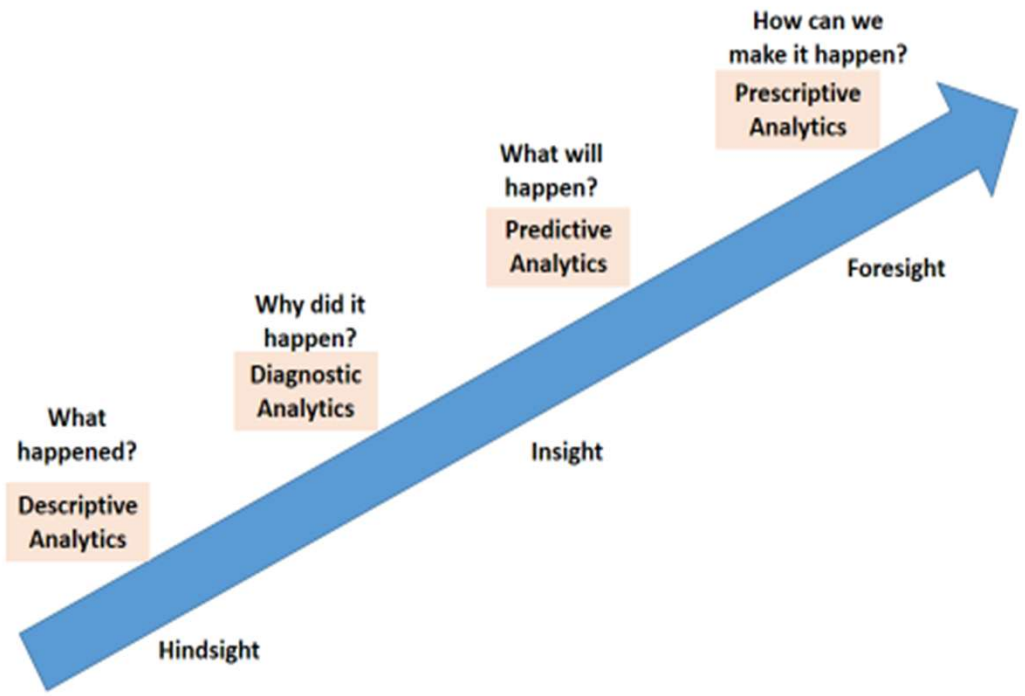
Transformation of Data





- Apply functions / transformations on data to get to the next level till it is actionable insight useful for the business
- Keep attaching more meta-data for context and make it meaningful

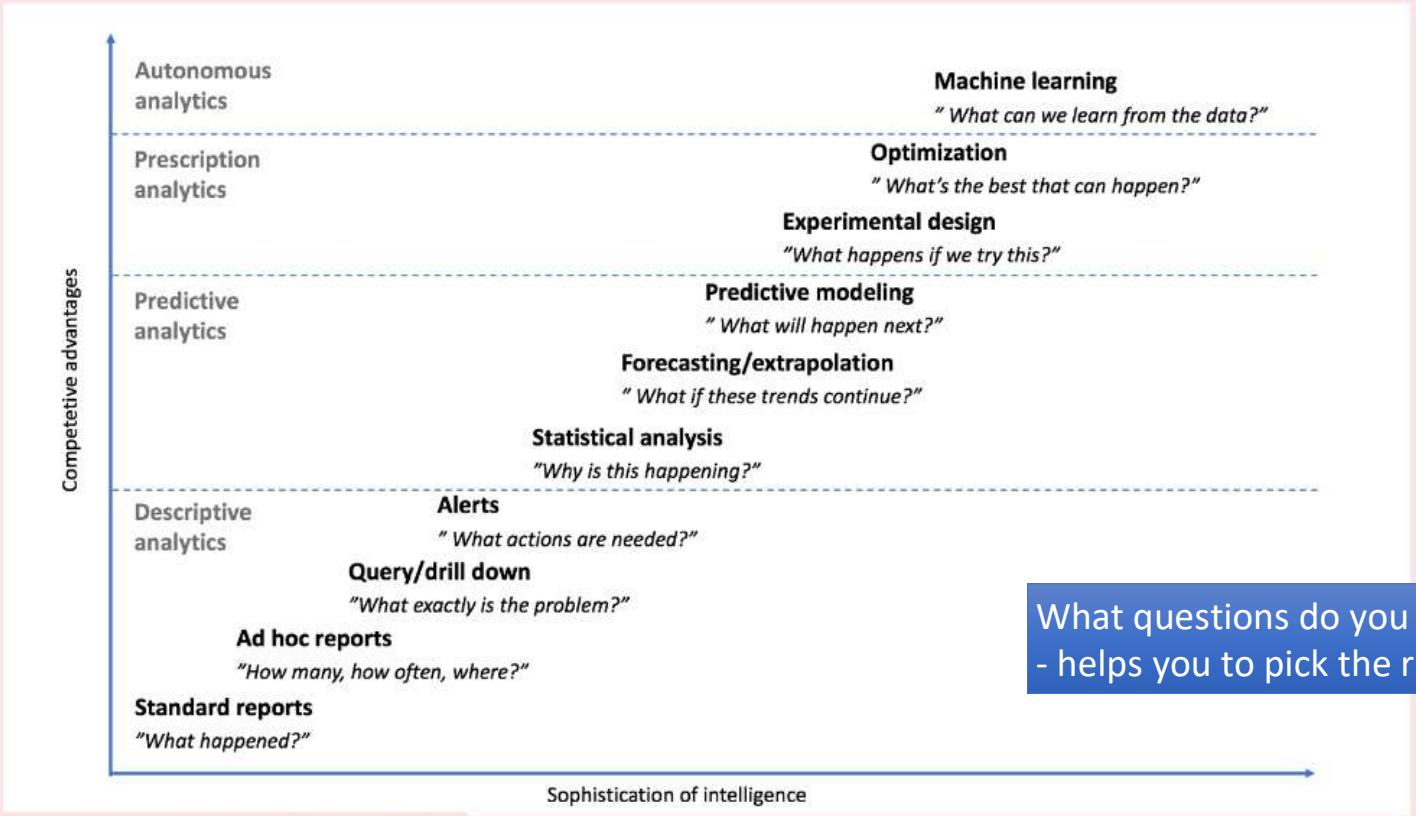
Analytics 1.0, 2.0, 3.0



Analytics 1.0 → deals with historical data to report on events, occurrences of the past.
Analytics 2.0 → helps to predict what will happen in future
Analytics 3.0 → is about predicting what will happen and to best leverage the situation when it happens.

Analytics Maturity and Competitive Advantage

Analytics Maturity & Competitive Advantage



What questions do you want to answer - helps you to pick the right technology

source: Competing on Analytics, Thomas Davenport

Types of Analytics - Descriptive

- Provides ability to alert, explore and report using mostly internal and external data
- Business Intelligence (BI) or Performance reporting
- Provides access to historical and current data
- Reports on events, occurrences of the past
- Usually data from legacy systems, ERP, CRM used for analysis
- **Based on relational databases and warehouse**
- Structured and not very large data sets
- Sometimes also referred as Analytics 1.0
- Era : mid 1950s to 2009

- Questions asked
  - ✓ What happened?
  - ✓ Why did it happen? (Diagnostic analysis)

E.g. number of infections is significantly higher this month than last month and highly correlated with factor X

Types of Analytics - Predictive

- Uses past data to predict the future
- Uses quantitative techniques like segmentation, forecasting etc. but also makes use of descriptive analytics for data exploration
- Uses technologies like models and rule based systems
- Based on large data set gathered over period of time
- Externally sourced data also used
- Unstructured data may be included
- **Hadoop clusters, SQL on Hadoop data etc. technologies used**
- aka Analytics 2.0
- Era : from 2005 to 2012

- Key questions
  - ✓ What will happen?
  - ✓ Why it will happen?
  - ✓ When will it happen?

E.g. number of infections will reach X in month Y and likely cause will be Z

Types of Analytics - Prescriptive

- Uses data from past to make prophecies of future and at the same time make recommendations to leverage the situation to one's advantage
- Suggests optimal behaviors and actions
- Uses a variety of quantitative techniques like optimization and technologies like models, machine learning and recommendations engines
- Data is blend from Big data and legacy systems, ERP, CRM etc.
- In-memory analysis etc.
- Aka Analytics 3.0 = Descriptive + Predictive + Prescriptive
- post 2012
- Questions:
  - ✓ What will happen
  - ✓ When will it happen
  - ✓ Why will it happen
  - ✓ What actions should be taken

E.g. number of infections will reach X in month Y and likely cause will be Z. W is the best recommended action to keep the number in month Y below X/2.

Alternative categorization of Analytics

- Basic Analytics
  - ✓Slicing and dicing of data to help with basic insights
  - ✓Reporting on historical data, basic visualizations etc.
- Operationalized Analytics
  - ✓Analytics integrated in business processes
- Advanced Analytics
  - ✓Forecasting the future by predictive modelling
- Monetized Analytics
  - ✓Used for direct business revenue

Topics for today

- Distributed Computing – Reliability and Availability
- Analytics
  - Definitions
  - Maturity model
  - Types
- **Big Data Analytics**
  - **Characterization**
  - **Adoption challenges**
  - **Requirements**
  - **Technology challenges**
- Popular technologies - Hadoop, Spark

Will help you to pitch a Big Data project proposal

Will help you to build the system

Big Data Analytics

Working with datasets with huge volume, variety and velocity beyond storage and processing capability of RDBMS

Better , Faster decision in real time

Richer, faster insights into customers, partners and business

Uses Principle Of Locality to move code near to Data

Big Data Analytics

Competitive Advantage

IT's collaboration with business users and Data Scientists

Technology enabled Analytics

Support for both batch and stream processing of data

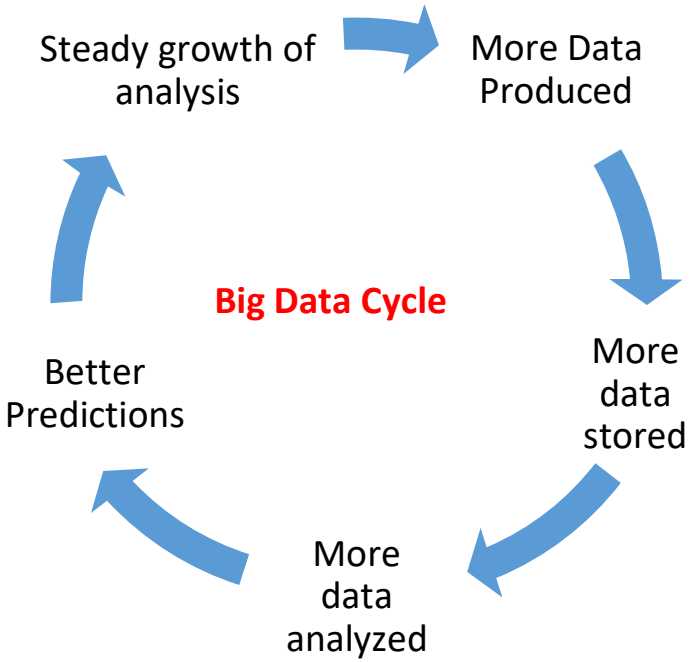
What makes you think about this differently ?

What Big Data Analytics is NOT

Things to know to avoid friction in Big Data projects

Why the sudden hype ?

- Data is growing at 40% compound annual rate
  - ✓ 45 ZB in 2020
  - ✓ In 2010, 1.2 trillion Gigabytes data generated
  - ✓ In 2012, reached to 2.4 trillion Gigabytes
  - ✓ Volume of world-wide data expected to double every 1.2 years
  - ✓ Every day 2.5 quintillion bytes of data is created
  - ✓ 90% of today's data is generated in last few years only!
  - ✓ Walmart processes one million customer transaction per hour
  - ✓ 500 million "tweets" are posted by users every day
  - ✓ 2.7 billion "likes" and comments by Facebooks users per day
- Cost of storage has hugely dropped
- Large number of user friendly analytics tools available for data processing



Adoption Challenges in Organizations

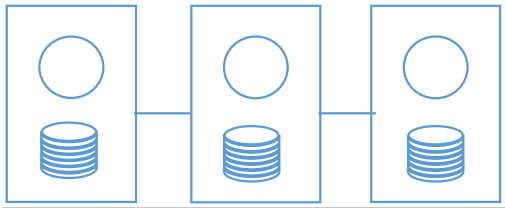
- Obtaining executive sponsorship for investments in big data and its related activities
- Getting business units to share data / information across organizational silos
- Finding right skills (Business Analysts/Data Scientists and Data Engineers) that can manage large amount of variety of data and create insights from it
- Determining approach to scale rapidly and elastically , address storage and processing of large volume, velocity and variety of Big data
- Deciding whether to use structured or unstructured, internal or external data to make business decisions
- Choosing optimal way to report findings and analysis of big data
- Determining what to do with the insights created from big data

Requirements of Big Data analytics

- Cheap abundant storage
- Processing options
  - batch / streaming,
  - disk based / memory based
- **Open source platforms, e.g. Hadoop, Spark**
- Parallel and distributed systems with high throughput rather than low latency
- Cloud or other flexible resource allocation arrangements
  - Flexibility to setup and tear down infrastructure for quick projects across various teams

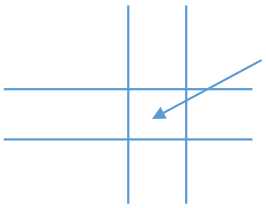
Technology challenges (1)

- Scale
  - ✓ Need is to have storage that can best withstand large volume, velocity and variety of data
  - ✓ Scale vertically / horizontally ?
  - ✓ RDBMS / NoSQL ?
  - ✓ How does compute scale with storage - coupled or de-coupled, i.e. good idea to put common nodes for compute and storage



compute + data on same node: locality helps, but does compute scale with storage ?

- Security *
  - ✓ Most of recent NoSQL big data platforms have poor security mechanisms, e.g. challenges:
    - ✓ Fine grain control in semi-structured data, esp with columnar storage
    - ✓ Options for inconsistent data complicate matters
    - ✓ Larger attack surface across distributed nodes
    - ✓ Often encryption is turned off for performance
  - ✓ Lack of authorization techniques while safeguarding big data
    - ✓ May contain PII data (personally identifiable info)



Easier in RDBMS to control at row / column / cell level with always consistent values in a tightly coupled system

* Ref: NoSQL security

Technology challenges (2)

- Schema
  - ✓ Need is to have dynamic schema, static / fixed schemas don't fit
- Continuous availability
  - ✓ Needs 24 * 7 * 365 support as data is continuously getting generated and needs to be processed
  - ✓ Almost all RDBMS, NoSQL big data platforms has some sort of downtime
    - ✓ Memory cleanup, replica rebalancing, indexing, …
    - ✓ Most of the large-scale NoSQL systems also need weekly maintenance

Technology challenges (3)

- Consistency
  - ✓ Should one go for strict consistency or eventual consistency? Is this like social media comments or application needs consistent reads ?
- Partition Tolerant
  - ✓ When a system get's partitioned by hardware / software failures. How to build partition tolerant systems ? When faults happen is consistent data available ?
  - ✓ We will discuss options in CAP Theorem
- Data quality
  - ✓ How to maintain data quality – data accuracy, completeness, timeliness etc.?
  - ✓ Do we have appropriate metadata in place esp with semi/un-structured data ?

Popular technologies

- How to manage voluminous, varied, scattered and high velocity data ?
  - ✓ Think beyond an RDBMS depending on use case but not necessarily to replace it

- Some popular technologies
  - ✓NoSQL databases (covered in session 4 – 5)
  - ✓Distributed and parallel processing (covered in session 6)
  - ✓ Hadoop (more details in session 7-10)
    - ✓ File based large scale parallel data processing tasks
  - ✓ In-memory computing (more details in session 11-14)
    - ✓ Usage of main memory (RAM) helps to manage data processing tasks faster
  - ✓ Big Data Cloud (more details in session 15-16)
    - ✓ Helps to save cost and better management of resources using a services model

Topics for today

- Distributed Computing – Reliability and Availability
- Analytics
  - Definitions
  - Maturity model
  - Types
- Big Data Analytics
  - Characterization
  - Adoption challenges
  - Requirements
  - Technology challenges
- **Popular technologies - Hadoop, Spark**

Will help you to pitch a Big Data project proposal

Will help you to build the system

Common Bigdata Platforms and their users

Introduction to Hadoop

What problems does Hadoop solve

Storage of huge amount of data

✓Problems with partitioning
- ✓ Multiple partitions of data for parallel access but more systems means more failures
- ✓ Multiple nodes can make the system expensive
- ✓ Arbitrary data - binary, structured …

✓ Solution
- ✓ Replication Factor (RF) for failures : Number of data copies of a given data item / data block stored across the network
- ✓ Uses commodity heterogenous hardware
- ✓ Multiple file formats

Processing the huge amount of data

✓ Problems
- ✓ Data is spread across systems, how to process it in quick manner?
- ✓ Challenge is to integrate data from different machines after processing

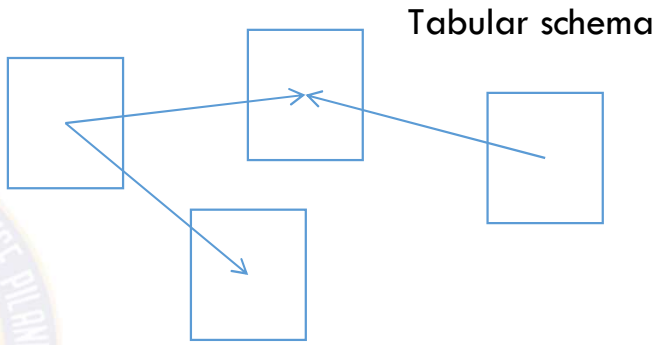
✓ Solution
- ✓ MapReduce programming model to process huge amount of data with high throughput
- ✓ Compute is close to storage for handling large data sets

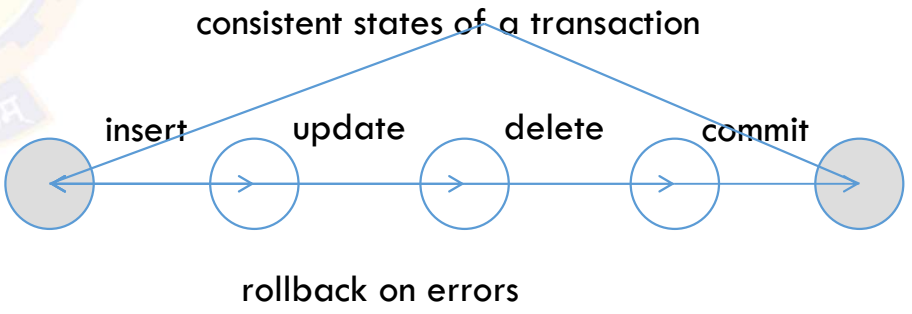
Basic Design Principles of Hadoop

1. *All roads lead to scale-out:* scale-up architectures are rarely used and scale-out is the standard in big data processing.
2. *Share nothing*: communication and dependencies are bottlenecks, individual components should be as independent as possible to allow to proceed regardless of whether others fail.
3. *Expect failure*: components will fail at inconvenient times.
4. *Smart software, dumb hardware*: push smartness in the software, responsible for allocating generic hardware.
5. *Move processing, not data*: perform processing locally on data. What gets moved through the network are program binaries and status reports, which are dwarfed in size by the actual data set.
6. *Build applications, not infrastructure*. Instead of placing focus on data movement and processing, work on job scheduling, error handling, and coordination.

What's different from a Distributed Database

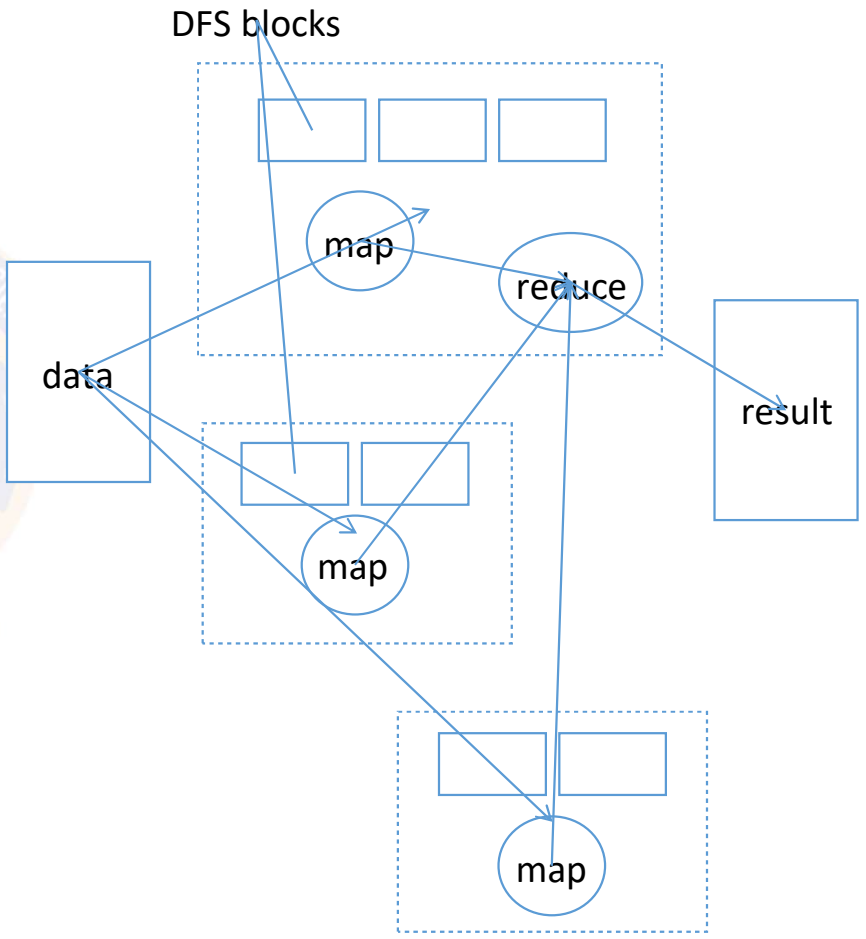
- Distributed Databases
  - Data model
    - Tables and relations
    - Schema is predefined (during write)
    - Supports partitioning
    - Fast indexed reads
    - Read and write many times
  - Compute model
    - Generate notations of a transaction
    - ACID properties
    - Allow distributed transactions
    - OLTP workloads



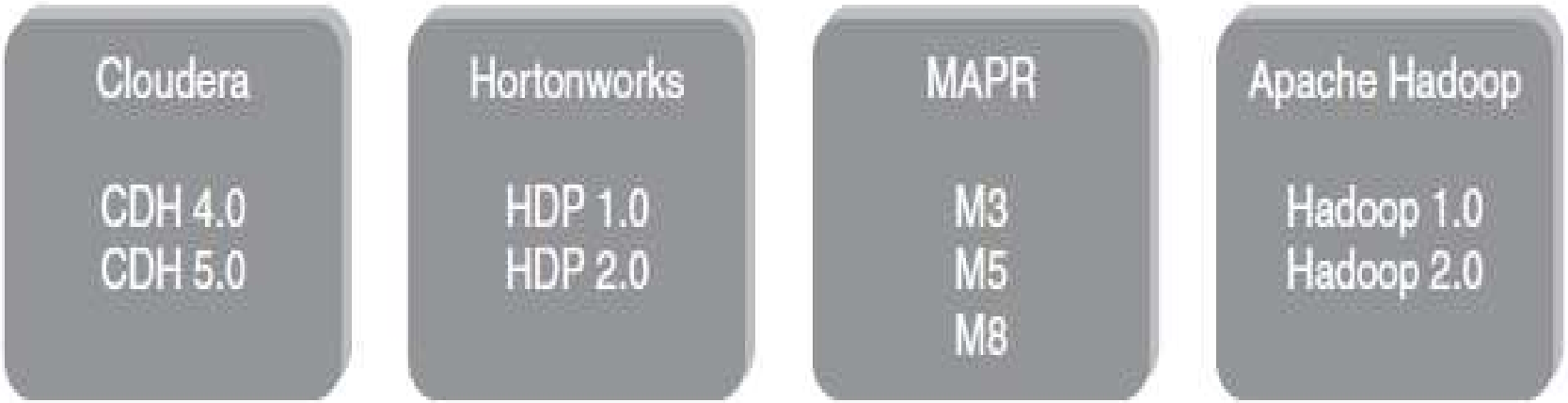


Hadoop in contrast …

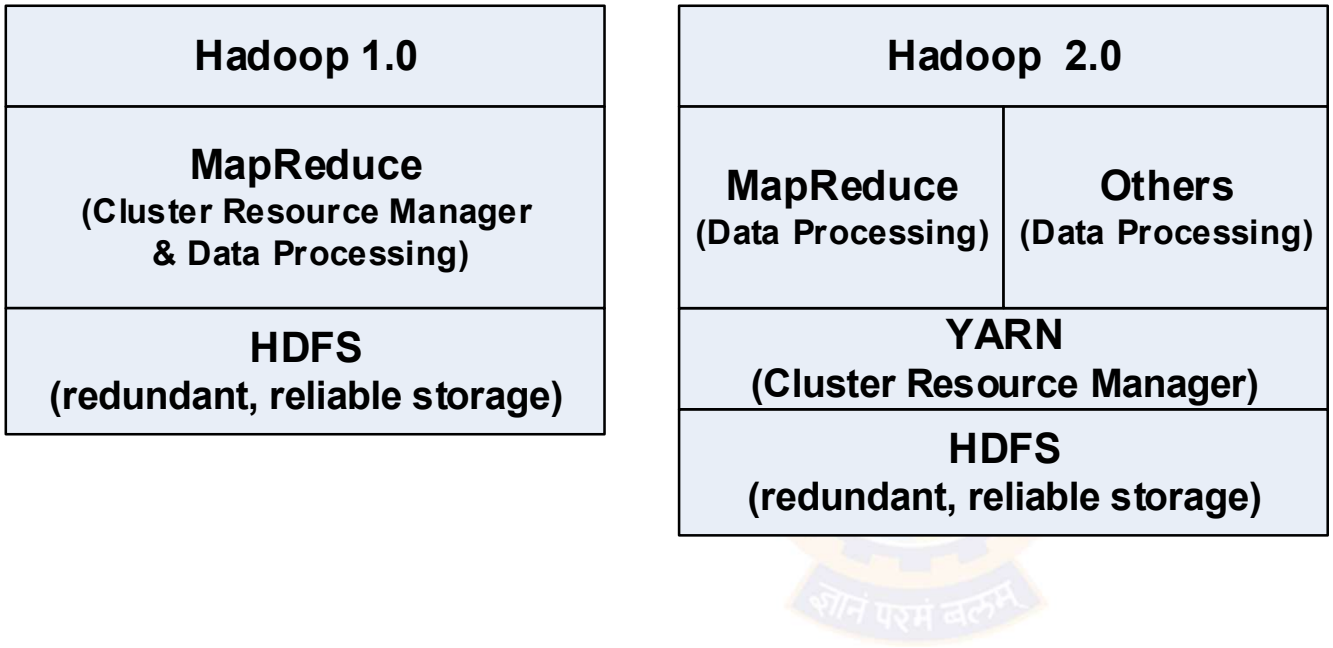
- Data model
  - Flat files supporting multiple formats, including binary
  - No pre-defined schema (i.e. during write)
  - Divides files automatically into blocks
  - Handles large files
  - Optimized for write
  - Write once and read many times (WORM) workload
  - Meant for scan workloads with high throughput
- Compute model
  - Generate notations of a job divided into tasks
  - MapReduce compute model
  - Every task is a map or a reduce
  - High latency analytics, data discovery workloads



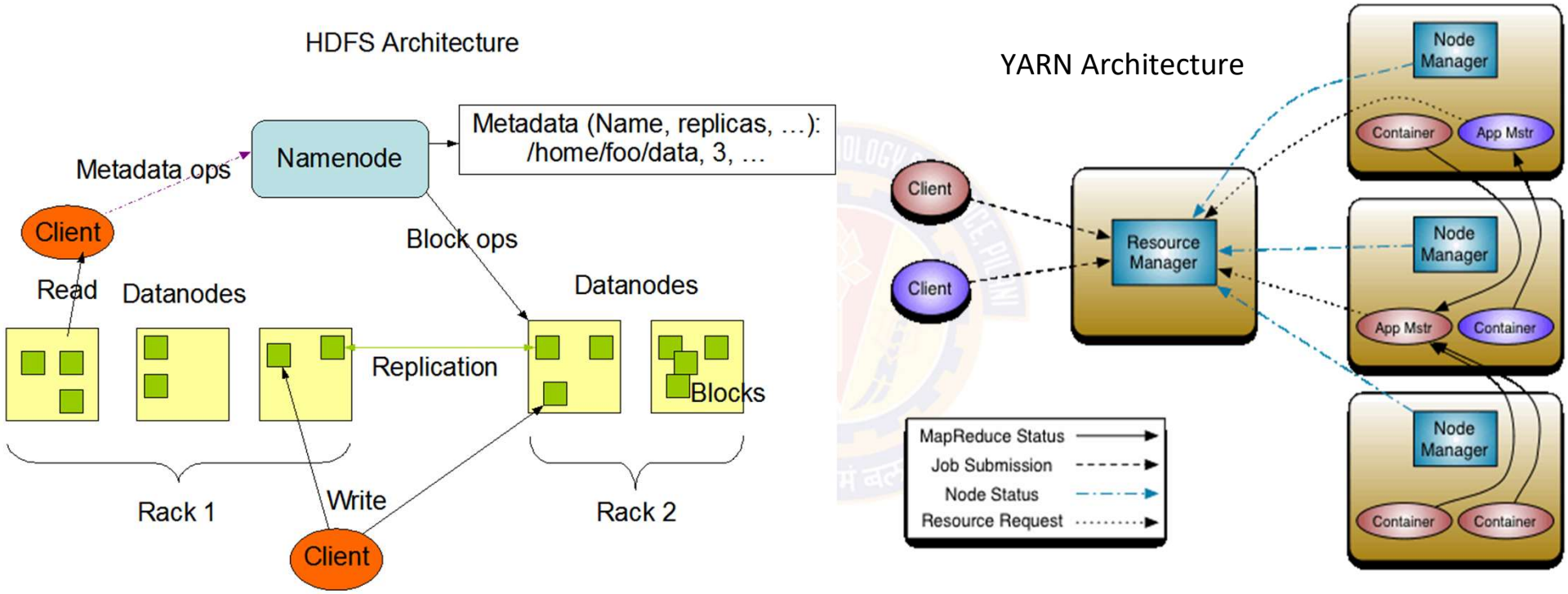
Hadoop Distributions



Versions of Hadoop



Hadoop 2.0 high level architecture



Advantages of using Hadoop

- Low cost – open source and low-cost commodity storage
- Computing power – many nodes can be used for computation
- Scalability – simple to add nodes in system for parallel processing and storage
- Storage Flexibility – can store unstructured data easily
- Inherent data protection – protects against hardware failures

Hadoop ecosystem

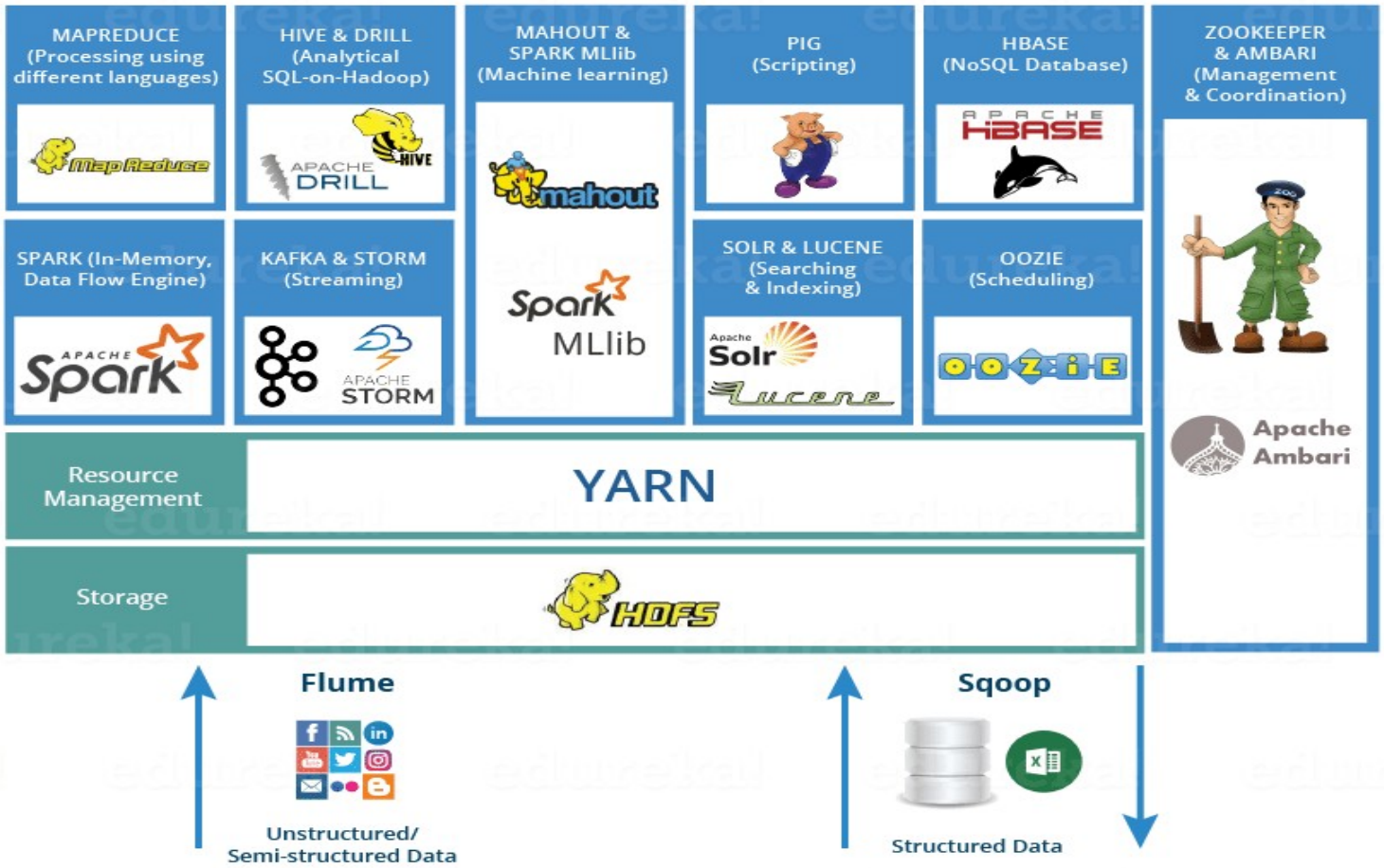


Image Source : [Edureka]

Hadoop Ecosystem Components

Components that help with Data Ingestion are:

1. Sqoop
2. Flume

Components that help with Data Processing are:

1. MapReduce
2. Spark

Components that help with Data Analysis are:

1. Pig
2. Hive
3. Impala

Hadoop Ecosystem Components for Data Ingestion

Sqoop:

- Sqoop stands for SQL to Hadoop. It can provision the data from external RDBMS system on to HDFS and populate tables in Hive and HBase.

Flume:

- Flume is an important log aggregator (aggregates logs from different machines and places them in HDFS) component in the Hadoop Ecosystem.

Hadoop Ecosystem Components for Data Processing

MapReduce:

- It is a programing paradigm that allows distributed and parallel processing of huge datasets. It is based on Google MapReduce.

Spark:

- It is both a programming model as well as a computing model. It is an open source big data processing framework.
- It is written in Scala. It provides in-memory computing for Hadoop.
- Spark can be used with Hadoop coexisting smoothly with MapReduce (sitting on top of Hadoop YARN) or used independently of Hadoop (standalone).

Hadoop ecosystem components for Data Analysis

Pig

- It is a high level scripting language used with Hadoop. It serves as an alternative to MapReduce. It has two parts:
- Pig Latin: It is a SQL like scripting language.
- Pig runtime: is the runtime environment.

Hive:

- Hive is a data warehouse software project built on top of Hadoop. Three main tasks performed by Hive are summarization, querying and analysis

Impala:

- It is a high performance SQL engine that runs on Hadoop cluster. It is ideal for interactive analysis. It has very low latency measured in milliseconds. It supports a dialect of SQL called Impala SQL.

RDBMS Vs Hadoop

PARAMETERS	RDBMS	HADOOP
System	Relational Database Management System.	Node Based Flat Structure.
Data	Suitable for structured data.	Suitable for structured, unstructured data. Supports variety of data formats in real time such as XML, JSON, text based flat file formats, etc.
Processing	OLTP	Analytical, Big Data Processing
Choice	When the data needs consistent relationship.	Big Data processing, which does not require any consistent relationships between data.
Processor	Needs expensive hardware or high-end processors to store huge volumes of data.	In a Hadoop Cluster, a node requires only a processor, a network card, and few hard drives.
Cost	Cost around $10,000 to $14,000 per terabytes of storage.	Cost around $4,000 per terabytes of storage.

Hadoop – Different modes of operation

- **Standalone Mode**
  - Runs on single system on single JVM (Java Virtual Machine), called Local mode
  - None of the Daemon will run
  - Resource Manager and Node Manager is available for running jobs.
  - Mainly used Hadoop in this Mode for the Purpose of Learning, testing, and debugging.
- **Pseudo-distributed Mode** (Demo cluster Setup)
  - Uses only single node
  - All the daemons: Namenode, Datanode, Secondary Name node, Resource Manager, Node Manager, etc. will be running as a separate process on separate JVMs
  - Daemons run on different java processes that is why it is called a Pseudo-distributed.
- **Fully-Distributed Mode**
  - Hadoop runs on clusters of Machine or nodes.
  - Few of the nodes run the Master Daemon's that are Namenode and Resource Manager
  - Rest of the nodes run the Slave Daemon's that are DataNode and Node Manager.
  - Data is distributed across different Data nodes.
  - *Production Mode* of Hadoop cluster providing high availability of data by replication

Map Reduce overview

The term MapReduce actually refers to two separate and distinct tasks that Hadoop programs perform:

Map & ***Reduce***

- Map : This is the first job, which takes a data set and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs).

- Reduce : The reduce job takes the output from a map as input and combines those data tuples into a smaller set of tuples.

What the developer does

→ MapReduce is conceptually like a UNIX pipeline

- One function (Map) processes data
- Output is input to another function (Reduce)

cat file01 | sed -E 's/[\t]+/\n/g' | sort | uniq -c > output1

input | **map** | shuffle | **reduce** | output

→ Developer specifies two functions:

- map() - User code
- reduce() - User code

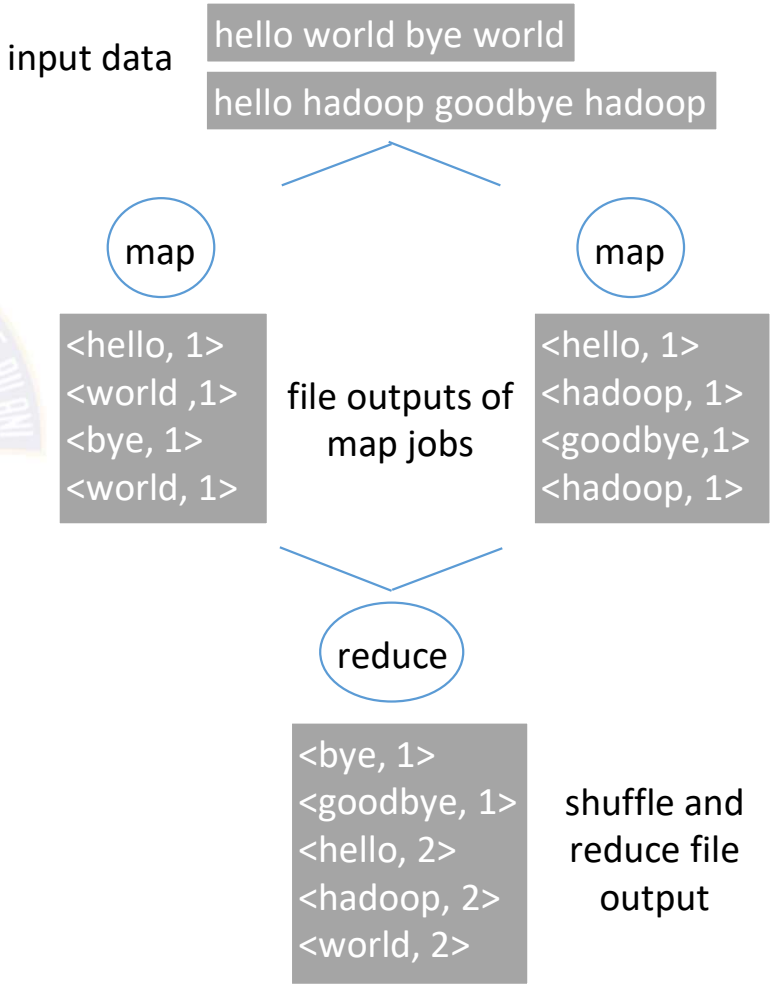
→ Rest of the job is done by the MapReduce framework

→ Tune the configuration parameters of the MapReduce framework for performance

Example - Word count

```
public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
```

```
public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
```



Hands on with Apache Hadoop

- Walkthrough with Hadoop cluster setup in Pseudo distributed mode

- Running MapReduce jobs

Introduction to Spark

Issues with MapReduce on Hadoop

✓ MapReduce revolutionized big data processing, enabling users to store and process huge amounts of data in parallel at very low costs.

✓ MapReduce is an ideal platform to implement **complex batch applications** as diverse as:

- sifting through system logs
- running ETL
- computing web indexes
- recommendation systems etc.

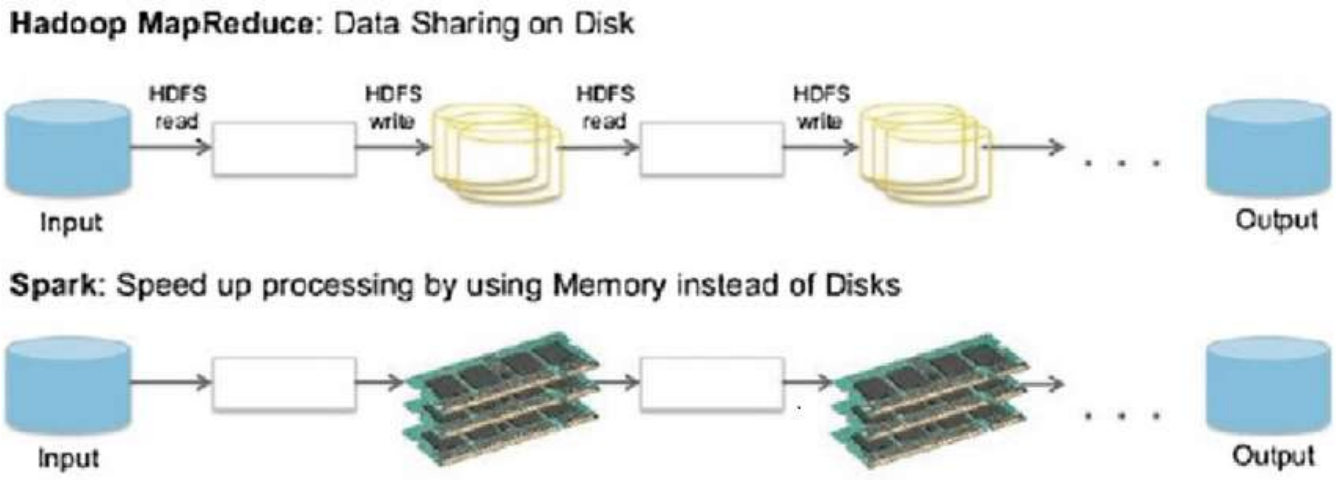
✓ Its reliance on persistent storage to provide fault tolerance and its one-pass computation model make **MapReduce** a poor fit for

- **low-latency / interactive applications**
- iterative computations, such as machine learning and graph algorithms
  - There are extensions for iterative MapReduce that we study later

Adapted from : https://databricks.com/blog/2013/11/21/putting-spark-to-use.html

In-Memory computing

- In-memory computing
  - ✓ Means using a type of middleware software that allows one to store data in RAM, across a cluster of computers, and process it in parallel
- For example,
  - ✓ Operational datasets typically stored in a centralized database which you can now store in **"connected" RAM** across multiple computers.
  - ✓ RAM is roughly 5,000 times faster than traditional spinning disk.
  - ✓ Native support for parallel processing makes it faster



Note:
Could be batch or streaming

Apache Spark: Fast In-Memory Computing for Your Big Data Applications

- Interactive Data Analysis
- Faster Batch
- Performance for Iterative Algorithms
- Real-Time Stream Processing
- Faster Decision-Making
- Unified Pipelines

Fast and easy big data processing with Spark (next class)

- At its core, Spark provides a general programming model that enables developers to write application by composing arbitrary operators, such as
  - ✓ mappers
  - ✓ reducers
  - ✓ joins
  - ✓ group-bys
  - ✓ filters
- This composition makes it easy to express a wide array of computations, including iterative machine learning, streaming, complex queries, and batch.
- Spark keeps track of the data that each of the operators produce , and enables applications to reliably store this data in memory using **RDDs***.
  - ✓ This is the key to Spark's performance, as it allows applications to avoid costly disk accesses.

* RDD: Resilient Distributed Dataset

Example - Word count

```
Val line = sparkContext.textFile("hdfs://...")
  .flatMap(line => line.split(" "))
  .map(word => (word, 1))
  .reduceByKey(_ + _)
  .saveAsTextFile("hdfs://...")
```

convert a file into lines

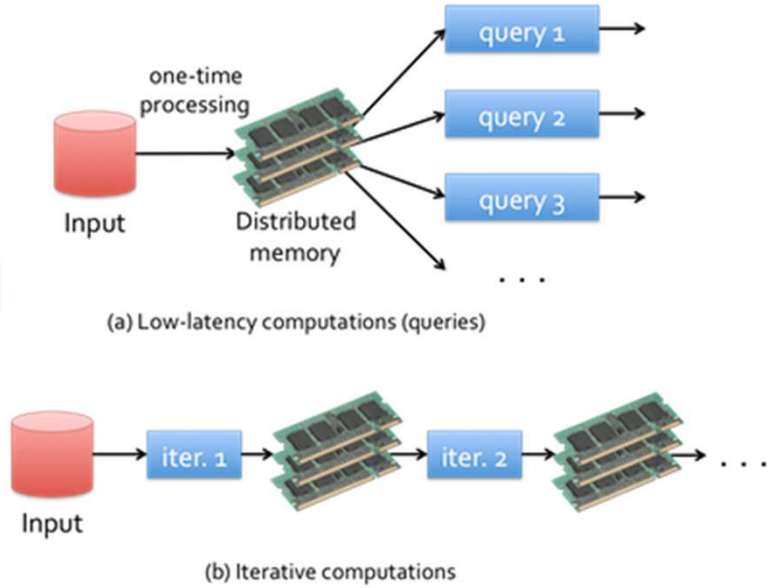
map: transform each word into <k,v> pair

reduce: sum up values for each key

- Can read data from many sources, including Local / HDFS
- Map / reduce output is written to memory instead of files
- Memory content can be written out to files etc.
- A rich set of primitives on top of MapReduce model to make it easier to program

Ideal Apache Spark applications

- Low-latency computations
  - ✓by caching the working dataset in memory and then performing computations at memory speeds
- Efficient iterative algorithm
  - ✓by having subsequent iterations share data through memory, or repeatedly accessing the same dataset



Hands on with Apache Spark

- Walkthrough with Spark Installation
- Running MapReduce jobs
- Spark-shell (Scala)
- PySpark (Python) -

  https://spark.apache.org/docs/latest/api/python/index.html
- sparkR (R) - (https://spark.apache.org/docs/latest/sparkr.html)

Fault tolerance comparison of Hadoop and Spark

- Hadoop and Spark are fault tolerant. They can handle node failures, data corruption, and network issues without affecting the overall execution.
- However, Hadoop and Spark have different approaches to fault tolerance
- Hadoop
  - When any node fails, the workload is redistributed to the remaining active nodes. So, there is no special standby node (N to N)
  - When task fails on one node, YARN restarts it (stateless mode)
  - Relies on data replication and checkpointing to ensure fault tolerance, which means that it duplicates the data blocks across multiple nodes and periodically saves the state of the computation to disk.
  - HDFS maintains a secondary Namenode with automatic checkpointing of updates happening on Namenode.
  - Hadoop provides high reliability and durability, but also consumes more disk space and network bandwidth
- Spark
  - Spark provides fault-tolerance to RDDs by rebuilding them from their lineage
  - Spark relies on data lineage and lazy evaluation to ensure fault tolerance, which means that it tracks the dependencies and transformations of the RDDs and only executes them when needed.
  - Spark provides high performance and flexibility, but also requires more memory and computation power.

Summary

- Concepts of fault tolerance - availability and reliability
  - Calculating MTTR and availability of systems
  - HA cluster configurations
- Basic concepts and definitions of analytics and Big Data analytics
- Overview of some systems / technologies that support Big Data Analytics
  - Hadoop/MapReduce, Spark/In-memory computing …
- Hands on with Hadoop Cluster, MapReduce, and Spark

Next Session:
Big Data Lifecycle, CAP Theorem, NoSQL