Course Introduction

BITS Pilani
Pilani Campus

Pravin Y Pawar

BITS Pilani
Pilani Campus

DSECL ZC556
Stream Processing and Analytics
Lecture No. 1.0

Course Design

Course Objectives

The course aims

- ✓ To introduce the applications of streaming data systems
- ✓ To introduce the architecture of streaming data systems
- ✓ To introduce the algorithmic techniques used in streaming data systems
- ✓ To present survey of tools and techniques required for streaming data analytics

Course Design (cont…)

Learning Outcomes

- Understand the components of streaming data systems with their capabilities and characteristics
- Learn the relevant architecture and best practices for processing and analysis of streaming data
- Gain knowledge about the development of system for data aggregation, delivery and storage using Open source tools
- Get familiarity with the advance streaming applications like Streaming SQL, Streaming machine learning

Course Design (cont…)

Textbook

- Streaming Data : Understanding The Real-Time Pipeline, Andrew G. Psaltis, 2017, Manning Publications
- Real-Time Analytics : Techniques to Analyze and Visualize Streaming Data, Byron Ellis, 2014, Wiley

Reference Books

- Big Data – Principles and best practices of scalable real-time data systems, Nathan Marz, James Warren, 2017, Manning Publications
- Designing Data Intensive Applications, Martin Kleppmann, O'Reilly

Course Design (cont…)

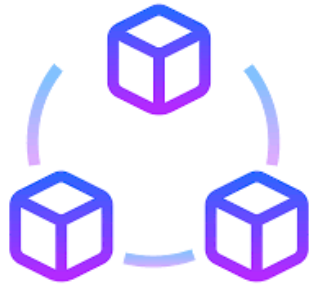
Software Requirement

- Apache Kafka
- Apache Storm
- Apache Spark
- AWS
- Azure

Course Design (cont…)

Modular Structure

M1	Scalable Streaming Data Systems
M2	Streaming Data Systems Architecture
M3	Streaming Data Frameworks
M4	Streaming Analytics
M5	Advanced Streaming Applications



Course Design (cont…)

Evaluation Scheme

	Name	Mode	Duration	Weight	Date
EC1	Assignment1	Take home	10 days	10%	TBA
EC1	Assignment 2	Take home	15 days	15%	TBA
EC1	Quiz	Online	TBA	05%	TBA
EC2	Midsem	Closed Book	2 hours	30%	TBA
EC3	Comprehensive	Open Book	3 hours	40%	TBA

Reference

Course Handout

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Reliable, Scalable and Maintainable Data Applications

Pravin Y Pawar

Data Processing Applications

Data Intensive Applications

- Deals with
  - ✓ huge amount of data
  - ✓ complex data
  - ✓ fast moving data

- Typically built with several building blocks like
  - ✓ Databases
  - ✓ Caches
  - ✓ Search Indexes
  - ✓ Streaming processing
  - ✓ Batch processing

Source : The topics are adapted from “Designing Data Intensive Applications” by Martin Kleppmann

Data Systems

Described

- Many different tools are integrated together for the data processing task
- Users are unaware about the seamless integration of tools / systems
- The boundaries between the categories of tools is diminishing
- No single tool can fit for all the data processing requirements
- Need to make sure this integration works properly end to end

Non Functional Requirements for Data Systems

Three Requirements

- Reliability
  - ✓ System should continue work correctly even in cases of failures
- Scalability
  - ✓ System should be able to cope with increase in load
- Maintainability
  - ✓ System should be able to work currently and adaptive to changes in future

Reliability

Described

- System should continue to work correctly, even when things go wrong
  - ✓ Should carry out expected operation correctly
  - ✓ Should handle wrong user inputs
  - ✓ Should prevent unauthorized access and usage

- Fault
  - ✓ Things that can go wrong
  - ✓ One component of system deviating from its working
  - ✓ Fault-tolerant / Resilient
    - ❖ Which can anticipate fault and deal with it properly

- Failure
  - ✓ System as a whole stops providing services

Reliability(2)

Faults

- Hardware Faults
  - ✓ Hard disk crash, RAM faults, Network issues etc
  - ✓ Add redundancy to individual components

- Software Faults
  - ✓ Software bugs, multi-threading issues
  - ✓ Happens under very unusual set of circumstances, hard to reproduce again
  - ✓ No straight away answer, need to rethink about the assumptions made while designing the system

- Human Errors
  - ✓ Developers designs the system, Operators maintains it
  - ✓ 10-25% outages are caused by wrong configurations done by operators
  - ✓ Design systems that minimizes opportunities for error

Scalability

Described

- Systems ability to cope with load
- Load impacts the performance of system
- Load defined differently for different data systems
  - ✓ Request per second, number of concurrent users etc

- How system reacts when load increases?
  - ✓ If the system resources are kept same?
  - ✓ What additions needs to be done in systems resources?

Maintainability

Described

- Easy to write a software , but very difficult to maintain it
- Involves
  - ✓ Bug fixing
  - ✓ Keeping existing systems operational
  - ✓ Detecting root cause of failures
  - ✓ Adapting to new platforms

- Legacy systems needs to be maintained as they are critical for the business operations.
- Data Systems should be designed in such a way that
  - ✓ They are easily operable
  - ✓ They are simple to understand
  - ✓ They are easy to adapt to new changes

Maintainability (2)

Approaches

- Operable
  - ✓ Easy to work with systems should be developed
  - ✓ Appropriate documentation to be provided
  - ✓ Monitoring capabilities should be present
  - ✓ Support for automation and integration with other tools should be possible
- Simplified
  - ✓ Complexity slows down everything , makes maintenance hard, more vulnerable to errors
  - ✓ Simpler system does not mean compromise on features
  - ✓ Use Abstraction, it hides lot of complexity behind a clean interface
  - ✓ Finding good abstractions is hard
- Extensible
  - ✓ Making changes should be easy
  - ✓ Adding new features, accommodating new requirements should be possible

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

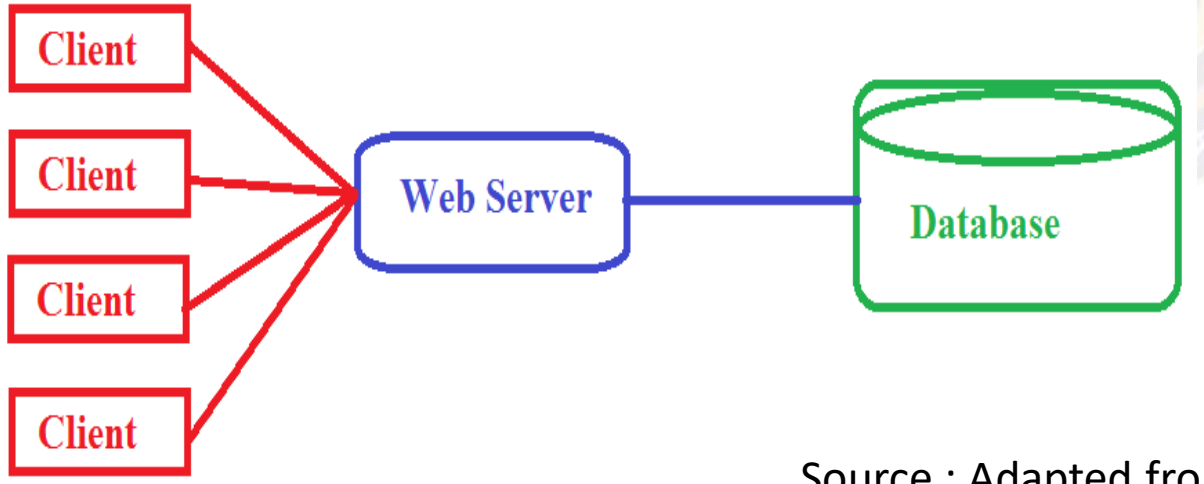
Scaling with Traditional Databases

Pravin Y Pawar

Web Analytics Application

Example Analytics Application

- Designing an application to monitor the page hits for a portal
- Every time a user visiting a portal page in browser, the server side keeps track of that visit
- Maintains a simple database table that holds information about each page hit
- If user visits the same page again, the page hit count is increased by one
- Uses this information for doing analysis of popular pages among the users



Column	Data Type
Id	Integer
User_ID	Integer
Page_URL	Varchar
Page_count	Long

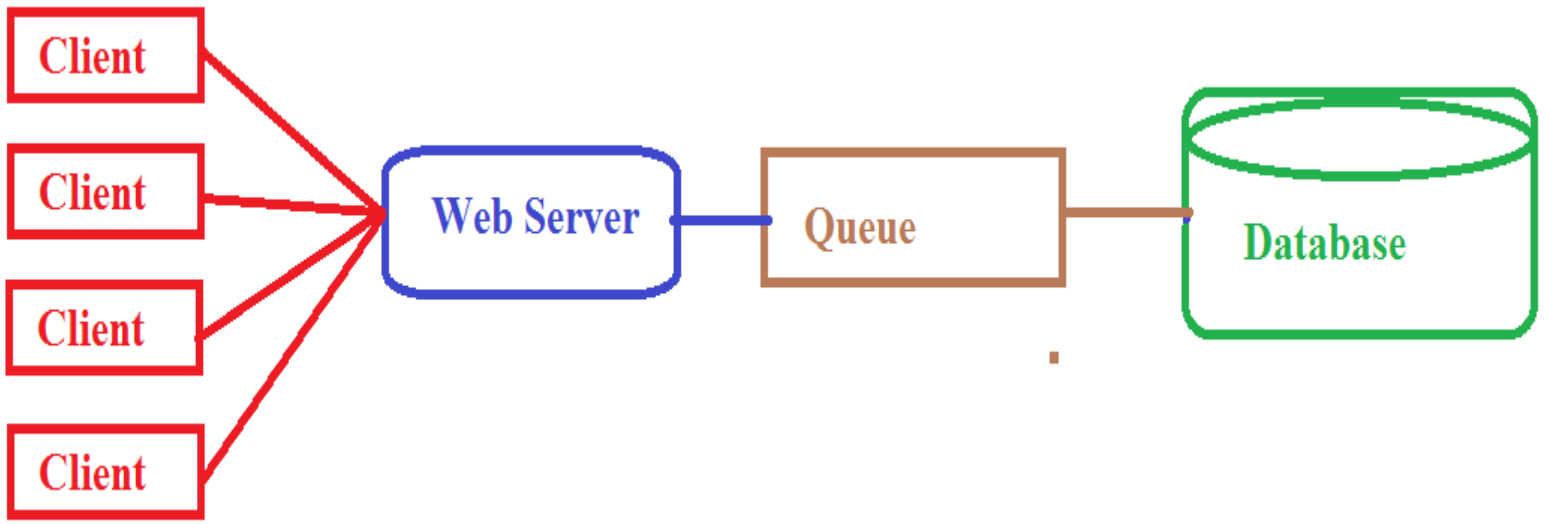
Source : Adapted from Big Data by Nathan Marz

Scaling with intermediate layer

Using a queue

- Portal is very popular, lot of users visiting it
  - ✓ Many users are concurrently visiting the pages of portal
  - ✓ Every time a page is visited, database needs to be updated to keep track of this visit
  - ✓ Database write is heavy operation
  - ✓ Database write is now a bottleneck

- Solution
  - ✓ Use an intermediate queue between the web server and database
  - ✓ Queue will hold messages
  - ✓ Message will not be lost

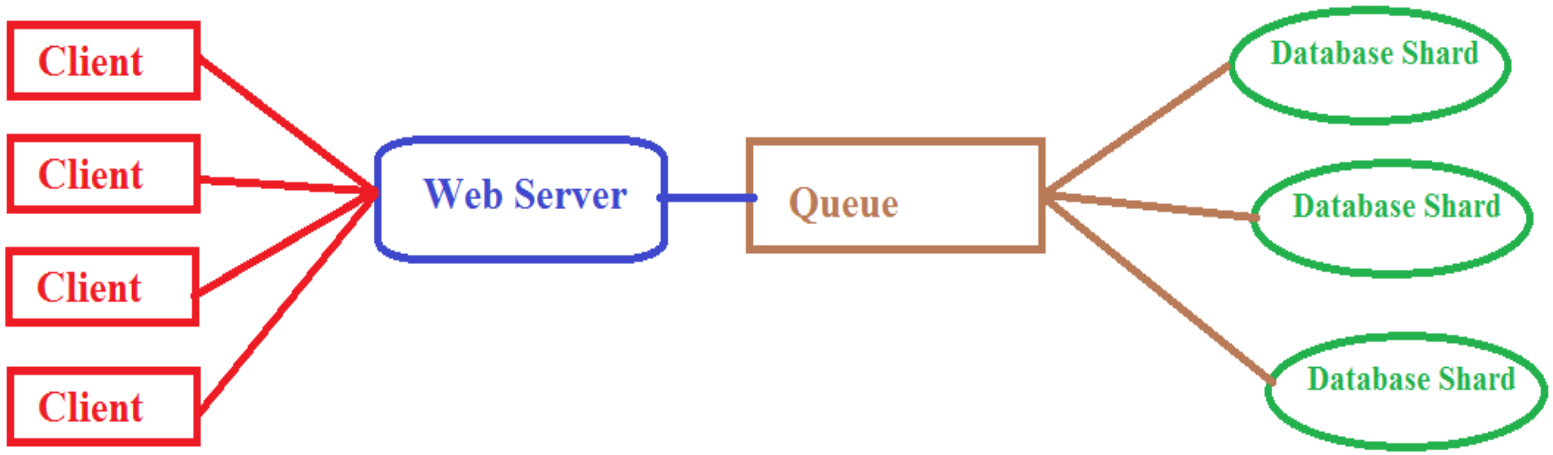


Scaling with Database Partitions

Using Database shards

- Application is too popular
  - ✓ Users are using it very heavily, increasing the load on application
  - ✓ Maintaining the page view count is becoming difficult even with queue

- Solution
  - Use database partitions
  - ✓ Data is divided into partitions which are hosted on multiple machines
  - ✓ Database writes are parallelized
  - ✓ Scalability increasing
  - ✓ Also complexity increasing



Issues Begins

Bottlenecks

- Disks are prone to failure, hence partition can be inaccessible
- Complicated to manage many number of shards
- Repartitioning is again required when load increased
- More buggy application code as complexity increasing
- Difficult to retrieve from the mistakes done either by application code or humans

Rise of Big Data Systems

How it helps

- Main issue with traditional data processing applications
  - ✓ Hard to make them scalable
  - ✓ Hard to keep them simple
- Because everything is managed by application code
  - ✓ Which is more prone to mistakes due to buggy implementations

- New edge systems aka Big Data Systems
  - ✓ Handles high data volume, at very fast rate coming from variety of sources
  - ✓ Systems aware about the distributed nature, hence capable of working with each other
  - ✓ Application does need to bother about common issues like sharding, replication etc
  - ✓ Scalability is achieved by horizontal scaling – just add new machines
  - ✓ Developers more focused on application logic rather than maintaining the environment

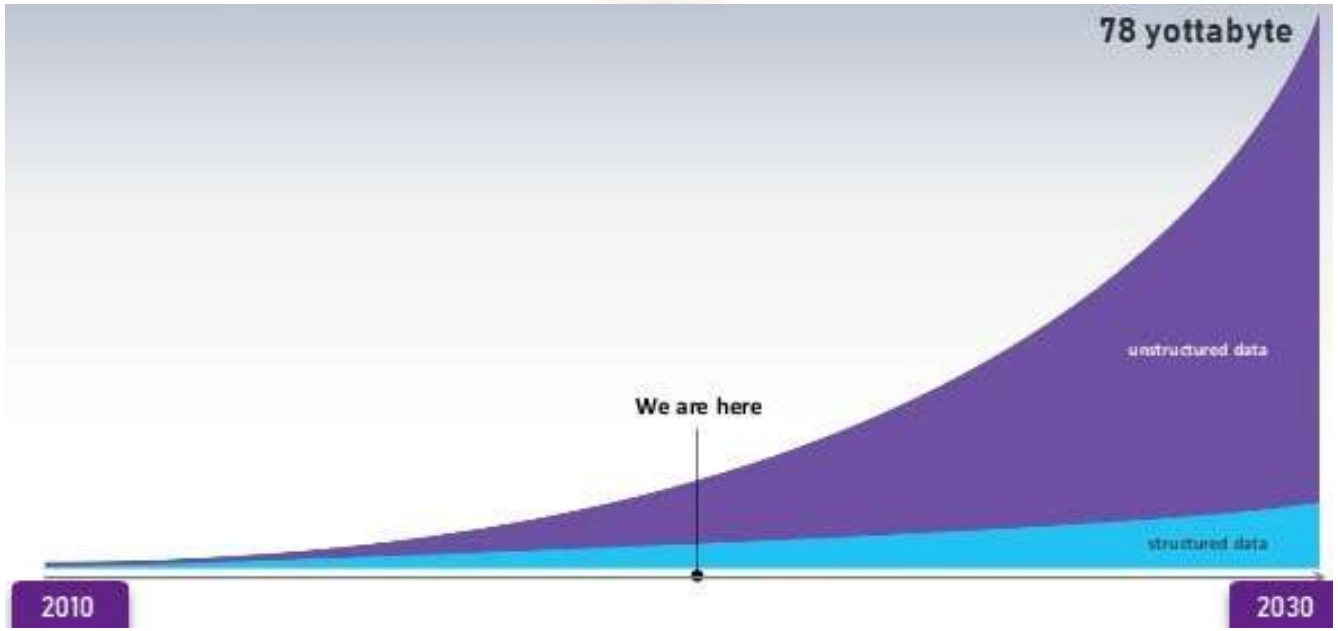
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Big Data Systems

Pravin Y Pawar

Data Growth

Data Generation

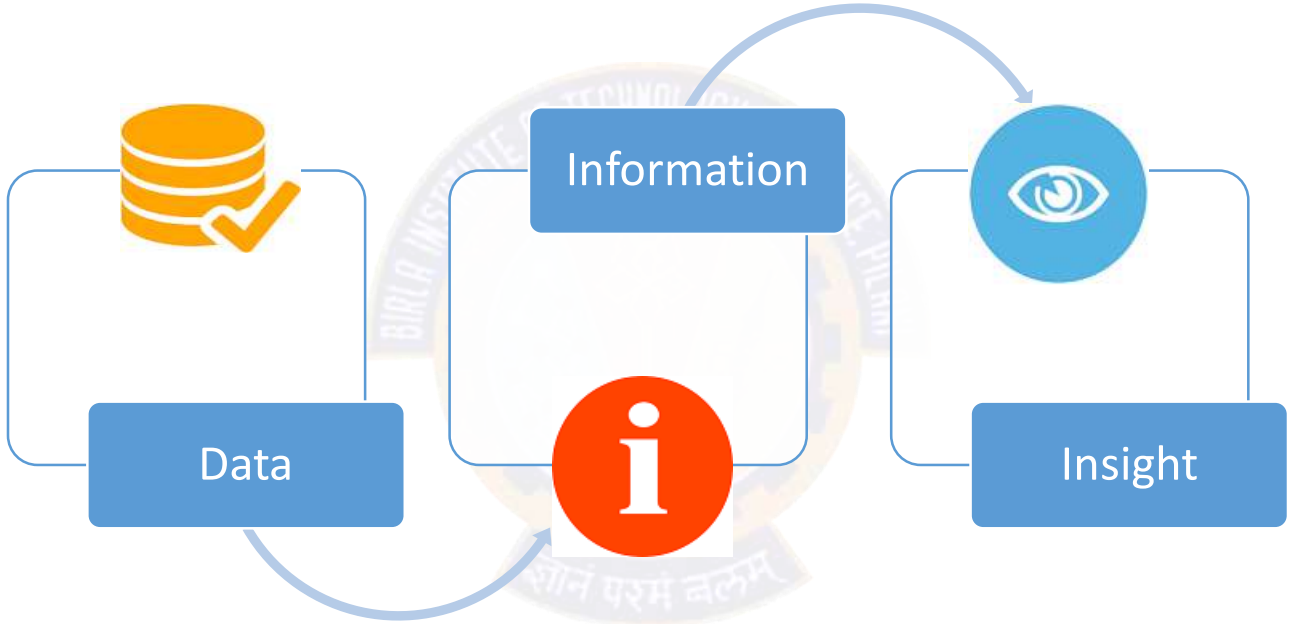


Source : https://www.guru99.com/what-is-big-data.html

Big Data Systems

What?

- New Paradigm for Data Analytics



Data Classification

Types of Data

- Structured

Databases

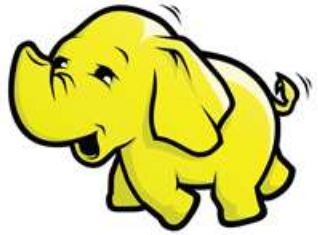
- Semi-structured

XML

- Unstructured

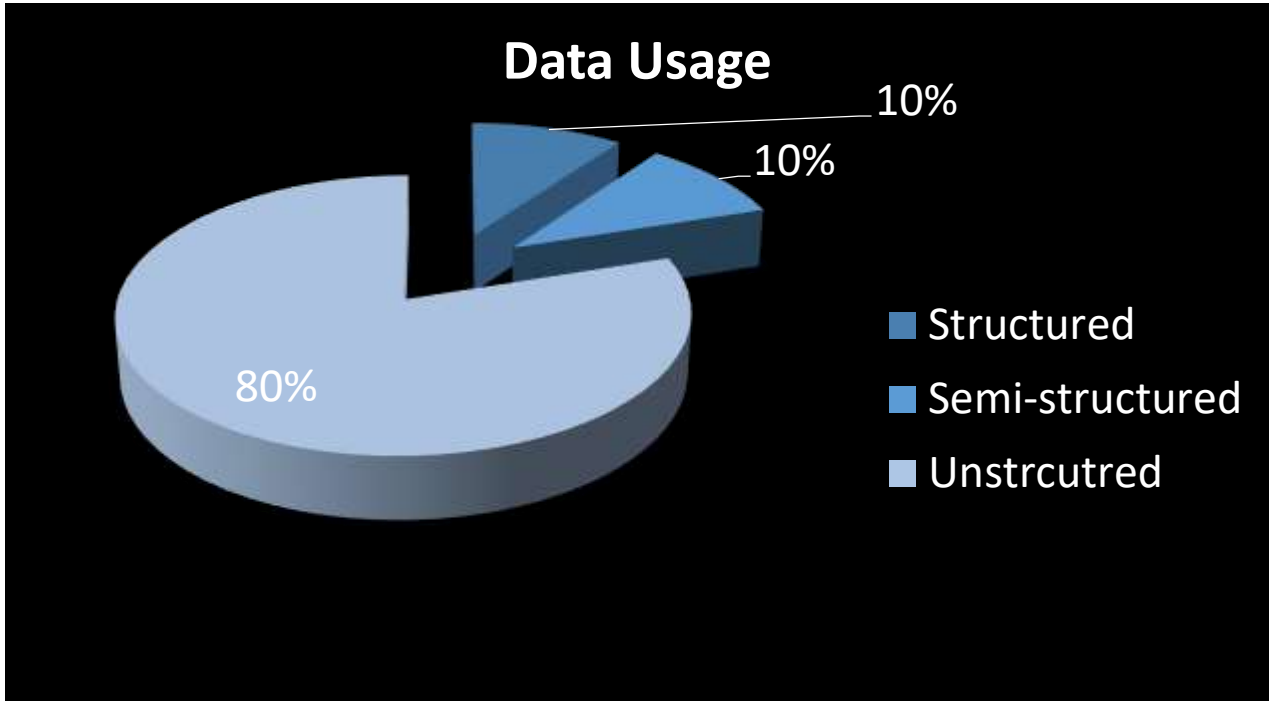
Web pages

Images



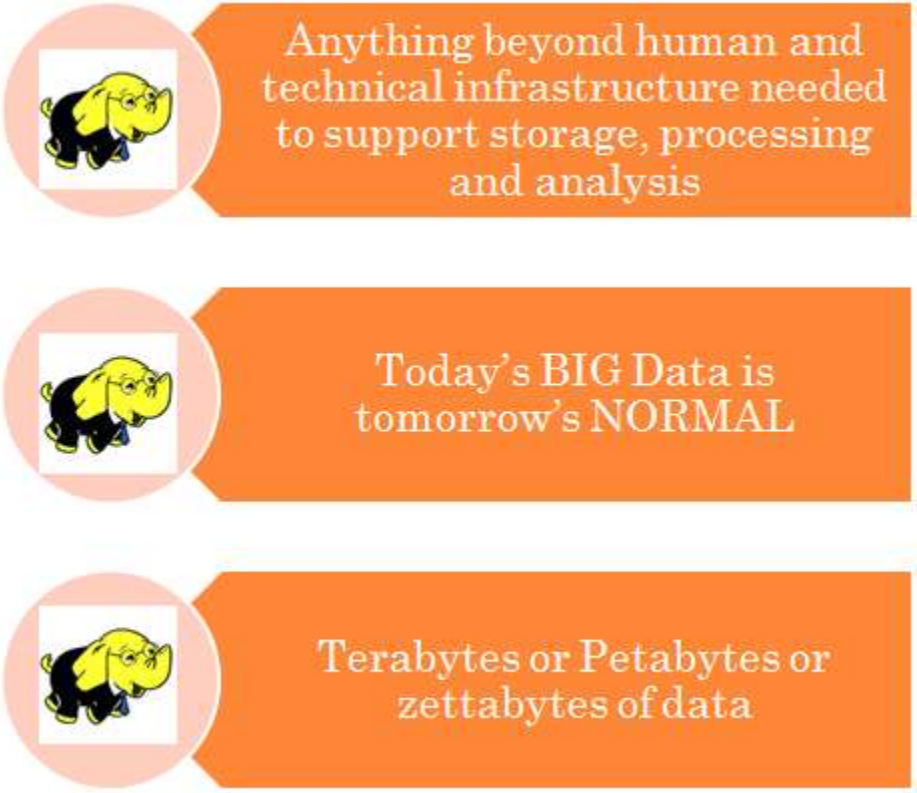
Data Usage Pattern

Usage Stats

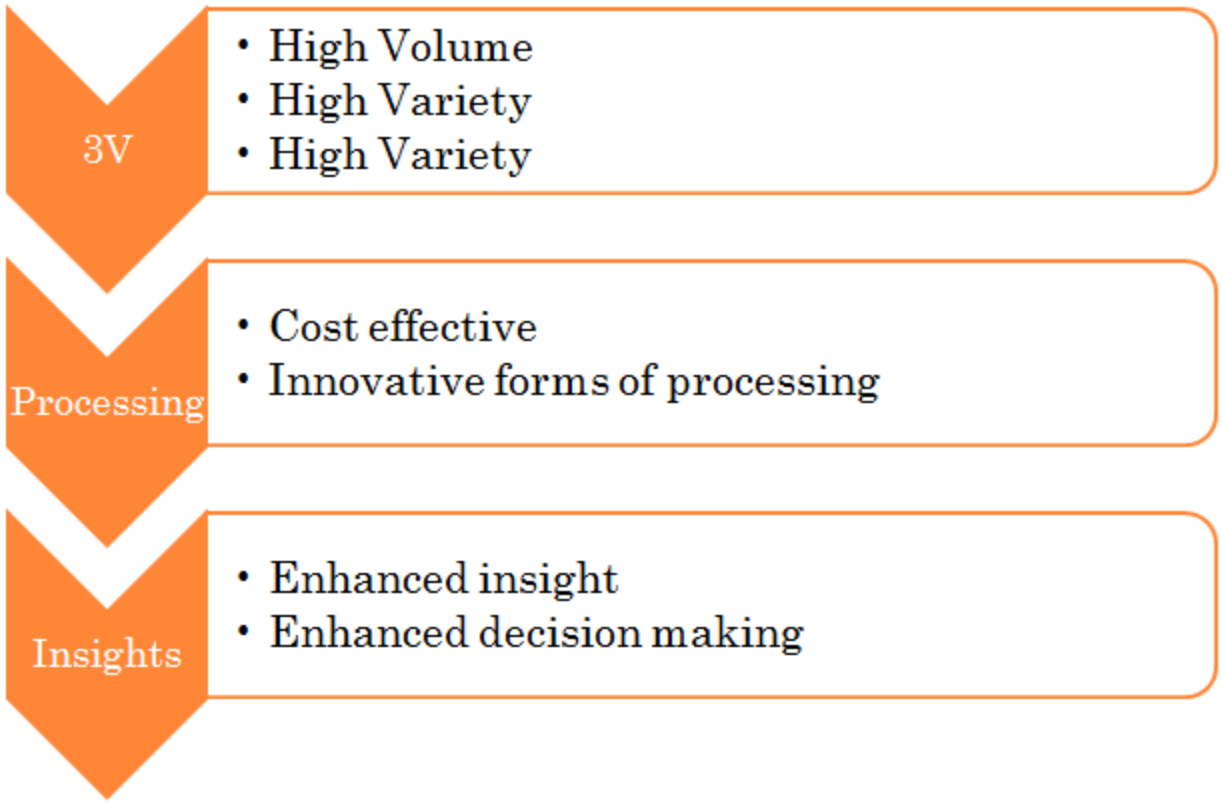


Big Data

Defined

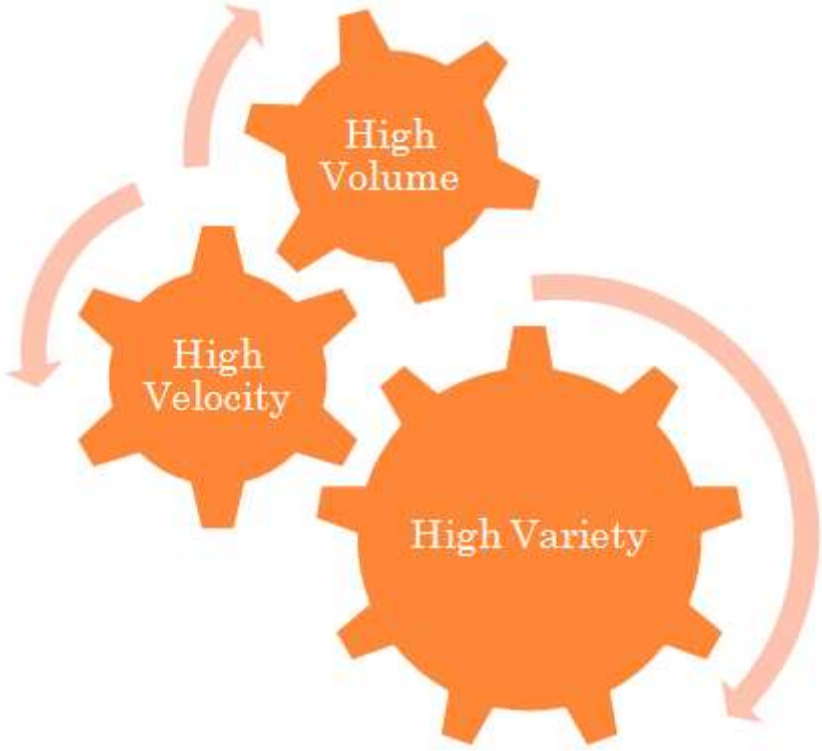


Big data – by Gartner



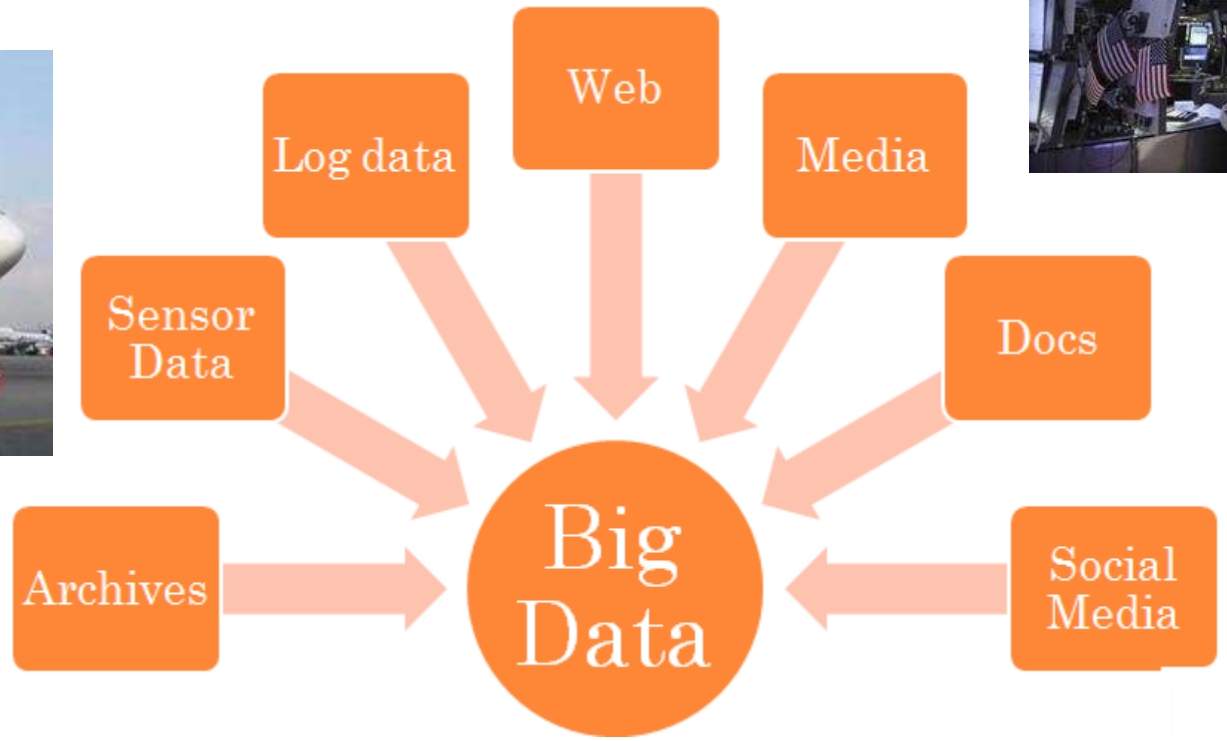
3 V’s of Big Data

Volume, Velocity and Variety



Sources of Big Data

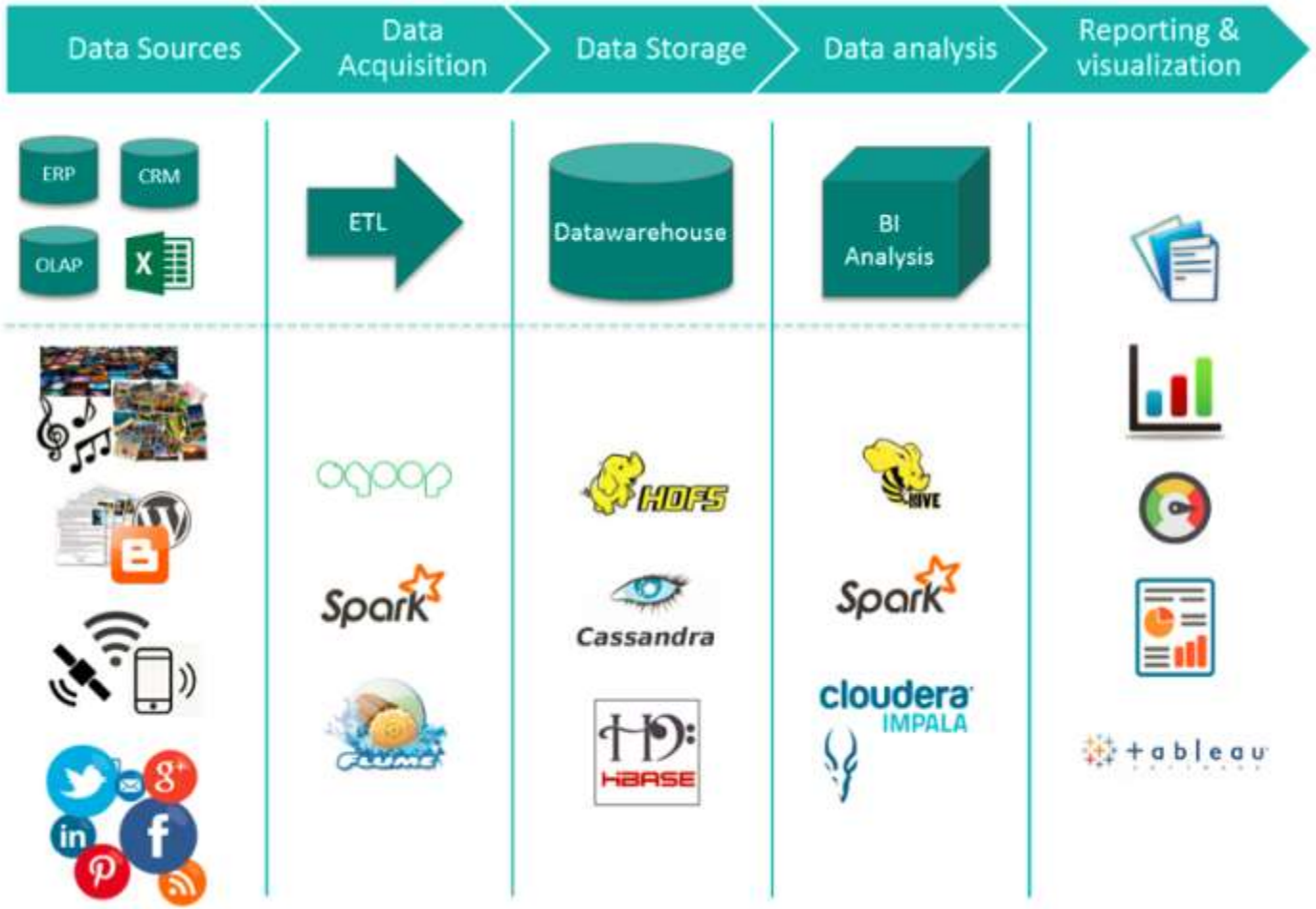
New Age Data Sources



Source : https://www.guru99.com/what-is-big-data.html

Big Data Ecosystem

Landscape of Big Data Systems



Source : https://medium.com/@jain6968/big-data-ecosystem-b0e4c923d7aa

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Desired Properties of Big Data Systems

Pravin Y Pawar

Data Systems

Defined

- System which answers the users questions based on the data accumulated over the period or in real time
  - ✓ What are the sales figures for last 6 months?
  - ✓ How is the health of data center at this moment?
- Data is different from information
- Information is derived from the data
- Data systems makes use of data to answer users queries

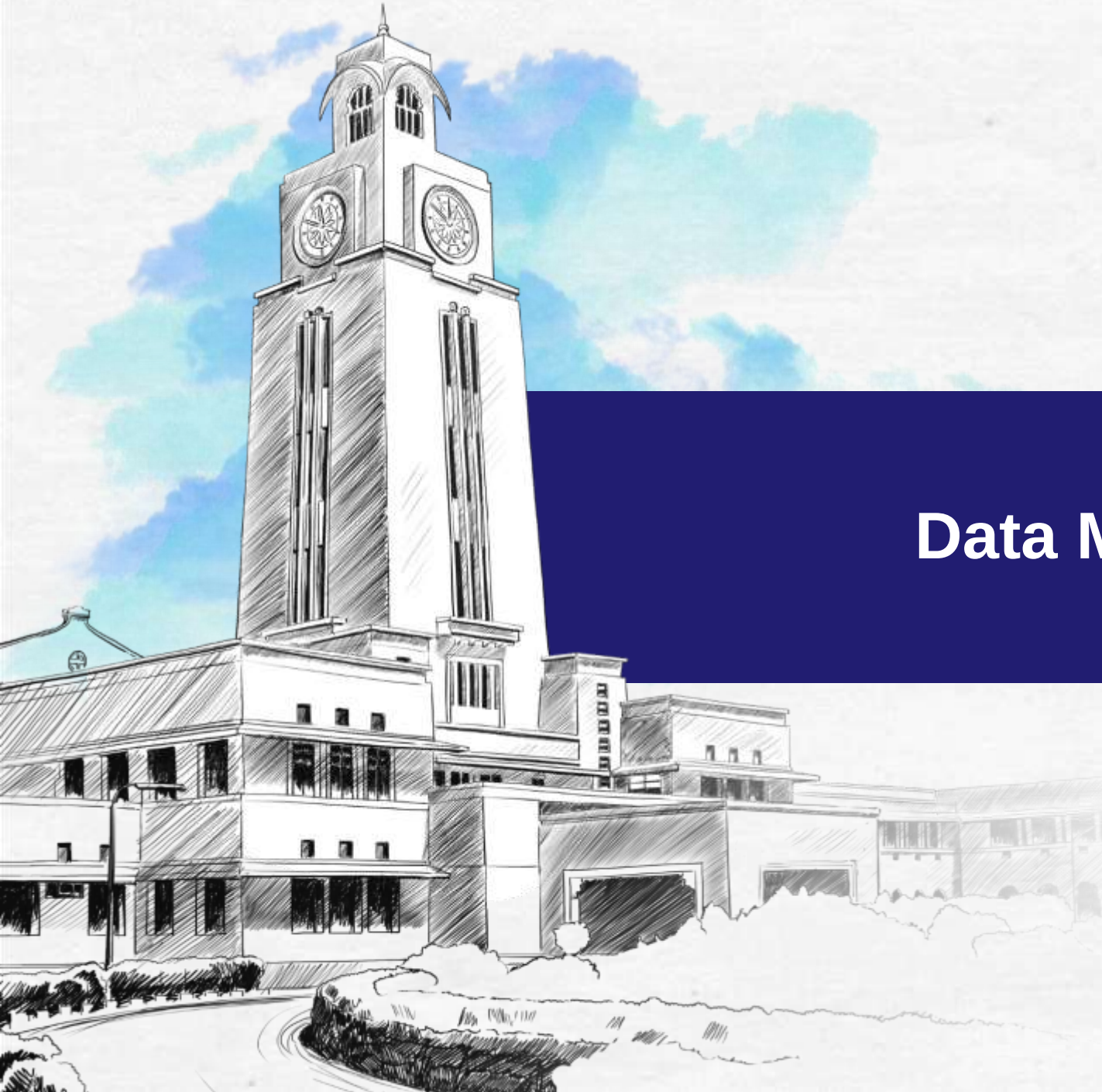
- Query = function (all data)

Source : Big Data , Nathan Marz

Properties of Big Data Systems

Properties

- Fault tolerance
  - ✓ Correct behavior of system even in case of failures
- Low latency
  - ✓ Both read and write response times should be as low as possible
- Scalability
  - ✓ Easy to manage the load just by adding the additional machines
- Extensibility
  - ✓ Easy to update or add new features in the system
- Maintainability
  - ✓ Easy to keep system running without facing critical issues
- Debuggability
  - ✓ Should be diagnose systems health if system behaves in inappropriate manner



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Model of Big Data Systems

Pravin Y Pawar

Properties of Data

Three Properties

- Rawness
  - ✓ Fine grained data
  - ✓ Many interesting queries can be answered
  - ✓ Unstructured data is more rawer than structured data

- Immutability
  - ✓ No deletion or updates to data, only new addition
  - ✓ Original data is untouched
  - ✓ Easy to retrieve back from the failures / mistakes
  - ✓ No indexing required, enforcing simplicity

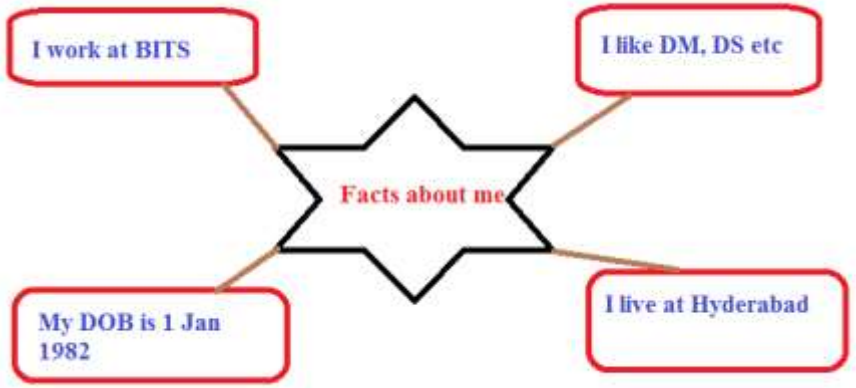
- Eternity
  - ✓ Consequence of immutability
  - ✓ Data is always pure and true
  - ✓ Achieved with adding timestamp with the data

Source : Big Data , Nathan Marz

Fact based Model for Data

Facts

- Fundamental unit of data
- Data can not derived from anything else
- Fact is atomic and time stamped
- Can be made unique by adding unique identifier to it
- Example facts
  - ✓ I work for BITS Pilani
  - ✓ My DOB is 1 January 1982
  - ✓ I currently live in Hyderabad
  - ✓ I was living at Pune between 2010 to 2015
  - ✓ I have interest in subjects like data mining, data science etc



Fact based Model for Data (2)

Benefits

- Fact based model
  - ✓ Stores raw data as atomic facts
  - ✓ Makes facts immutable by adding time stamp value to it
  - ✓ Makes it uniquely identifiable by attaching identifier

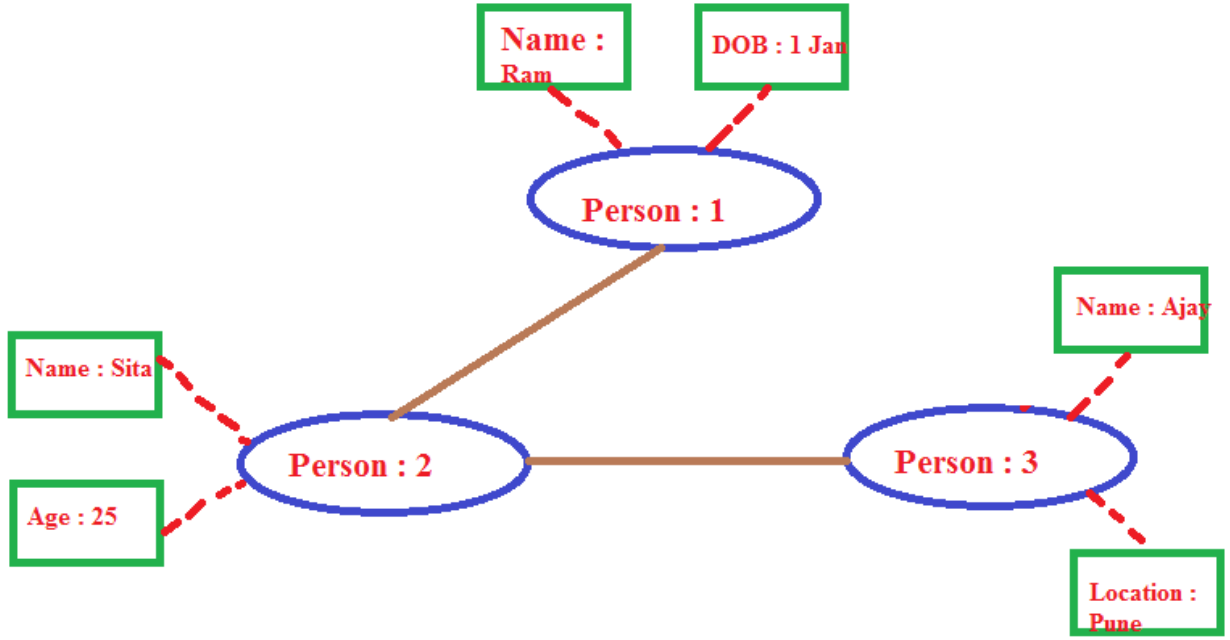
- Benefits
  - ✓ Data is query able at any time
  - ✓ Data is tolerant to human errors
  - ✓ Data can be stored both in structured and unstructured formats

Fact based Model for Data (3)

Structure / Schema

- Graph schemas captures the structure of dataset stored using fact based model
- Depicts the relationship between nodes, edges and properties

- Nodes
  - ✓ Entities in system , for example, person
- Edges
  - ✓ Relationship between entities,
  - ✓ For example, person A knows person B
- Properties
  - ✓ Information about entities



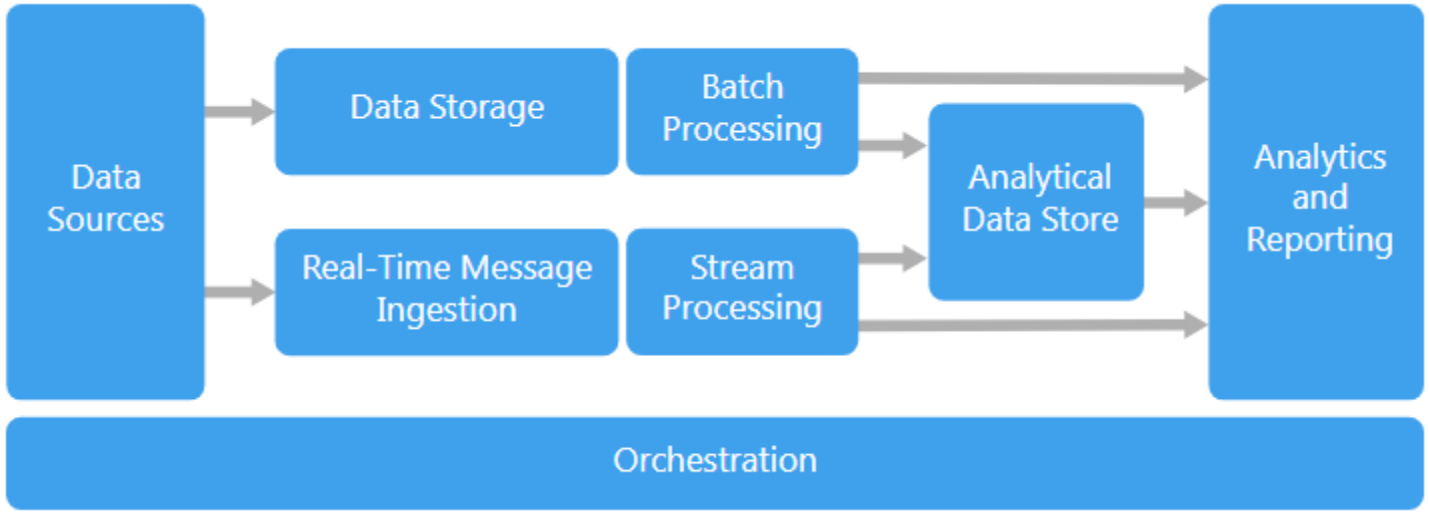
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Generalized Architecture of Big Data Systems

Pravin Y Pawar

Big data architecture style

- is designed to handle the ingestion, processing, and analysis of data that is too large or complex for traditional database systems.



Source : https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data

Big Data Applications

Workloads

- Big data solutions typically involve one or more of the following types of workload:

  - ✓ Batch processing of big data sources at rest
  - ✓ Real-time processing of big data in motion
  - ✓ Interactive exploration of big data
  - ✓ Predictive analytics and machine learning

Big Data Systems Components

Components

- Most big data architectures include some or all of the following components:
  - ✓ Data sources
    - All big data solutions start with one or more data sources like databases, files, IoT devices etc
  - ✓ Data Storage
    - Data for batch processing operations is typically stored in a distributed file store that can hold high volumes of large files in various formats.
  - ✓ Batch processing
    - Because the data sets are so large, often a big data solution must process data files using long-running batch jobs to filter, aggregate, and otherwise prepare the data for analysis. Usually these jobs involve reading source files, processing them, and writing the output to new files.
  - ✓ Real-time message ingestion
    - ✓ If the solution includes real-time sources, the architecture must include a way to capture and store real-time messages for stream processing.
  - ✓ Stream processing
    - After capturing real-time messages, the solution must process them by filtering, aggregating, and otherwise preparing the data for analysis. The processed stream data is then written to an output sink.
  - ✓ Analytical data store
    - Many big data solutions prepare data for analysis and then serve the processed data in a structured format that can be queried using analytical tools. The analytical data store used to serve these queries can be a Kimball-style relational data warehouse, as seen in most traditional business intelligence (BI) solutions.
  - ✓ Analysis and reporting
    - The goal of most big data solutions is to provide insights into the data through analysis and reporting.
  - ✓ Orchestration
    - Most big data solutions consist of repeated data processing operations, encapsulated in workflows, that transform source data, move data between multiple sources and sinks, load the processed data into an analytical data store, or push the results straight to a report or dashboard. To automate these workflows, you can use an orchestration technology such Azure Data Factory or Apache Oozie and Sqoop.

Big data architecture Usage

When to use this architecture

- Consider this architecture style when you need to:

  - ✓ Store and process data in volumes too large for a traditional database
  - ✓ Transform unstructured data for analysis and reporting
  - ✓ Capture, process, and analyze unbounded streams of data in real time, or with low latency

Big data architecture Benefits

Advantages

- Technology choices
  - ✓ Variety of technology options in open source and from vendors are available
- Performance through parallelism
  - ✓ Big data solutions take advantage of parallelism, enabling high-performance solutions that scale to large volumes of data.
- Elastic scale
  - ✓ All of the components in the big data architecture support scale-out provisioning, so that you can adjust your solution to small or large workloads, and pay only for the resources that you use.
- Interoperability with existing solutions
  - ✓ The components of the big data architecture are also used for IoT processing and enterprise BI solutions, enabling you to create an integrated solution across data workloads.

Big data architecture Challenges

Things to ponder upon

- Complexity
  - ✓ Big data solutions can be extremely complex, with numerous components to handle data ingestion from multiple data sources. It can be challenging to build, test, and troubleshoot big data processes.

- Skillset
  - ✓ Many big data technologies are highly specialized, and use frameworks and languages that are not typical of more general application architectures. On the other hand, big data technologies are evolving new APIs that build on more established languages.

- Technology maturity
  - Many of the technologies used in big data are evolving. While core Hadoop technologies such as Hive and Pig have stabilized, emerging technologies such as Spark introduce extensive changes and enhancements with each new release.

Thank You!

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Real time systems

Pravin Y Pawar

Responding in Real Time

Few Examples

- World is now operating more and more in the *now*
- Ability to process data as it arrives (processing data in the present)

- Your twitter friend / celebrity posts a tweet, and you see it almost immediately…
- You are tracking flights around New Delhi Live…
- You are using the nseindia.com tracker page to track your favourite stocks…

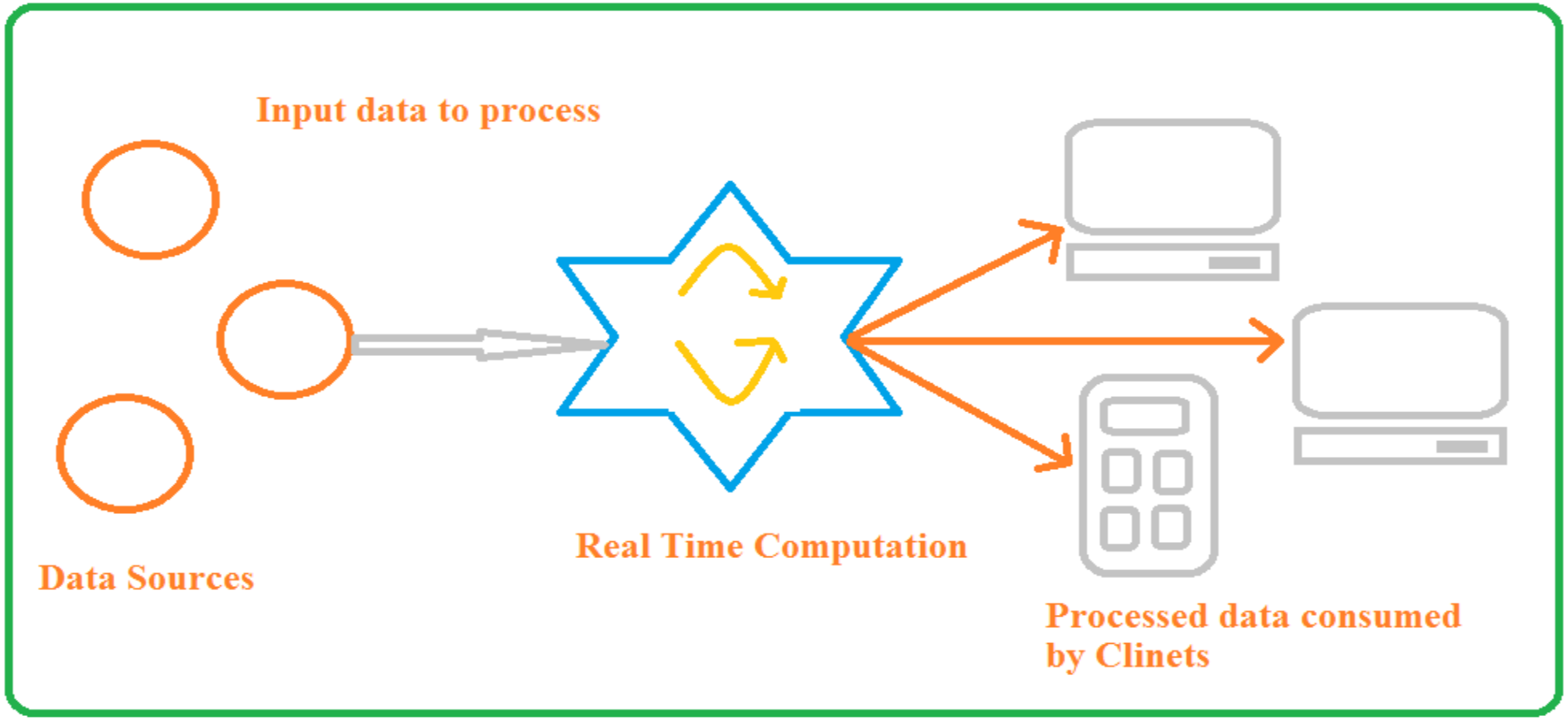
Classification of Real Time Systems

Classification	Examples	Latency measured in	Tolerance for delay
Hard	Pacemaker, anti-lock brakes	Microseconds–milliseconds	None—total system failure, potential loss of life
Soft	Airline reservation system, online stock quotes, VoIP (Skype)	Milliseconds–seconds	Low—no system failure, no life at risk
Near	Skype video, home automation	Seconds–minutes	High—no system failure, no life at risk

Source : Streaming Data by Andrew Psaltis

A generic real-time system with consumers

Consumers are connected to the service online



Source : Adapted from Streaming Data by Andrew Psaltis

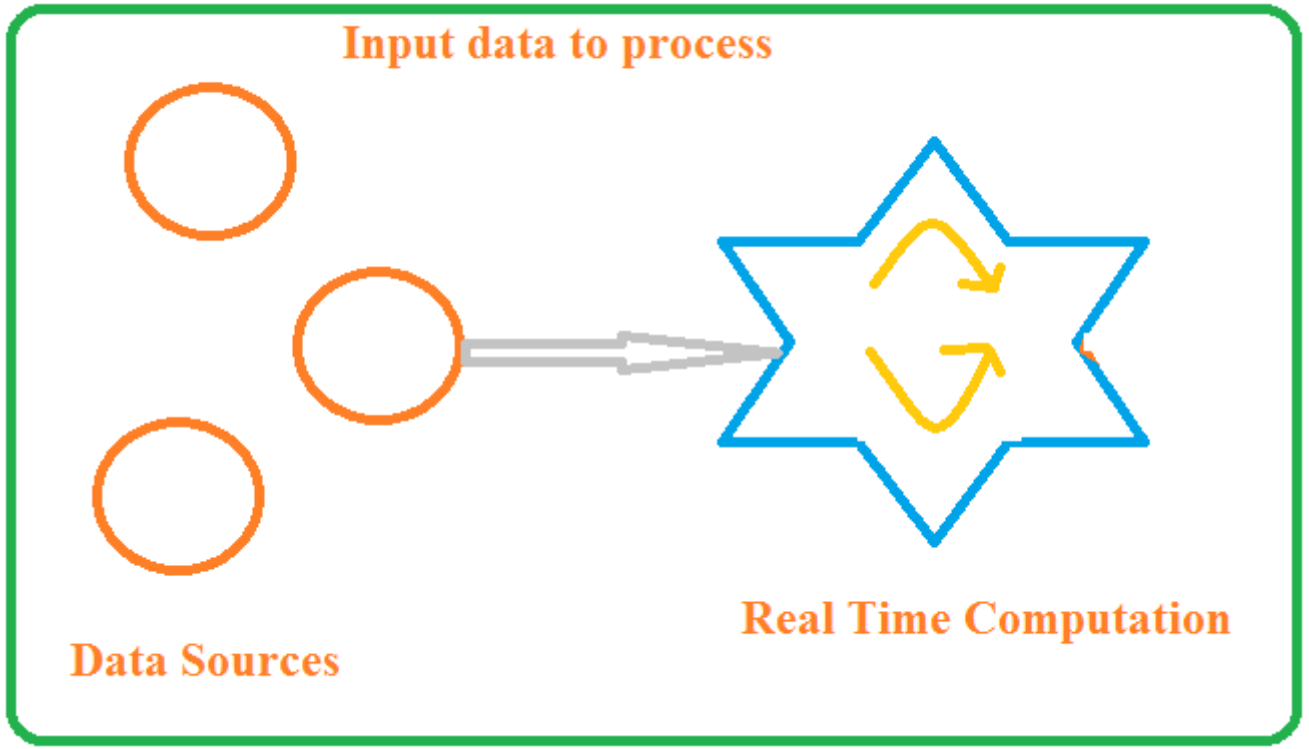
A generic real-time system with consumers(2)

Examples discussion

- In each of the examples, is it reasonable to conclude that
  - ✓the time delay may only last for seconds?
  - ✓no life is at risk?
  - ✓an occasional delay for minutes would not cause total system failure?

A generic real-time system without consumers

Consumers are not connected to the service but service continues its operation



Source : Adapted from Streaming Data by Andrew Psaltis

A generic real-time system without consumers(2)

Examples discussion

- A tweet is posed on twitter
- The Live Flight Tracking service tracking flights
- Real time quotes monitoring application is tracking the stock quotes

- Does focusing on the data processing and taking consumers of the data out of picture change your answer?

- Difference between soft real time and near real time becoming blurry
- Together they are just termed as real time

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

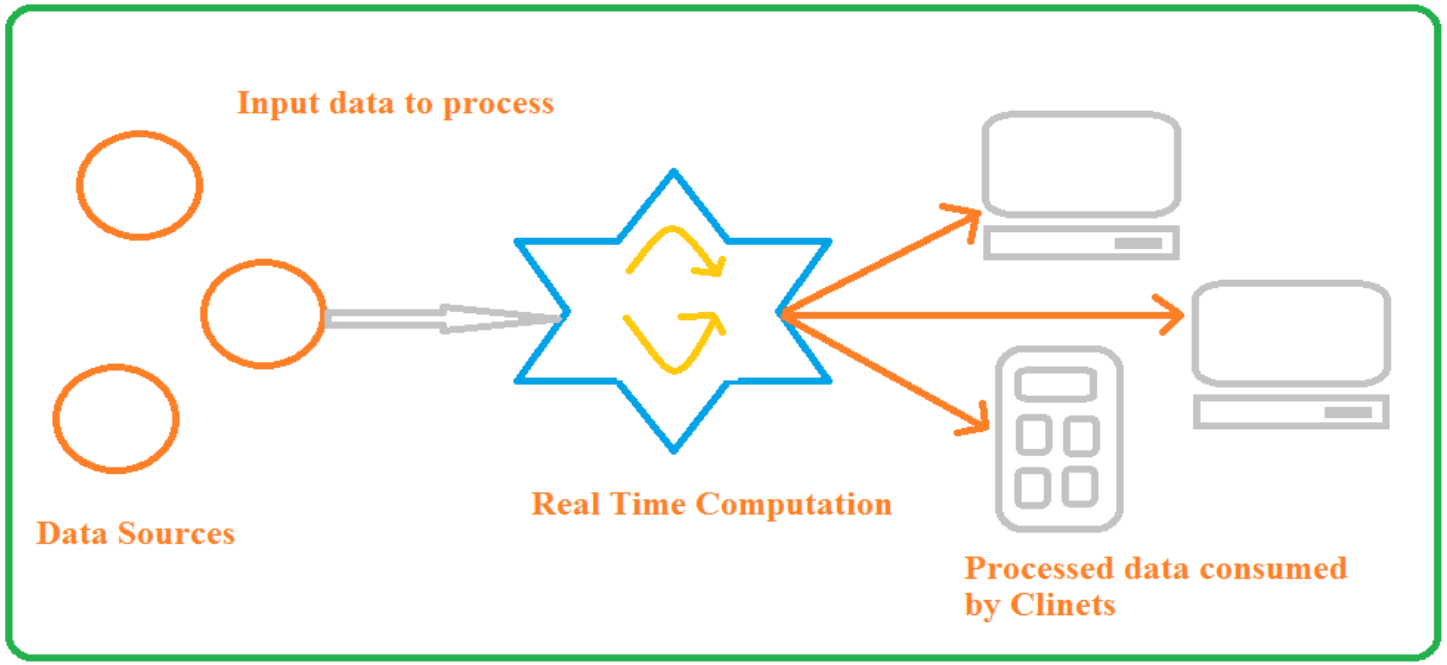
Difference between real-time and streaming systems

Pravin Y Pawar

Real-time systems again

Soft or near real time systems

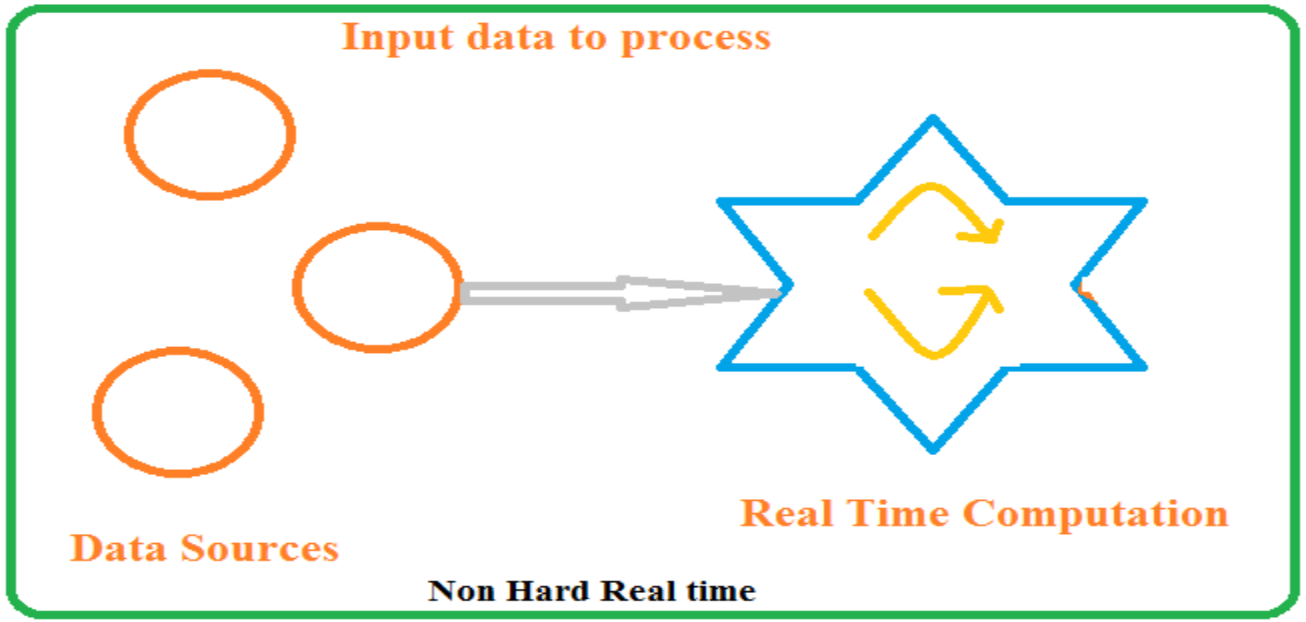
- Based on the delay experience by consumers – soft or near real time systems
- Difference between them is blurring , hard to distinguish
- Two parts are present in larger picture



Source : Adapted from Streaming Data by Andrew Psaltis

Breaking the real time system

- Lets divide the real time system into two parts
  - ✓ Left part Non-hard real-time system
  - ✓ Right part client consuming data



Source : Adapted from Streaming Data by Andrew Psaltis

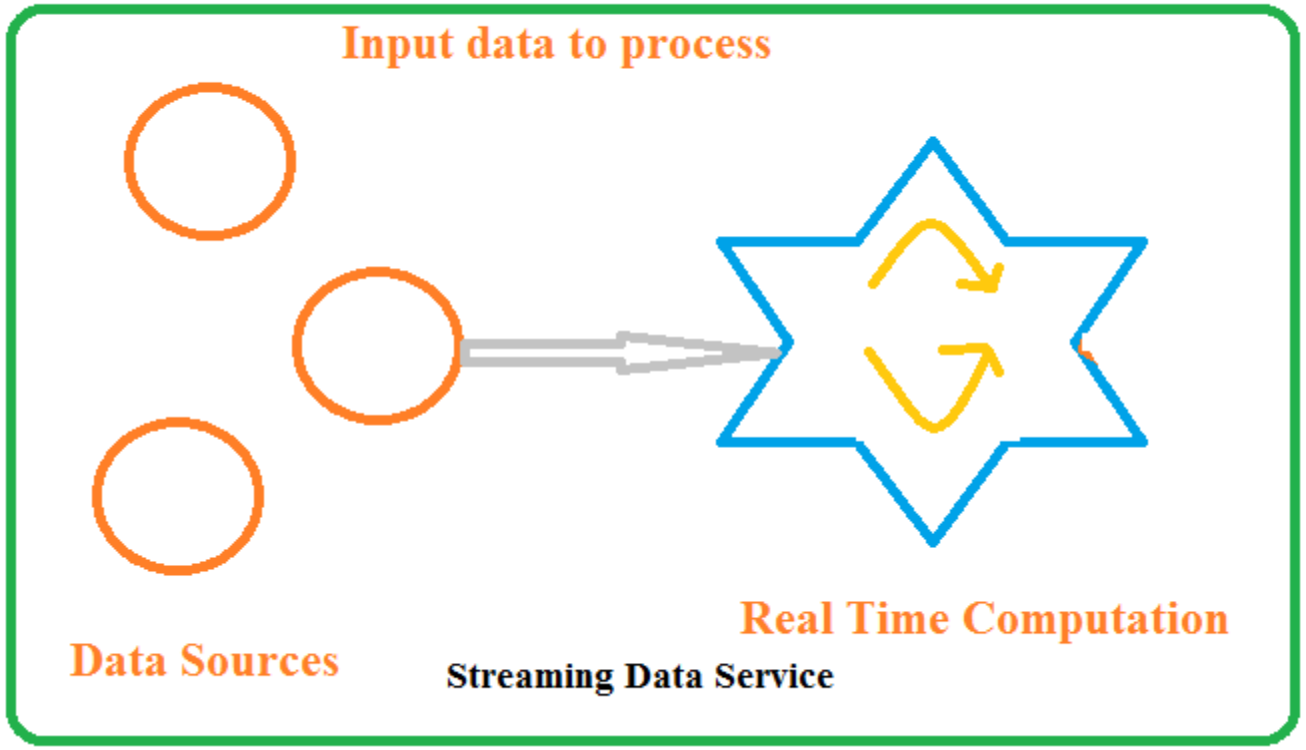
Streaming Data Systems

Defined

- Computation part of real time system operating in non-hard real-time manner
- But Client not consuming the processed data
  - ✓ Due to network issues, application requirement, no application running
- Clients consume data when they need it

- Streaming data system
  - ✓ Non-hard real time system with clients that consume data when they need it

First view of Streaming Data System



Source : Adapted from Streaming Data by Andrew Psaltis

Streaming Data Systems(2)

Examples revisited

- Lets divide the earlier discussed examples into two parts and identify the streaming part of it

- Twitter
  - ✓ A streaming system that processes tweets and allows clients to request the tweets when needed
- Flight Tracking System
  - ✓ A streaming system that processes most recent flight status data and allows client to request the latest data for particular flight
- Real time Quotes Tracking System
  - ✓ A streaming system that processes the price quotes of stocks and allows clients to request the latest quote of stock

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Difference between Batch Processing and Stream Processing

Pravin Y Pawar

Batch Processing System

Defined

- An efficient way of processing high volumes of data is where a group of transactions is collected over a period of time
- Data is collected, entered, processed and then the batch results are produced
  - ✓ Hadoop is focused on batch data processing
- Requires separate programs for input, process and output
- Huge volume of storage is required
- Data is sorted and then processed in sequential manner
- No hard timelines defined
- Sequential jobs are executed in repeated manner over fixed interval
- An example is payroll and billing systems

Streaming Data Systems

Defined

- Involves a continual input, process and output of data
- Data must be processed in a small time period (or near real time)
  - ✓ Apache Storm, Spark Stream processing are frameworks meant for the same
- Allows an organization the ability to take immediate action for those times when acting within seconds or minutes is significant
- No storage required if event need not be stored
- Data is processed as and when its made available to the system
- Processing has to happen in fixed / hard time lines
- Examples Radar systems, customer services and bank ATMs

Difference

	Batch processing	Stream processing
Data scope	Queries or processing over all or most of the data in the dataset.	Queries or processing over data within a rolling time window, or on just the most recent data record.
Data size	Large batches of data.	Individual records or micro batches consisting of a few records.
Performance	Latencies in minutes to hours.	Requires latency in the order of seconds or milliseconds.
Analyses	Complex analytics.	Simple response functions, aggregates, and rolling metrics.

Source : https://aws.amazon.com/streaming-data/

Frameworks

Below is list of batch and real time data processing solutions:

Solution	Developer	Type	Description
Storm	Twitter	Streaming	Twitter's new streaming big-data analytics solution
S4	Yahoo!	Streaming	Distributed stream computing platform from Yahoo!
Hadoop	Apache	Batch	First open source implementation of the MapReduce paradigm
Spark	UC Berkeley AMPLab	Batch	Recent analytics platform that supports in-memory data sets and resiliency
Disco	Nokia	Batch	Nokia's distributed MapReduce framework
HPCC	LexisNexis	Batch	HPC cluster for big data

Source : https://www.datasciencecentral.com/profiles/blogs/batch-vs-real-time-data-processing

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Streaming Data Applications

Pravin Y Pawar

Why is stream Processing needed?

Reason 1

- Some data naturally comes as a never-ending stream of events. To do batch processing, you need to store it, stop data collection at some time and processes the data.
- Then you have to do the next batch and then worry about aggregating across multiple batches.
- In contrast, streaming handles never ending data streams gracefully and naturally.
- You can detect patterns, inspect results, look at multiple levels of focus, and also easily look at data from multiple streams simultaneously.

Source : https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97

Why is stream Processing needed? (2)

Reason 2

- Stream processing naturally fit with time series data and detecting patterns over time.
- For example, if you are trying to detect the length of a web session in a never-ending stream (this is an example of trying to detect a sequence).
- It is very hard to do it with batches as some session will fall into two batches.
- Stream processing can handle this easily.

Why is stream Processing needed? (3)

Reason 3

- Batch processing lets the data build up and try to process them at once while stream processing process data as they come in hence spread the processing over time.
- Hence stream processing can work with a lot less hardware than batch processing.
- Furthermore, stream processing also enables approximate query processing via systematic load shedding.
- Hence stream processing fits naturally into use cases where approximate answers are sufficient.

Why is stream Processing needed? (4)

Reason 4

- Finally, there are a lot of streaming data available (e.g. customer transactions, activities, website visits) and they will grow faster with IoT use cases (all kind of sensors).
- Streaming is a much more natural model to think about and program those use cases.

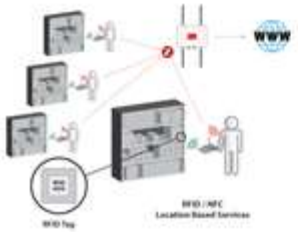
Streaming Data Sources

Data Collection Devices

Smart Machinery

Phones and Tablets

Home Automation



RFID Systems

Digital Signage

Security Systems

Medical Devices

Source : https://mapr.com/blog/real-time-streaming-data-pipelines-apache-apis-kafka-spark-streaming-and-hbase/

Streaming Data Examples

- Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance, detects any potential defects in advance, and places a spare part order automatically preventing equipment down time.
- A financial institution tracks changes in the stock market in real time, computes value-at-risk, and automatically rebalances portfolios based on stock price movements.
- A real-estate website tracks a subset of data from consumers' mobile devices and makes real-time property recommendations of properties to visit based on their geo-location.
- A solar power company has to maintain power throughput for its customers, or pay penalties. It implemented a streaming data application that monitors of all of panels in the field, and schedules service in real time, thereby minimizing the periods of low throughput from each panel and the associated penalty payouts.
- A media publisher streams billions of click stream records from its online properties, aggregates and enriches the data with demographic information about users, and optimizes content placement on its site, delivering relevancy and better experience to its audience.
- An online gaming company collects streaming data about player-game interactions, and feeds the data into its gaming platform. It then analyzes the data in real-time, offers incentives and dynamic experiences to engage its players.

Source : https://aws.amazon.com/streaming-data/

Who is using Stream Processing?

Streaming Data Use cases

- Algorithmic Trading, Stock Market Surveillance,
- Smart Patient Care
- Monitoring a production line
- Supply chain optimizations
- Intrusion, Surveillance and Fraud Detection
- Most Smart Device Applications: Smart Car, Smart Home ..
- Smart Grid — (e.g. load prediction and outlier plug detection see Smart grids, 4 Billion events, throughout in range of 100Ks)
- Traffic Monitoring, Geofencing, Vehicle, and Wildlife tracking — e.g. TFL London Transport Management System
- Sports analytics — Augment Sports with real-time analytics
- Context-aware promotions and advertising
- Computer system and network monitoring
- Predictive Maintenance, (e.g. Machine Learning Techniques for Predictive Maintenance)
- Geospatial data processing

Source : https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Usage of Stream Processing

Pravin Y Pawar

Uses of Stream Processing

- Example Applications
  - ✓ Fraud detection applications monitors the usage of credit cards for detection of unexpected patterns
  - ✓ Stock trading system looks for the opportunities of stock values and execute the trades based on those patterns
  - ✓ Manufacturing systems keeps an eye on the assembly / manufacturing process for the defective or malfunctioning machine
  - ✓ Intelligence systems can keep track of activities of aggressors and raise alarm if something unusual is noticed about them
  - ✓ Etc.

Concepts : Designing Data Intensive Applications by Martin Kleppmann

Other Applications

Complex Event Processing CEP

- Developed in 90’s
- Can specify rules to search for certain patterns of events in streams
- Can use programming constructs or GUIs to specify the conditions for patterns
- Outcomes can be displayed visually or alerts can be raised when condition is fulfilled
- Outcomes are complex event in nature, hence is the name
- Continuous queries model is used for identifying patterns
  - ✓ Query is always running
  - ✓ Events flows through it

- Examples
  - ✓ Esper
  - ✓ IBM Infosphere
  - ✓ TIBCO StreamBase
  - ✓ Oracle CEP

Other Applications (2)

Stream Analytics

- Recent application area
- Difference between CEP and Streaming Analytics diminishing
- More oriented towards aggregations and metrics over large number of events
  - ✓ Measuring rate at which system heart-beat messages are transmitted
  - ✓ Calculation rolling averages for certain periods
  - ✓ Comparing the statistics about two or more streams
- Usually windows of time are used
- Techniques used can produce exact outcomes or probabilistic outcomes

- Examples
  - ✓ Apache Storm
  - ✓ Spark Streaming
  - ✓ Flink
  - ✓ Apache Samza

Other Applications (3)

Materialized Views

- Materialized views are kind of dependent applications like caches, search index, data warehouses etc.
- When something changes in underlying systems like databases, all dependents needs to be updated as well
- Can be achieved by tracking what all things are changing in the applications and derived apps can be updated based on tracked changes

Other Applications (4)

Stream Searching

- CEP mostly monitors multiple event together, correlation between the incoming events
- Sometimes individual events also needs to be monitored for certain complex conditions
- For example,
  - ✓ User is interested to get informed when a particular property matching his requirement and budget is listed for sale / rent
  - ✓ User is interested to get informed about the availability of the products which are currently unavailable for sale

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Sources of Streaming Data

Pravin Y Pawar

Streaming Data Sources

1. Operational Monitoring

- Operational Monitoring
  - ✓ Ex: Tracking the performance of the physical systems that power the Internet
  - ✓ Temperature of the processor, speed of the fan and the voltage draw of their power supplies , state of their disk drives (processor load, network activity, and storage access times) etc…
  - ✓ Data is collected and aggregated in real time

Source : Adapted from Real-Time Analytics , Byron Ellis

Streaming Data Sources (2)

2. Web Analytics

- Web Analytics - track activity on a website
- Circulation numbers of a newspaper, the number of unique visitors for a webpage etc
- Structure of websites and their impact on various metrics of interest – A/B Testing – done in sequence vs parallel
- Applications – aggregation for billing – product recommendations (NetFlix)

Streaming Data Sources (3)

3. Online Advertising

- Contributes immensely for the large and growing portion of traffic.
- A visitor arrives at a website via a modern advertising exchange
- Call is made to a number of bidding agencies (perhaps 30 or 40 at a time), who place bids on the page view in real time by the exchange
- Auction is run, and the advertisement from the winning party is displayed
- (1) to (3) happens while the rest of the page is loading; the elapsed time is less than about 100 milliseconds
- A page with several advertisements needs simultaneous effort for each advertisements

Streaming Data Sources (4)

4. Social Media

- Sources like twitter, facebook, Instagram, google+, flickr….
- Data is collected and disseminated in real time
- “In 2011, Twitter users in New York City received information about an earth-quake outside of Washington, D.C. about 30 seconds before the tremors struck New York itself “
- Data highly unstructured and some form of “natural language” data that must be parsed, processed, and not well understood by automated systems

Streaming Data Sources (5)

5. Mobile data and IoT

- Measure the physical world
- Wristband that measures sleep activity
- Trigger an automated coffee maker when the user gets a poor night’s sleep and needs to be alert the next day.

Thank You!

In our next session: Difference between Batch Processing and Stream Processing

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Generalized Streaming Data Architecture

Pravin Y Pawar

Streaming Data Systems

Defined again

- … are layered systems that rely on several loosely coupled systems
  - Helps in achieving high availability
  - Helps in managing the system
  - Helps in maintaining the cost under control

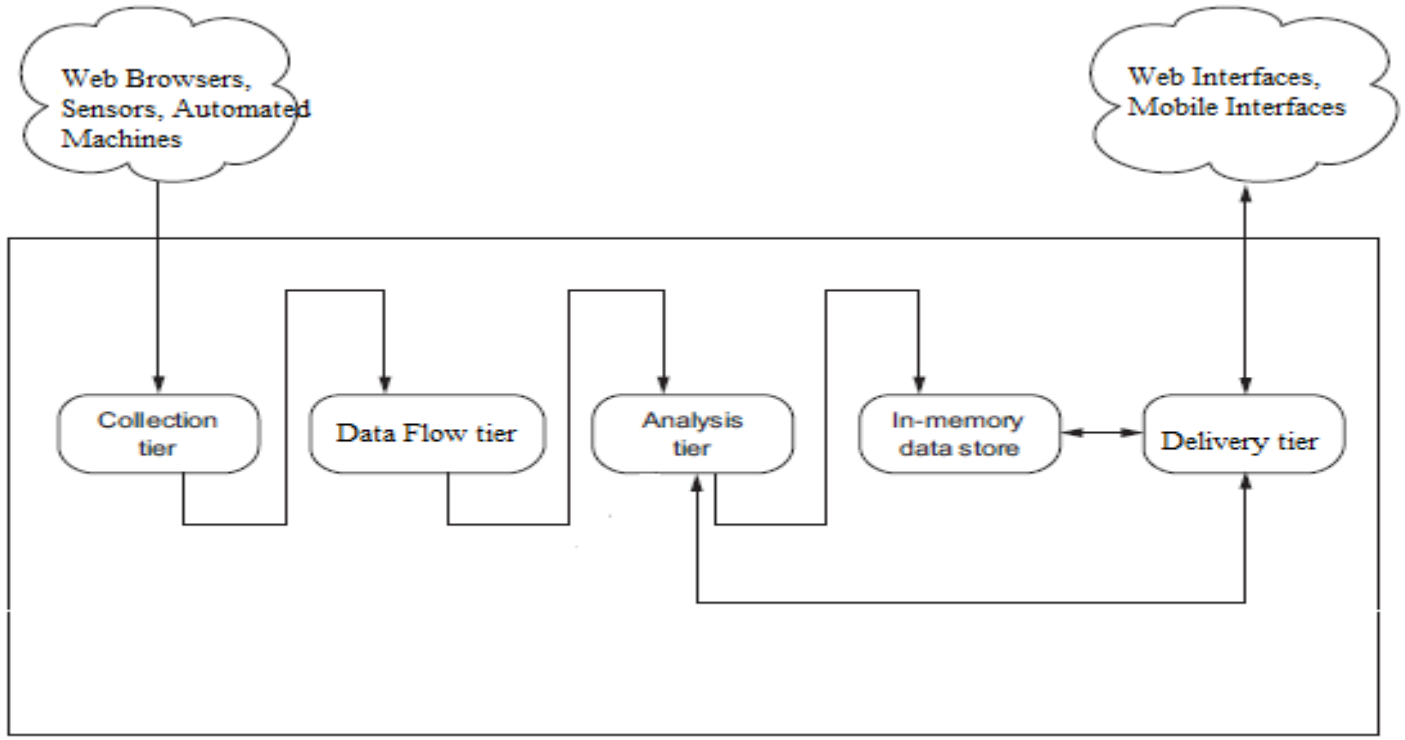
- All subsystems / components can reside on individual physical servers or can be co hosted on the single or more than one servers
- Not all components to be present in every system

Contents adapted from Real-Time Analytics , Byron Ellis

Streaming Data System Components

- Streaming Data System Architecture Components
  - Collection
  - Data Flow
  - Processing
  - Storage
  - Delivery

Generalized Architecture



Altered version, original concept : Andrew G. Psaltis

Architecture Components (1)

Collection System

- Mostly communication over TCP/IP network using HTTP
- Websites log data was the initial days use case
- W3C standard log data format was used
- Newer formats like JSON, AVRO, Thrift are available now

- Collection happens at specialized servers called edge servers
- Collection process is usually application specific
- New servers integrates directly with data flow systems
- Old servers may or may not integrate directly with data flow systems

Architecture Components (2)

Data Flow Tier

- Separation between collection tier and processing layer is required
  - Rates at which these systems works are different
  - What if one of system is not able to cope with another system?

- Required intermediate layer that takes responsibility of
  - accepting messages / events from collection layer
  - providing those messages / events to processing layer

- Real time interface to data layer for both producers and consumers of data
- Helps in guaranteeing the “at least once” semantics

Architecture Components (3)

Processing / Analytics Tier

- Based on “data locality” principle
- Move the software / code to a the location of data
- Rely on distributed processing of data
- Framework does the most of the heavy lifting of data partitioning, job scheduling, job managing
- Available Frameworks
  - Apache Storm
  - Apache Spark (Streaming)
  - Apache Kafka Streaming etc

Architecture Components (4)

Storage Tier

- In memory or permanent
- Usually in memory as data is processed once
- But can have use cases where events / outcomes needs to be persisted as well

- NoSQL databases becoming popular choice for permanent storage
  - MongoDB
  - Cassandra

- But usage varies as per the use case, still no database that fits all use cases

Architecture Components (5)

Delivery Layer

- Usually web based interface
  - Now a days mobile interfaces are becoming quite popular
  - Dashboards are built with streaming visualizations that gets continuously updated as underlying events are processed
  - HTML + CSS + Java script + Websockets can be used to create interfaces and update them
  - HTML5 elements can be used to render interfaces
  - SVG, PDF formats used to render the outcomes

- Monitoring / Alerting Use cases
- Feeding data to downstream applications

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Lambda Architecture

Pravin Y Pawar

Lambda Architecture

Defined

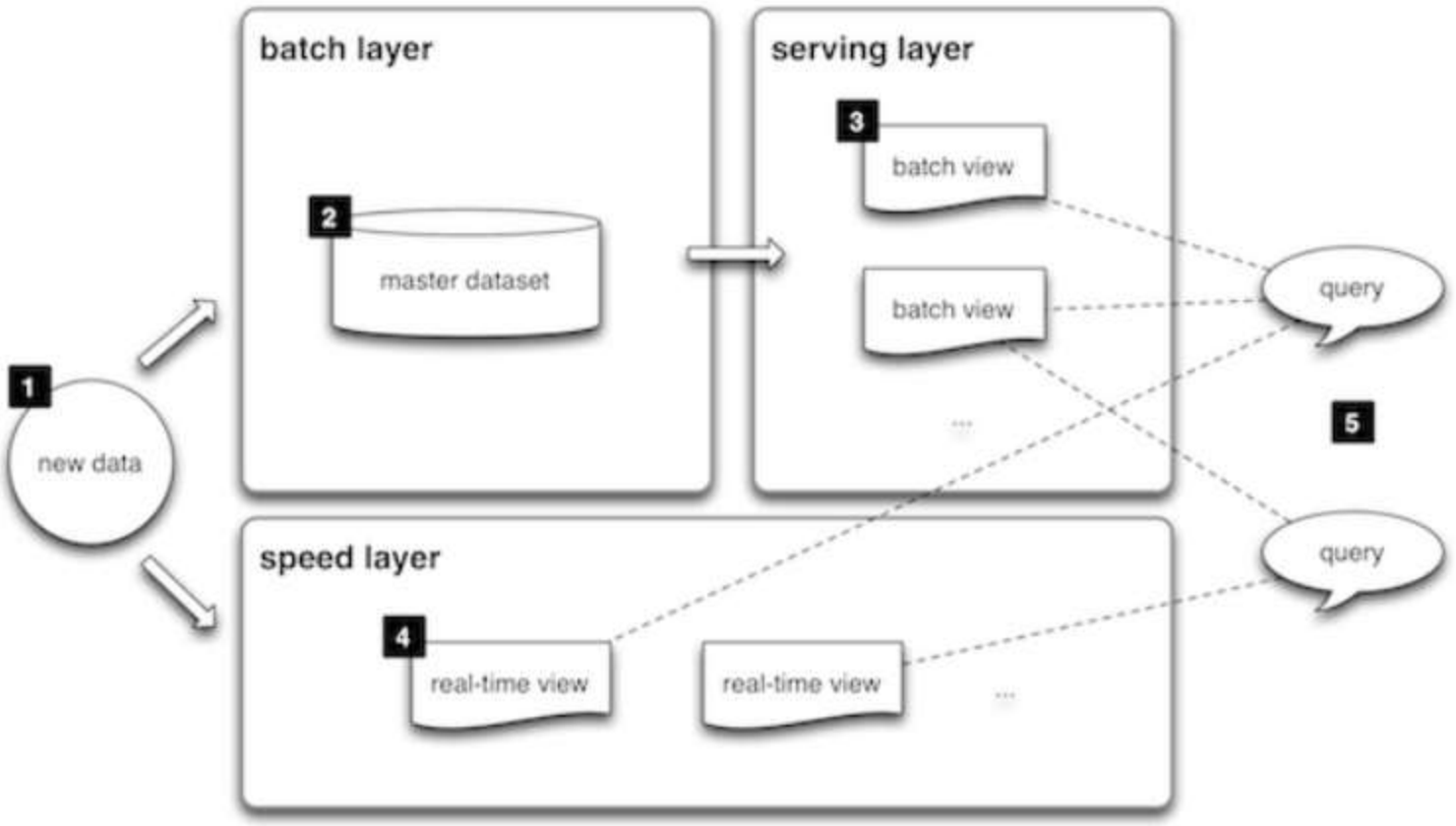
- Proposed by Nathan Marz based on his experience working on distributed data processing systems at Backtype and Twitter
- A generic, scalable and fault-tolerant data processing architecture

- Lambda Architecture
  - aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes
  - being able to serve a wide range of workloads and use cases
  - in which low-latency reads and updates are required.

- The resulting system should be linearly scalable, and it should scale out rather than up.

Lambda Architecture (2)

Block Diagram



Source : http://lambda-architecture.net/

Lambda Architecture (3)

Basic Flow of Events

1. All data entering the system is dispatched to both the batch layer and the speed layer for processing.
2. The batch layer has two functions:
   (i) managing the master dataset (an immutable, append-only set of raw data)
   (ii) to pre-compute the batch views.
3. The serving layer indexes the batch views so that they can be queried in low-latency, ad-hoc way.
4. The speed layer compensates for the high latency of updates to the serving layer and deals with recent data only.
5. Any incoming query can be answered by merging results from batch views and real-time views.

Architectural Components (1)

Batch layer

- New data comes continuously, as a feed to the data system.
- It gets fed to the batch layer and the speed layer simultaneously.
- It looks at all the data at once and eventually corrects the data in the stream layer.
- Here we can find lots of ETL and a traditional data warehouse.
- This layer is built using a predefined schedule, usually once or twice a day.

- The batch layer has two very important functions:
  - To manage the master dataset
  - To pre-compute the batch views.

Source : https://databricks.com/glossary/lambda-architecture

Architectural Components (2)

Speed Layer (Stream Layer)

- This layer handles the data that are not already delivered in the batch view due to the latency of the batch layer.
- In addition, it only deals with recent data in order to provide a complete view of the data to the user by creating real-time views.

- Speed layer provides the outputs on the basis enrichment process and supports the serving layer to reduce the latency in responding the queries.
- As obvious from its name the speed layer has low latency because it deals with the real time data only and has less computational load.

Architectural Components (3)

Serving Layer

- The outputs from batch layer in the form of batch views and from speed layer in the form of near-real time views are forwarded to the serving layer.

- This layer indexes the batch views so that they can be queried in low-latency on an ad-hoc basis.

Applications of Lambda Architecture

- User queries are required to be served on ad-hoc basis using the immutable data storage.
- Quick responses are required and system should be capable of handling various updates in the form of new data streams.
- None of the stored records shall be erased and it should allow addition of updates and new data to the database.

Pros and Cons of Lambda Architecture

- Pros
  - Batch layer of Lambda architecture manages historical data with the fault tolerant distributed storage which ensures low possibility of errors even if the system crashes.
  - It is a good balance of speed and reliability.
  - Fault tolerant and scalable architecture for data processing.

- Cons
  - It can result in coding overhead due to involvement of comprehensive processing.
  - Re-processes every batch cycle which is not beneficial in certain scenarios.
  - A data modelled with Lambda architecture is difficult to migrate or reorganize.

Source : https://towardsdatascience.com/a-brief-introduction-to-two-data-processing-architectures-lambda-and-kappa-for-big-data

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Kappa Architecture

Pravin Y Pawar

Bad thing about Lambda Architecture

- The problem with the Lambda Architecture is that maintaining code that needs to produce the same result in two complex distributed systems is exactly as painful as it seems like it would be.

- Programming in distributed frameworks like Storm and Hadoop is complex. Inevitably, code ends up being specifically engineered toward the framework it runs on. The resulting operational complexity of systems implementing the Lambda Architecture is the one thing that seems to be universally agreed on by everyone doing it.

Interesting read : https://www.oreilly.com/ideas/questioning-the-lambda-architecture

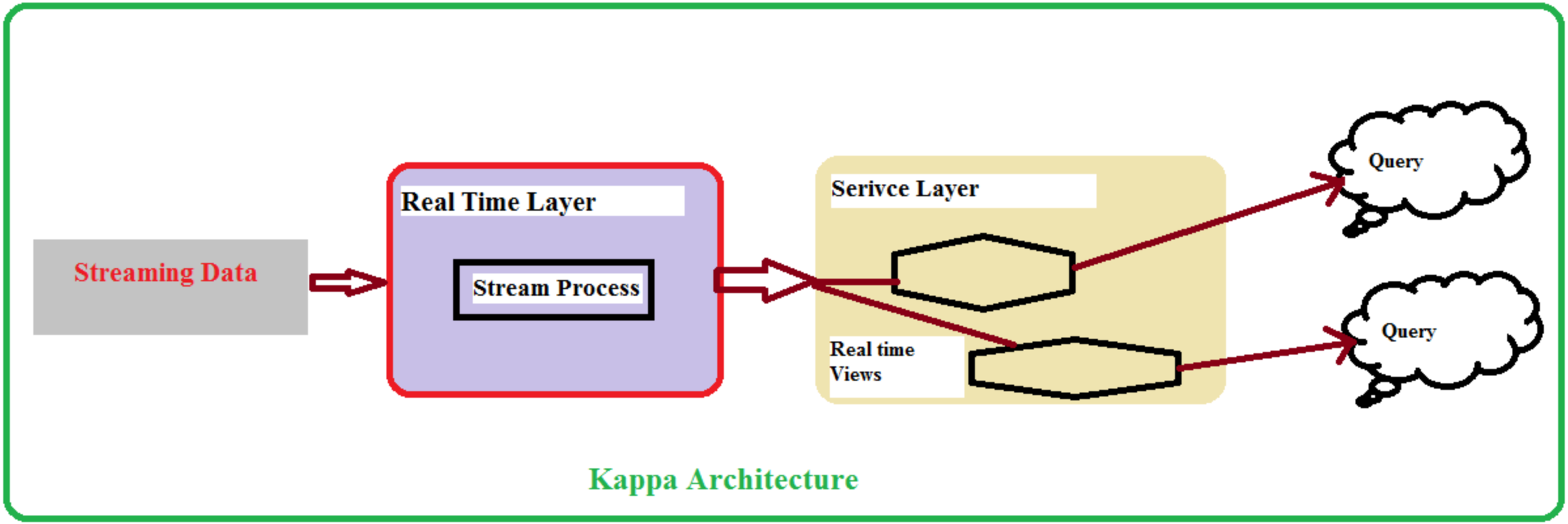
Kappa Architecture

Defined

- First described by Jay Kreps at Linkedin.
- It focuses on only processing data as a stream.
- It is not a replacement for the Lambda Architecture, except for where your use case fits.
- For this architecture, incoming data is streamed through a real-time layer and the results of which are placed in the serving layer for queries.

Kappa Architecture (2)

Block Diagram



Kappa Architecture (3)

- The idea is to handle both real-time data processing and continuous reprocessing in a single stream processing engine.
- This requires that the incoming data stream can be replayed (very quickly), either in its entirety or from a specific position.
- If there are any code changes, then a second stream process would replay all previous data through the latest real-time engine and replace the data stored in the serving layer.
- This architecture attempts to simplify by only keeping one code base rather than manage one for each batch and speed layers in the Lambda Architecture.
- In addition, queries only need to look in a single serving location instead of going against batch and speed views.

Interesting read : https://www.talend.com/blog/2017/08/28/lambda-kappa-real-time-big-data-architectures/

Pros and Cons of Kappa architecture

- Pros
  - Kappa architecture can be used to develop data systems that are online learners and therefore don’t need the batch layer.
  - Re-processing is required only when the code changes.
  - It can be deployed with fixed memory.
  - It can be used for horizontally scalable systems.
  - Fewer resources are required as the machine learning is being done on the real time basis.

- Cons
  - Absence of batch layer might result in errors during data processing or while updating the database that requires having an exception manager to reprocess the data or reconciliation.

Interesting read : https://towardsdatascience.com/a-brief-introduction-to-two-data-processing-architectures-lambda-and-kappa-for-big-data-

Real Time System Characteristics

BITS Pilani
Pilani Campus

Pravin Y Pawar

innovate achieve lead

BITS Pilani

Pilani Campus

BA ZC420, Real Time Analytics
Lecture No. 1.4

Agenda

- Distinguishing Features of Streaming Data
  - Data always in motion
  - Data structuring
  - Data Cardinality
- Features of Real-Time Architecture
  - High Availability
  - Low Latency
  - Horizontal Scalability

Distinguishing Features of Streaming Data

- Data always in motion
  - Streaming data
    - getting generated continuously
    - Always flowing
  - Two critical requirements
    - Collection system should be robust
    - Processing should be able to keep pace with collection
  - Solutions
    - Horizontal Scalability
    - Algorithmic handling of streaming data

Distinguishing Features of Streaming Data - II

- Data Structuring
  - Loosely structured
  - Various data sources
    - structured , unstructured data
    - Forming a joint schema is difficult
    - For example, social media streams
  - Young , evolving projects
    - Adds many dimensions to the data
    - Collect as much as data possible to make interesting analysis

Distinguishing Features of Streaming Data - III

- Data Cardinality
  - Number of unique values in data
  - Very few values appears often, many are very sparce

  - Challenges with Streaming data
  - Processing
    - Streaming data can be processed only once
    - Difficult to identify state of data
    - Batch processing on processed data can be used for estimation

  - Storage
    - Memory requirements are high while processing data
    - Linear amount of space required for storing state information

Features of Real-Time Architecture

- High Availability
  - Key distinguishing factor from batch / BI systems
  - Very critical for collection, flow and processing systems
  - Two Approaches
  - Distribution
    - Use multiple physical servers to distribute the load
  - Replication
    - Write to several machines
    - Master-slave configuration
      - Automatic failover
    - Master less configuration
      - Recovery is difficult in case of failure

Features of Real-Time Architecture - II

- Low Latency
  - Time taken to service a request
  - Streaming systems latency
    - Time taken to process the event from the moment it entered the system
  - Many streaming systems works in batches
    - Micro batching – processing in very small batches, milli seconds
    - Collection systems bothers about first definition of latency
    - Flow and processing components bother about second one
  - Tradeoff between speed and safety
    - If data can be safely lost, latency can be very small
    - If not, needs to live with lower limit of latency

Features of Real-Time Architecture - III

- Horizontal Scalability
  - Adding more physical servers to a clusters
  - Needs to care about amount of coordination required between the systems
  - Use of partitioning technique
  - Use principle of data locality – move program to data

Reference

- Real-Time Analytics , Byron Ellis
  - Chapter 1 : Introduction to Streaming Data
  - Chapter 2 : Designing Real-Time Streaming Architecture

Thank You!

In our next session: Streaming Data Systems Components

Thank You!

In our next session: Kappa Architecture

Thank You!

In our next session: Lambda Architecture

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

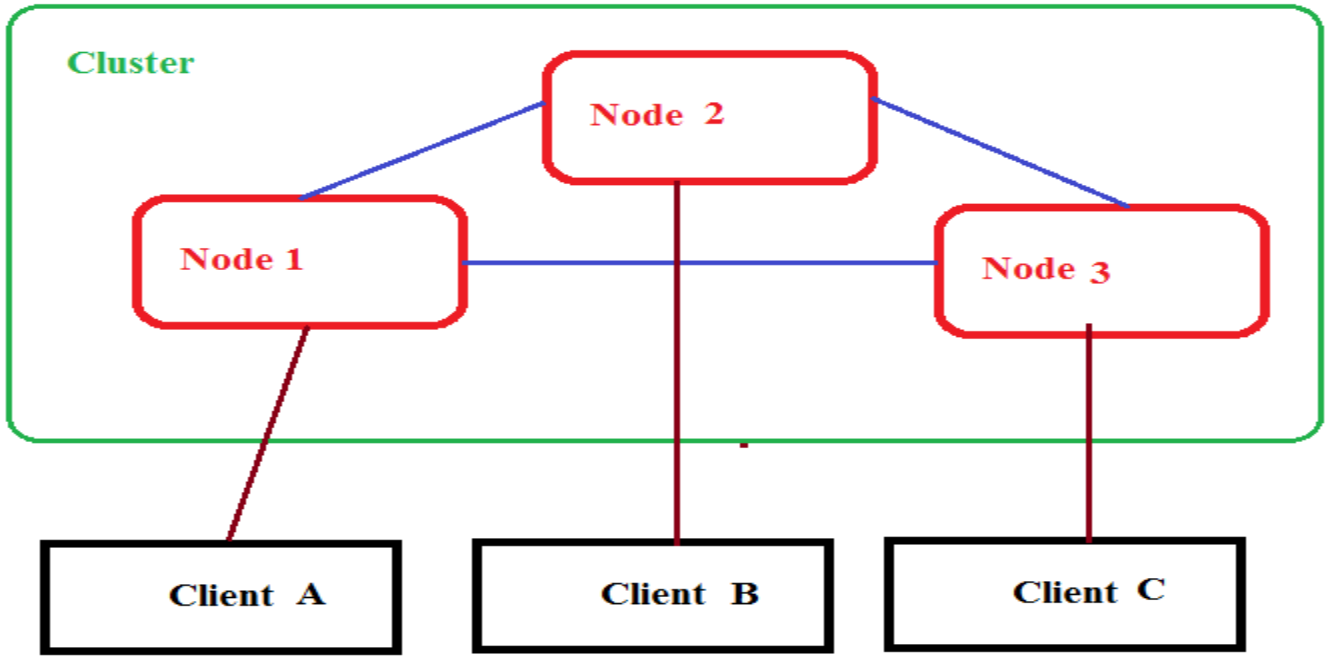
Service Configuration and Coordination Systems

Pravin Y Pawar

Distributed Applications

Overview

- Run on multiple systems in a network at a given time simultaneously
- Systems coordinates among themselves to carry out task in fast and efficient manner
- Collection of systems is “Cluster”
- System is termed as “Node”
- Client communicates with the systems in cluster



Motivation for Configuration and Coordination System

Need

- Applications needs to share the metadata and state

- Metadata is configuration information
  - ✓ Example, location of various database shards in server
- System state is data used to coordinate the application
  - ✓ Example, server currently acting as master

Motivation for Configuration and Coordination System (2)

Need (Continued)

- Managing metadata and state is too difficult
- Often leads to incorrect server behavior
- Causes unaccepted delays in processing or missed the processing entirely
- Needs a system-wide service that correctly and reliably implements distributed configuration and coordination

Distributed State Management

Challenges

- Unreliable network connections
- Clock Synchronization
- Consistency in applications state

Distributed State Management (2)

Unreliable network connections

- Networks are unreliable
  - Latency varies from time to time
  - Bandwidth changes
  - Connections are lost

- Split brain problem
  - Loss of connectivity between two groups of systems
  - Some amount of state is inaccessible
  - Disallow changes to distributed state until connectivity restored back
  - Allow one partition to remain functional while degrading the capabilities of other partition

Distributed State Management(3)

Clock Synchronization

- Depending upon type of application, synchronization needs to be precise
- Hardware clocks in servers not perfect and tend to drift over time
- May lead to disordering in the events

Distributed State Management(4)

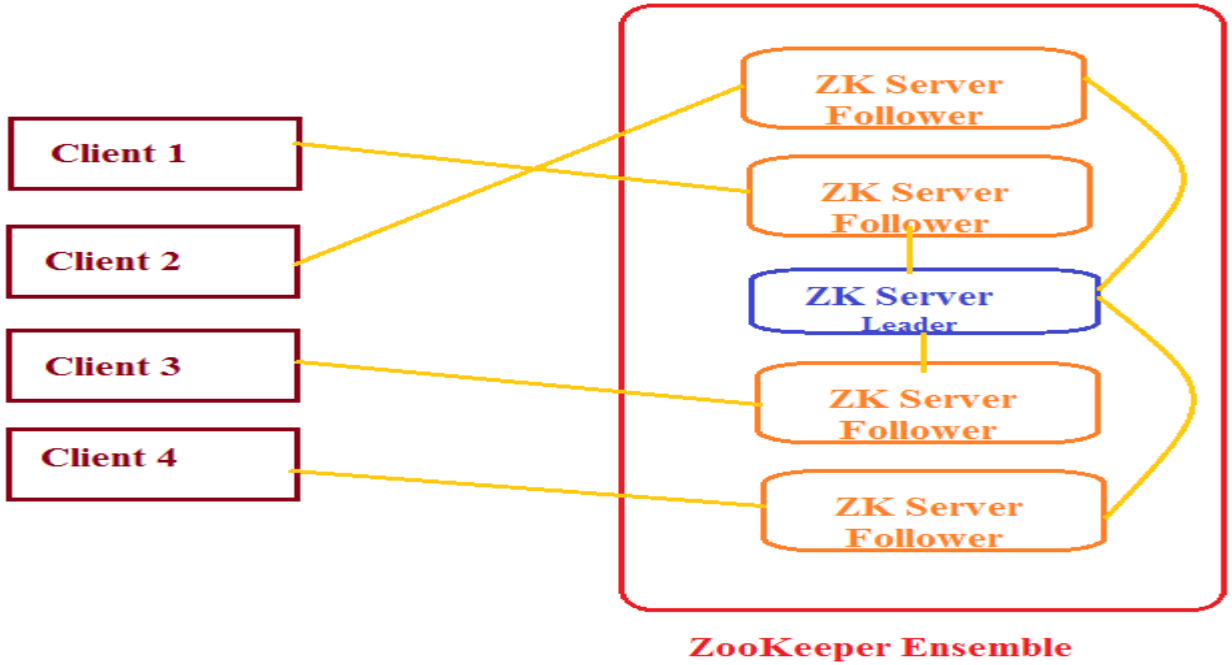
Consistency in applications state

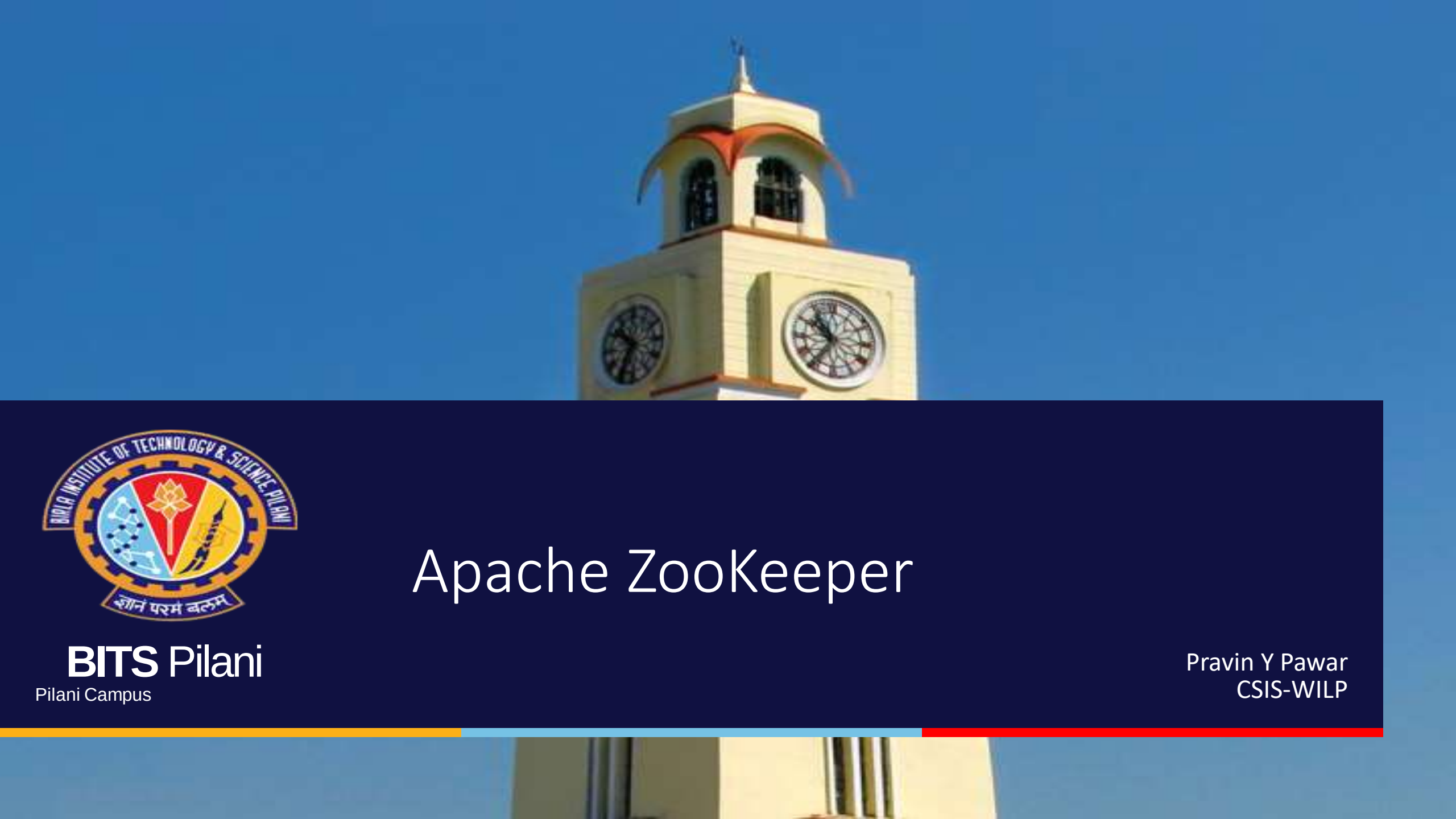
- State in distributed system has to be consistent in any case of failure
- Paxos algorithm or Multi-Paxos helps in maintaining the state
- But notoriously difficult to implement
- Recommended to use systems that already has implementations for the same

Example System

Apache ZooKeeper

- Designed at Yahoo!
- Distributed co-ordination service to manage large set of hosts
- Simple architecture and API to manage a service in a distributed environment
- Standard for organized service used by Hadoop, HBase, and other distributed frameworks





Apache ZooKeeper

BITS Pilani
Pilani Campus

Pravin Y Pawar
CSIS-WILP

Agenda

- Apache ZooKeeper Fundamentals
- ZooKeeper Workflow
- ZooKeeper CLI
- ZooKeeper Java APIs

Apache ZooKeeper

- Overview
  - Designed at Yahoo!
  - Distributed co-ordination service to manage large set of hosts
  - Simple architecture and API to manage a service in a distributed environment
  - Standard for organized service used by Hadoop, HBase, and other distributed frameworks

Apache ZooKeeper Fundamentals

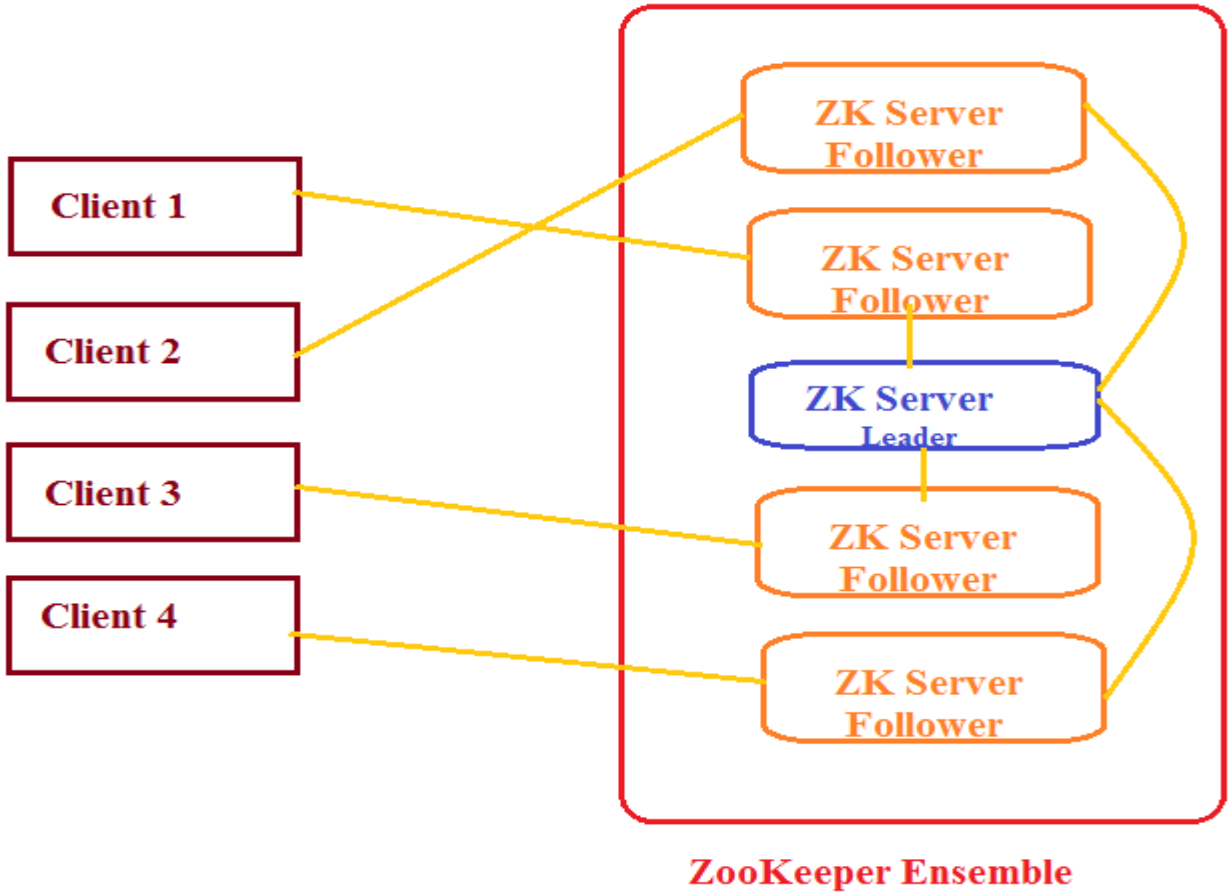
- ZooKeeper Services
  - Naming service
    - ✓ Helps in identifying the nodes in a cluster by name
  - Configuration management
    - ✓ Provides latest and up-to-date configuration information of system
  - Cluster management
    - ✓ Managing the addition / deletion of nodes and their status
  - Leader election
    - ✓ Helps in electing a node as leader for coordination purpose
  - Locking and synchronization service
    - ✓ Helps in automatic fail over recovery
  - Highly reliable data registry
    - ✓ Makes data available even when one or more nodes are down

Apache ZooKeeper Fundamentals

- Benefits of ZooKeeper
  - Simple architecture of distributed coordination process
  - Synchronization
    - ✓ Helps in coordination between the server processes
  - Serialization
    - ✓ Ensure your application runs consistently
  - Reliability
    - ✓ As no single point of failure in system
  - Atomicity
    - ✓ Data transfer either succeed or fail completely, but no transaction is partial.

Apache ZooKeeper Fundamentals

- Architecture



Apache ZooKeeper Fundamentals

- Architectural Components
  - Client
    - ✓ Node in distribution application, tries to connect to server in ensemble.
    - ✓ Sends heartbeat message to server to maintain the connection
    - ✓ In response, server sends message back to client, otherwise client connects to other server
  - Server
    - ✓ Node in our ZooKeeper ensemble, provides all the services to clients
    - ✓ Gives acknowledgement to client to inform that the server is alive.
  - Ensemble
    - ✓ Cluster of ZooKeeper servers. The minimum number of nodes - 3.
  - Leader
    - ✓ Server node which performs automatic recovery if any of the connected node failed.
    - ✓ Leaders are elected on service startup.
  - Follower
    - ✓ Server node which follows leader instruction.

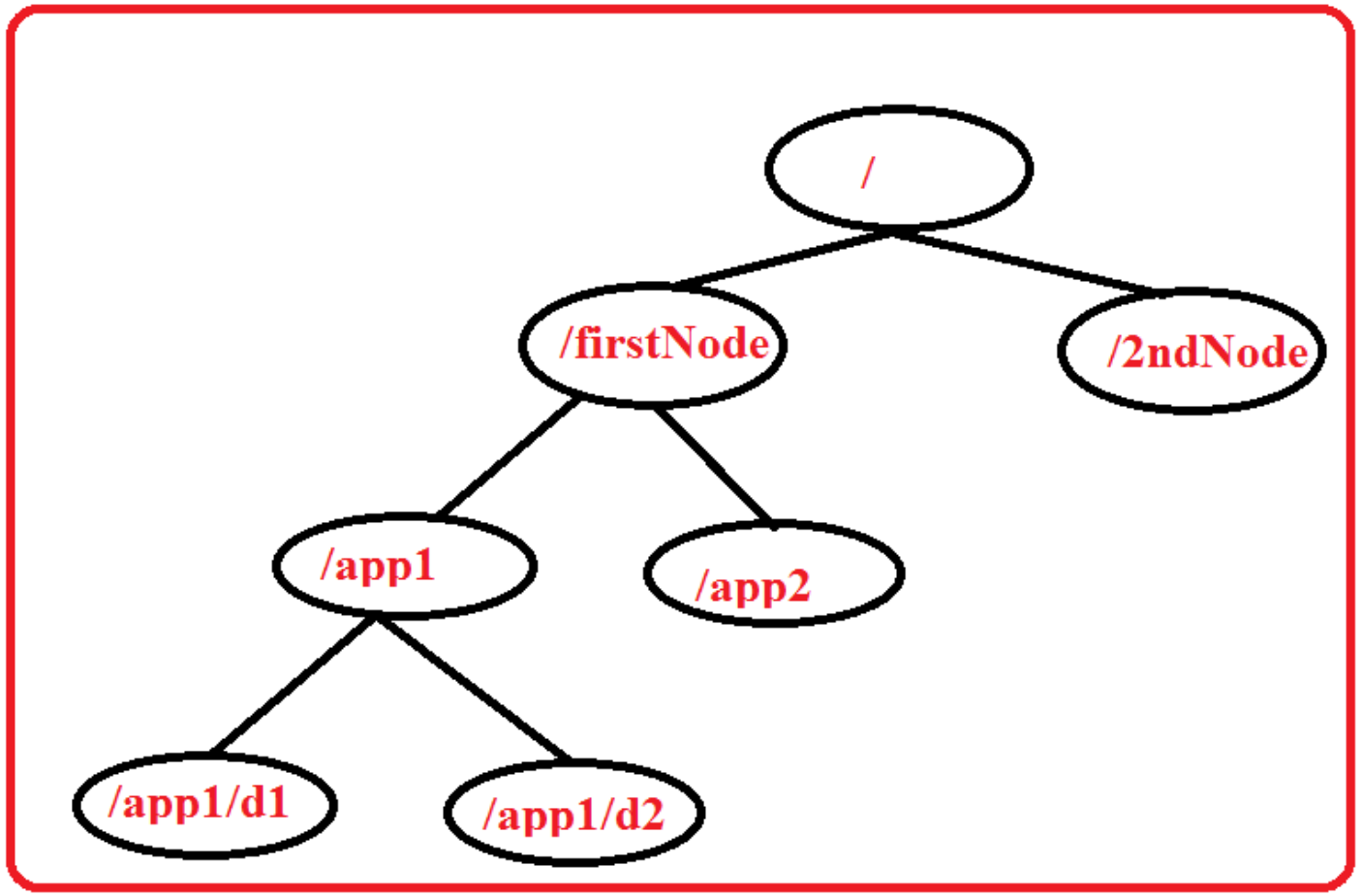
- https://zookeeper.apache.org/doc/r3.5.0-alpha/zookeeperOver.html

Apache ZooKeeper Fundamentals

- Hierarchical Namespace
  - ZooKeeper node is referred as znode.
  - Every znode is identified by a name and separated by a sequence of path .
  - Each znode can store upto 1MB of data
  - The main purpose of this structure is to store synchronized data and describe the metadata of the znode.
  - Every znode in the ZooKeeper data model maintains a stat structure.
  - A stat simply provides the metadata of a znode. It consists of Version number, Action control list , Timestamp, and Data length

Apache ZooKeeper Fundamentals

- Hierarchical Namespace(2)



Apache ZooKeeper Fundamentals

- Types of Znodes

  - ✓ Persistence znode
  - ✓ Ephemeral znode
  - ✓ Sequential znode

Apache ZooKeeper Fundamentals

- Persistence znode
  - ✓ is alive even after the client, which created that particular znode, is disconnected.
  - ✓ By default, all znodes are persistent.

- Ephemeral znode
  - ✓ are active until the client is alive.
  - ✓ When a client gets disconnected from the ZooKeeper ensemble, then the ephemeral znodes get deleted automatically.
  - ✓ are not allowed to have a children further
  - ✓ play an important role in Leader election

Apache ZooKeeper Fundamentals

- Sequential znode
  - can be either persistent or ephemeral
  - ZooKeeper sets the path of the znode by attaching a 10 digit sequence number to the original name.
  - For example, if a znode with path /myapp is created as a sequential znode, ZooKeeper will change the path to /myapp0000000001 and set the next sequence
  - number as 0000000002.
  - Sequential znodes play an important role in Locking and Synchronization.

Apache ZooKeeper

- Sessions
  - Requests in a session are executed in FIFO order.
  - Once a client connects to a server, the session will be established and a session id is assigned to the client.
  - The client sends heartbeats at a particular time interval to keep the session valid.
  - If the ZooKeeper ensemble does not receive heartbeats from a client for more than the period specified at the starting of the service, it decides that the client died.
  - When a session ends for any reason, the ephemeral znodes created during that session also get deleted.

Apache ZooKeeper

- Watches
  - Simple mechanism for the client to get notifications about the changes in the ZooKeeper ensemble.
  - Clients can set watches while reading a particular znode.
  - Watches send a notification to the registered client for any of the znode changes.
  - Znode changes are modification of data associated with the znode or changes in the znode's children.
  - Watches are triggered only once.
  - If a client wants a notification again, it must be done through another read operation.
  - When a connection session is expired, the client will be disconnected from the server and the associated watches are also removed.

ZooKeeper Workflow – Connection

- Connecting to Server

  - Once a ZooKeeper ensemble starts, it will wait for the clients to connect.
  - Clients will connect to one of the nodes in the ZooKeeper ensemble. It may be a leader or a follower node.
  - Once a client is connected, the node assigns a session ID to the particular client and sends an acknowledgement to the client.
  - If the client does not get an acknowledgment, it simply tries to connect another node in the ZooKeeper ensemble.
  - Once connected to a node, the client will send heartbeats to the node in a regular interval to make sure that the connection is not lost.

ZooKeeper Workflow – Read

- Reading data from znode

  - If a client wants to read a particular znode, it sends a read request to the node with the znode path and the node returns the requested znode by getting it from its own database.
  - For this reason, reads are fast in ZooKeeper ensemble.

ZooKeeper Workflow – Write

- Writing to znode
  - If a client wants to store data in the ZooKeeper ensemble, it sends the znode path and the data to the server.
  - The connected server will forward the request to the leader and then the leader will reissue the writing request to all the followers.
  - If only a majority of the nodes respond successfully, then the write request will succeed and a successful return code will be sent to the client.
  - Otherwise, the write request will fail. The strict majority of nodes is called as Quorum.

ZooKeeper Workflow – nodes failure

- Nodes in a ZooKeeper Ensemble
  - If we have a single node, then the ZooKeeper ensemble fails when that node fails. It contributes to "Single Point of Failure" and it is not recommended in a production environment.
  - If we have two nodes and one node fails, we don’t have majority as well, since one out of two is not a majority.
  - If we have three nodes and one node fails, we have majority and so, it is the minimum requirement. It is mandatory for a ZooKeeper ensemble to have at least three nodes in a live production environment.
  - If we have four nodes and two nodes fail, it fails again and it is similar to having three nodes. The extra node does not serve any purpose and so, it is better to add nodes in odd numbers, e.g., 3, 5, 7.
  - A write process is expensive than a read process in ZooKeeper ensemble, since all the nodes need to write the same data in its database. So, it is better to have less number of nodes than having a large number of nodes for a balanced environment.

ZooKeeper Command Line Interface

- CLI
  - ZooKeeper Command Line Interface is used to interact with the ZooKeeper ensemble for development purpose.
  - Can perform the following operation
    - ✓ Create znodes
    - ✓ Get data
    - ✓ Watch znode for changes
    - ✓ Set data
    - ✓ Create children of a znode
    - ✓ List children of a znode
    - ✓ Check Status
    - ✓ Remove / Delete a znode

ZooKeeper Java API

- Basics of ZooKeeper API
  - Application interacting with ZooKeeper ensemble is referred as ZooKeeper Client or simply Client.
  - Znode is the core component of ZooKeeper ensemble and ZooKeeper API provides a small set of methods to manipulate all the details of znode with ZooKeeper ensemble.
  - A client should follow the steps given below to have a clear and clean interaction with ZooKeeper ensemble.
    - ✓ Connect to the ZooKeeper ensemble. ZooKeeper ensemble assign a Session ID for the client.
    - ✓ Send heartbeats to the server periodically. Otherwise, the ZooKeeper ensemble expires the Session ID and the client needs to reconnect.
    - ✓ Get / Set the znodes as long as a session ID is active.
    - ✓ Disconnect from the ZooKeeper ensemble, once all the tasks are completed. If the client is in

ZooKeeper Java API (2)

- ZooKeeper Class
  - The central part of the ZooKeeper API is ZooKeeper class.
  - It provides options to connect the ZooKeeper ensemble in its constructor and has the following methods -
    - ✓ connect - connect to the ZooKeeper ensemble
    - ✓ create - create a znode
    - ✓ exists - check whether a znode exists and its information
    - ✓ getData - get data from a particular znode
    - ✓ setData - set data in a particular znode
    - ✓ getChildren - get all sub-nodes available in a particular znode
    - ✓ delete - get a particular znode and all its children
    - ✓ close - close a connection

Reference

- Apache ZooKeeper Documentation
  - https://zookeeper.apache.org/
  - https://zookeeper.apache.org/doc/r3.4.14/
  - https://www.tutorialspoint.com/zookeeper/zookeeper_api.htm
- Real-Time Analytics , Byron Ellis
  - Chapter 3 : Service Configuration and Coordination

Thank You!

In our next session: Data Flow Manager

Real Time System Characteristics

BITS Pilani
Pilani Campus

Pravin Y Pawar

innovate achieve lead

BITS Pilani

Pilani Campus

BA ZC420, Real Time Analytics
Lecture No. 1.4

Agenda

- Distinguishing Features of Streaming Data
  - Data always in motion
  - Data structuring
  - Data Cardinality
- Features of Real-Time Architecture
  - High Availability
  - Low Latency
  - Horizontal Scalability

Distinguishing Features of Streaming Data

- Data always in motion
  - Streaming data
    - getting generated continuously
    - Always flowing
  - Two critical requirements
    - Collection system should be robust
    - Processing should be able to keep pace with collection
  - Solutions
    - Horizontal Scalability
    - Algorithmic handling of streaming data

Distinguishing Features of Streaming Data - II

- Data Structuring
  - Loosely structured
  - Various data sources
    - structured , unstructured data
    - Forming a joint schema is difficult
    - For example, social media streams
  - Young , evolving projects
    - Adds many dimensions to the data
    - Collect as much as data possible to make interesting analysis

Distinguishing Features of Streaming Data - III

- Data Cardinality
  - Number of unique values in data
  - Very few values appears often, many are very sparce

  - Challenges with Streaming data
  - Processing
    - Streaming data can be processed only once
    - Difficult to identify state of data
    - Batch processing on processed data can be used for estimation

  - Storage
    - Memory requirements are high while processing data
    - Linear amount of space required for storing state information

Features of Real-Time Architecture

- High Availability
  - Key distinguishing factor from batch / BI systems
  - Very critical for collection, flow and processing systems
  - Two Approaches
  - Distribution
    - Use multiple physical servers to distribute the load
  - Replication
    - Write to several machines
    - Master-slave configuration
      - Automatic failover
    - Master less configuration
      - Recovery is difficult in case of failure

Features of Real-Time Architecture - II

- Low Latency
  - Time taken to service a request
  - Streaming systems latency
    - Time taken to process the event from the moment it entered the system
  - Many streaming systems works in batches
    - Micro batching – processing in very small batches, milli seconds
    - Collection systems bothers about first definition of latency
    - Flow and processing components bother about second one
  - Tradeoff between speed and safety
    - If data can be safely lost, latency can be very small
    - If not, needs to live with lower limit of latency

Features of Real-Time Architecture - III

- Horizontal Scalability
  - Adding more physical servers to a clusters
  - Needs to care about amount of coordination required between the systems
  - Use of partitioning technique
  - Use principle of data locality – move program to data

Reference

- Real-Time Analytics , Byron Ellis
  - Chapter 1 : Introduction to Streaming Data
  - Chapter 2 : Designing Real-Time Streaming Architecture
  - https://www.cs.cornell.edu/fbs/publications/viveLaDifference.pdf
  - https://omkarprabhu-98.github.io/2021/03/zab.html

BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Data Flow Manager

Pravin Y Pawar

Distributed Data Flows

Need

- Distributed state management is required in order to process the data in scalable way

- Distributed data flows consists of
  - ✓ Data collection
  - ✓ Data processing

- Systems for data flow management have matured over the years
  - ✓ In-house developments
  - ✓ Standard queuing systems like ActiveMQ
  - ✓ Services like Kafka and Flume

Distributed Data Flows systems

Requirement

- Systems should support
  - ✓ “At least once” delivery semantic
  - ✓ Solving “n+1” delivery problem

Data Delivery Semantic

- Three options for data delivery and processing
  - ✓ At most once delivery
  - ✓ At least once delivery
  - ✓ Exactly once delivery

At most once delivery semantic

- Systems used for monitoring purposes
- Important to inform the admins about the problems
- Not all data transmissions required
- Down-sample the data to improve performance
- Data loss is approximately known

Exactly once delivery semantic

- Financial systems or advertising systems
- Every message has to be delivered only once
- Data loss not affordable as it might be revenue loss
- Achieved through queuing systems like ActiveMQ, RabbitMQ
- Usually queue semantics implemented on server side

At least once delivery semantic

- Balance two extremes by providing reliable message delivery by pushing the message handling semantics to the consumer
- Consumers are free to implement message handling without bothered about other consumers
- Dependent on application logic and handled in application level only

The “n+1” problem

- In data processing pipeline, every time a new service or processing mechanism is added it must integrate with each of the other systems in place
  - ✓ Common antipattern
  - ✓ Handling interaction between systems becomes pain point

- Data flow systems standardizes the communication between the bus layer and each application , also it manages the physical flow of messages between systems
  - ✓ Allows any number of consumers and producers to communicate using common protocol

Example Systems

- High performance systems with sufficient scalability to support real time streaming
  - ✓ Apache Kafka
  - ✓ Flume by Cloudera

- Kafka
  - ✓ Directed towards users who are building applications from scratch, giving them the freedom to directly integrate a data motion system

- Flume
  - ✓ Design makes it well suited to environments that have existing applications that needs to be federated into single processing environment

Thank You!

In our next session : Streaming Data Processor