



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

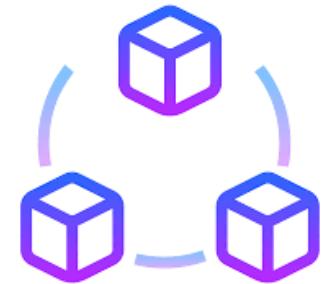


Course Design (cont...)



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



i

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

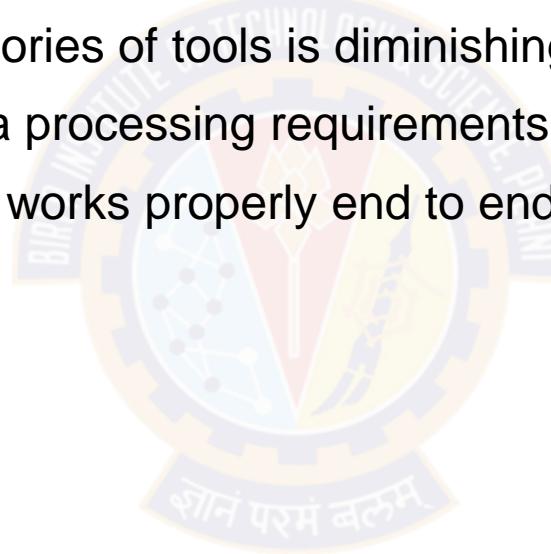


Source : The topics are adapted from “Designing Data Intensive Applications” by Martin Kleppmann

Data Systems

Described

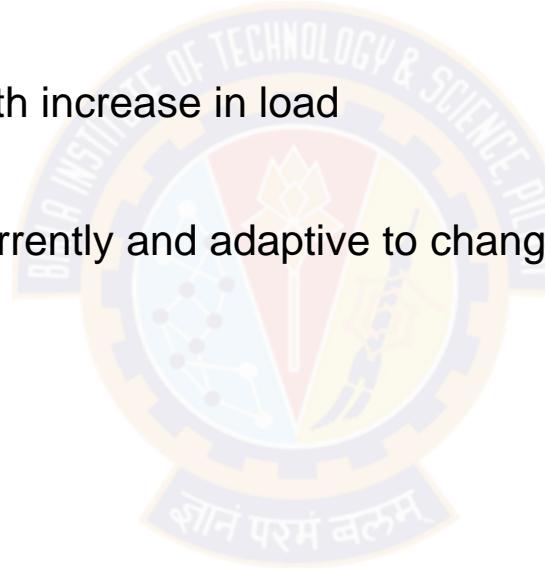
- Many different tools are integrated together for the data processing task
- Users are unaware about the seamless integration of tools / systems
- The boundaries between the categories of tools is diminishing
- No single tool can fit for all the data processing requirements
- Need to make sure this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



Non Functional Requirements for Data Systems

Three Requirements

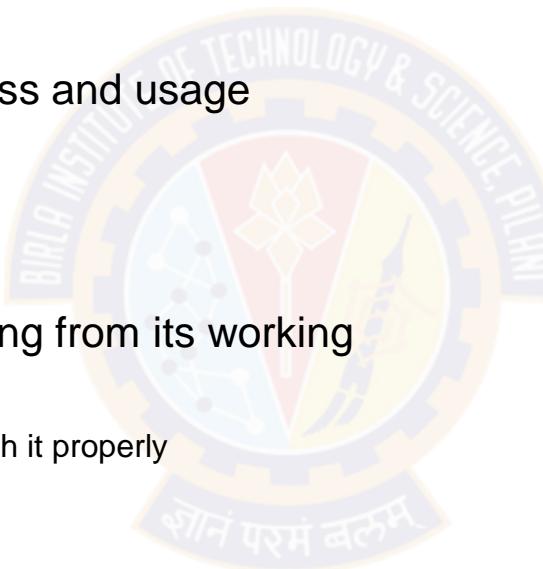
- Reliability
 - ✓ System should continue work correctly even in cases of failures
- Scalability
 - ✓ System should be able to cope with increase in load
- Maintainability
 - ✓ System should be able to work currently and adaptive to changes in future



Reliability

Described

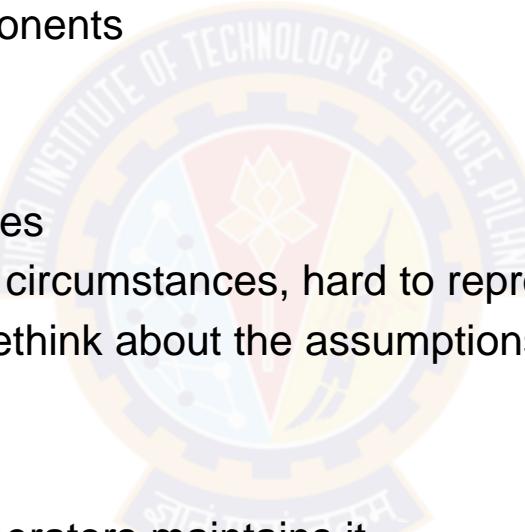
- System should continue to work correctly, even when things go wrong
 - ✓ Should carry out expected operation correctly
 - ✓ Should handle wrong user inputs
 - ✓ Should prevent unauthorized access and usage
- Fault
 - ✓ Things that can go wrong
 - ✓ One component of system deviating from its working
 - ✓ Fault-tolerant / Resilient
 - ❖ Which can anticipate fault and deal with it properly
- Failure
 - ✓ System as a whole stops providing services



Reliability(2)

Faults

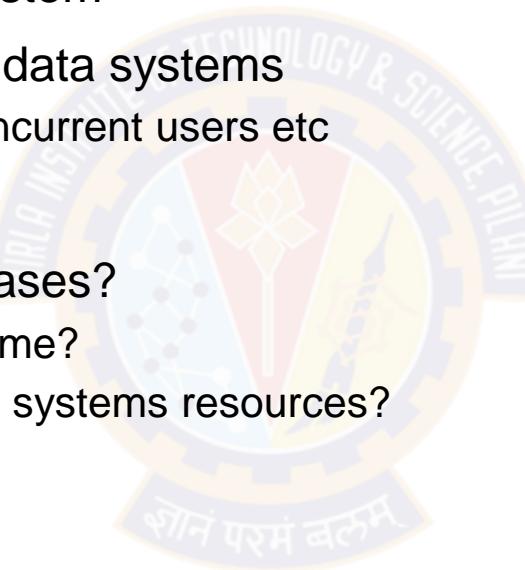
- Hardware Faults
 - ✓ Hard disk crash, RAM faults, Network issues etc
 - ✓ Add redundancy to individual components
- Software Faults
 - ✓ Software bugs, multi-threading issues
 - ✓ Happens under very unusual set of circumstances, hard to reproduce again
 - ✓ No straight away answer, need to rethink 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

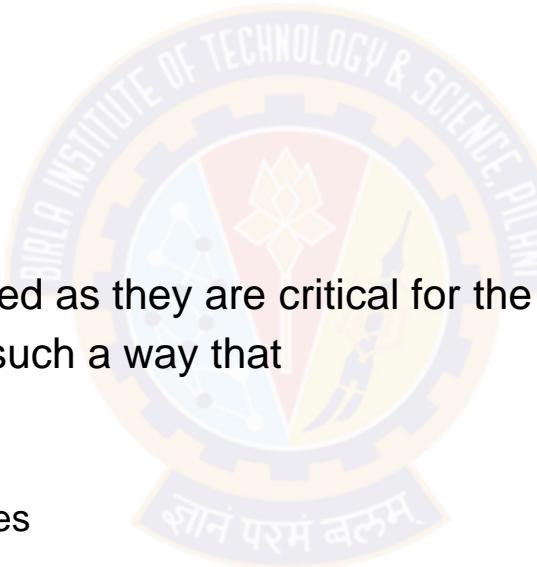
- Systems ability to cope with load
- Load impacts the performance of system
- Load defined differently for different data systems
 - ✓ Request per second, number of concurrent users etc
- How system reacts when load increases?
 - ✓ If the system resources are kept same?
 - ✓ What additions needs to be done in systems resources?



Maintainability

Described

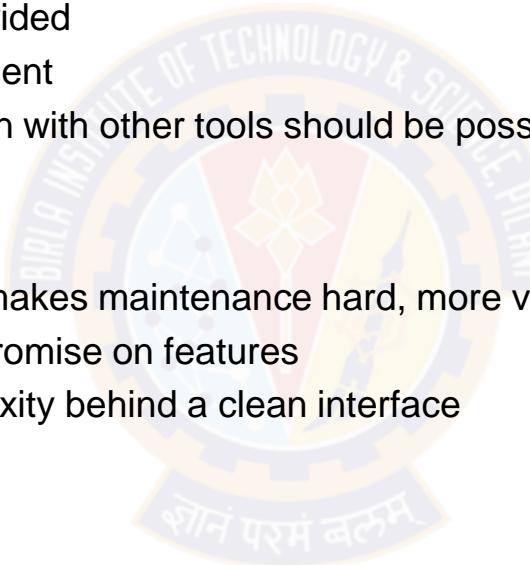
- Easy to write a software , but very difficult to maintain it
- Involves
 - ✓ Bug fixing
 - ✓ Keeping existing systems operational
 - ✓ Detecting root cause of failures
 - ✓ Adapting to new platforms
- Legacy systems needs to be maintained as they are critical for the business operations.
- Data Systems 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

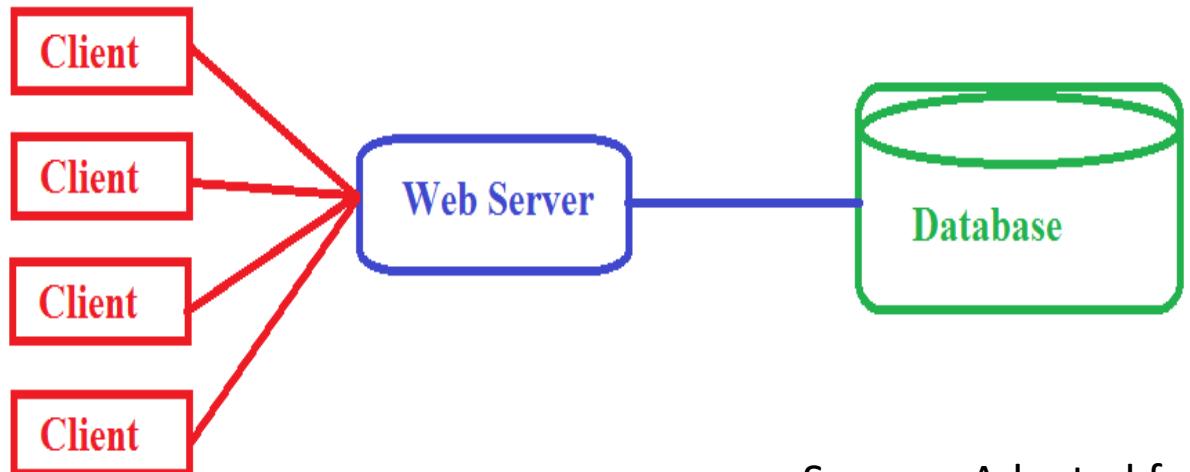
Scaling with Traditional Databases

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



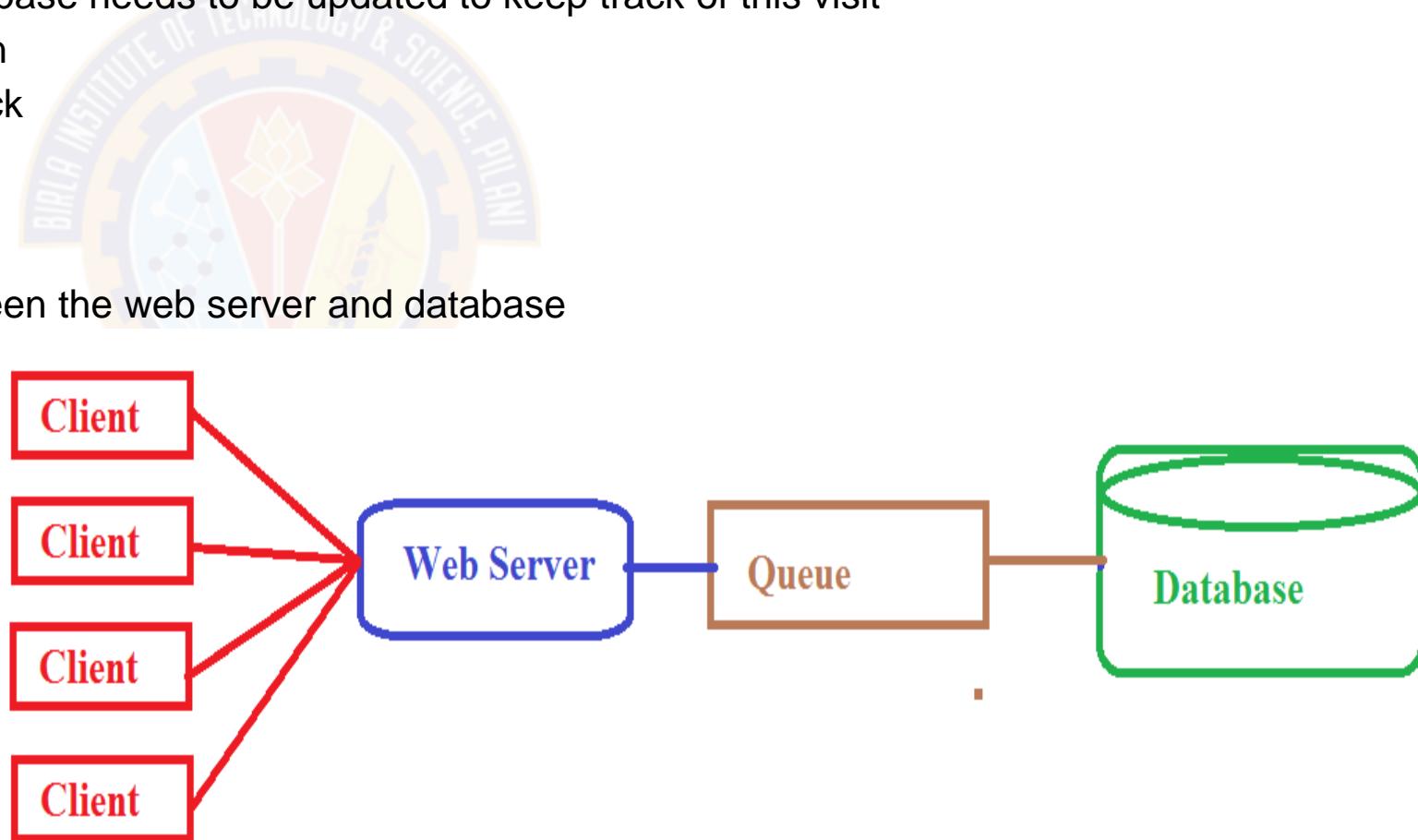
Column	Data Type
Id	Integer
User_ID	Integer
Page_URL	Varchar
Page_count	Long

Source : Adapted from Big Data by Nathan Marz

Scaling with intermediate layer

Using a queue

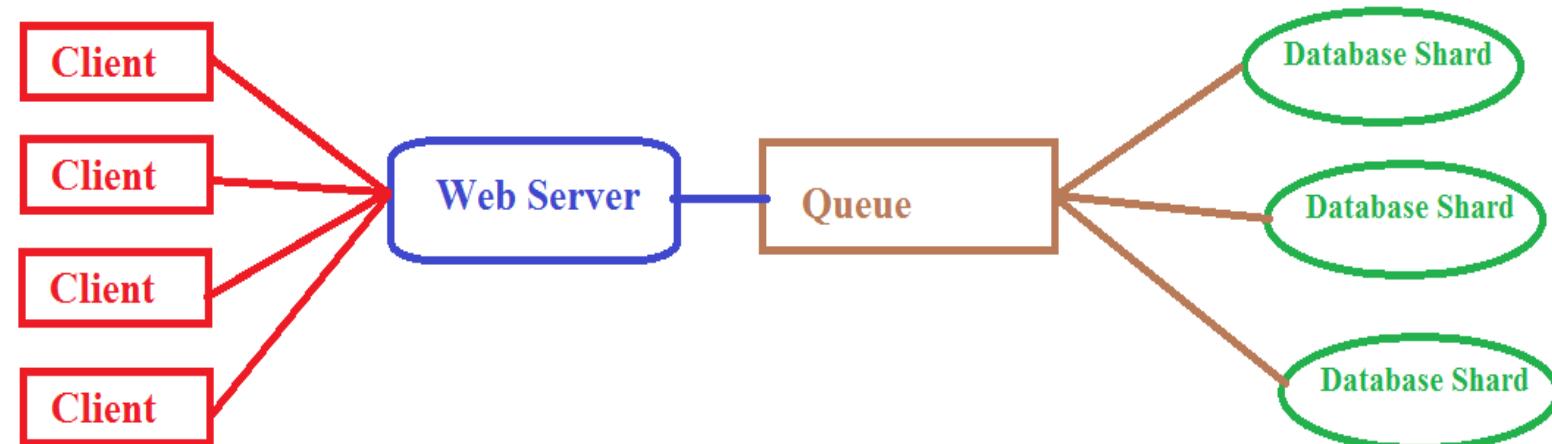
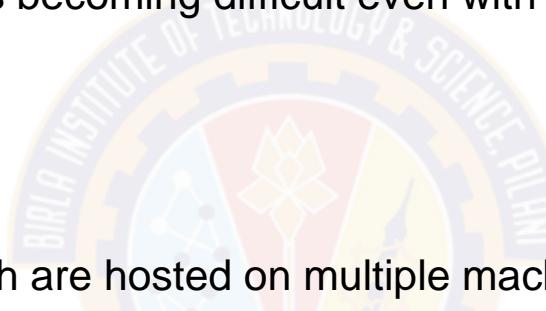
- Portal is very popular, lot of users visiting it
 - ✓ Many users are concurrently visiting the pages of portal
 - ✓ Every time a page is visited, database needs to be updated to keep track of this visit
 - ✓ Database write is heavy operation
 - ✓ Database write is now a bottleneck
- Solution
 - ✓ Use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

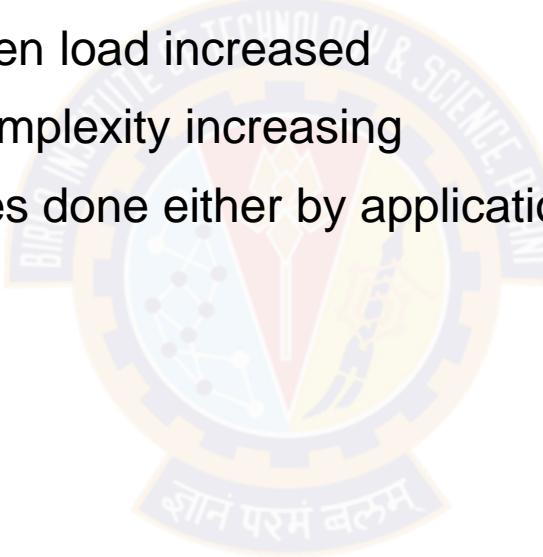
- Application is too popular
 - ✓ Users are using it very heavily, increasing the load on application
 - ✓ Maintaining the page view count is becoming difficult even with queue
- Solution
 - Use database partitions
 - ✓ Data is divided into partitions which are hosted on multiple machines
 - ✓ Database 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 not 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



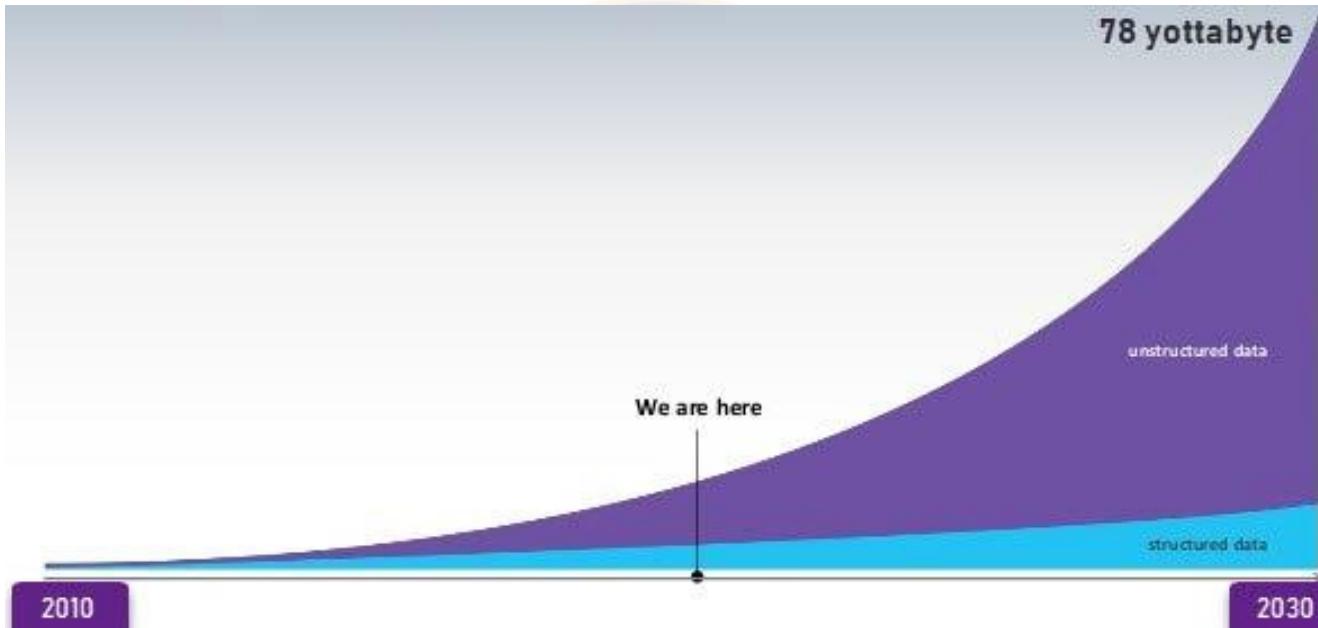
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Big Data Systems

Pravin Y Pawar

Data Growth

Data Generation

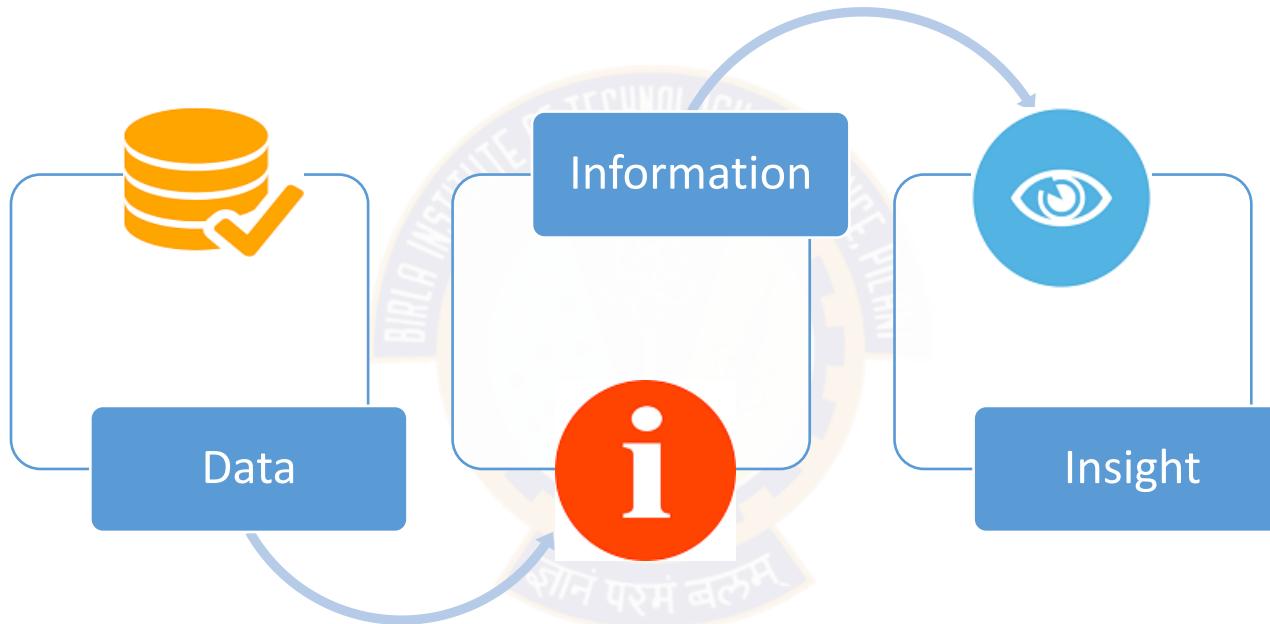


Source : <https://www.guru99.com/what-is-big-data.html>

Big Data Systems

What?

- New Paradigm for Data Analytics



Data Classification

Types of Data

- Structured

Databases



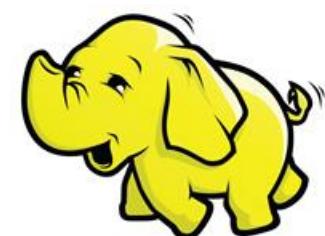
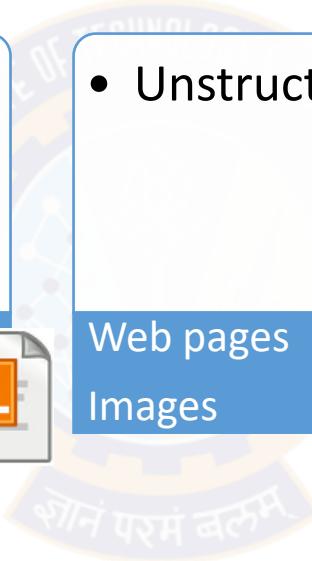
- Semi-structured

XML



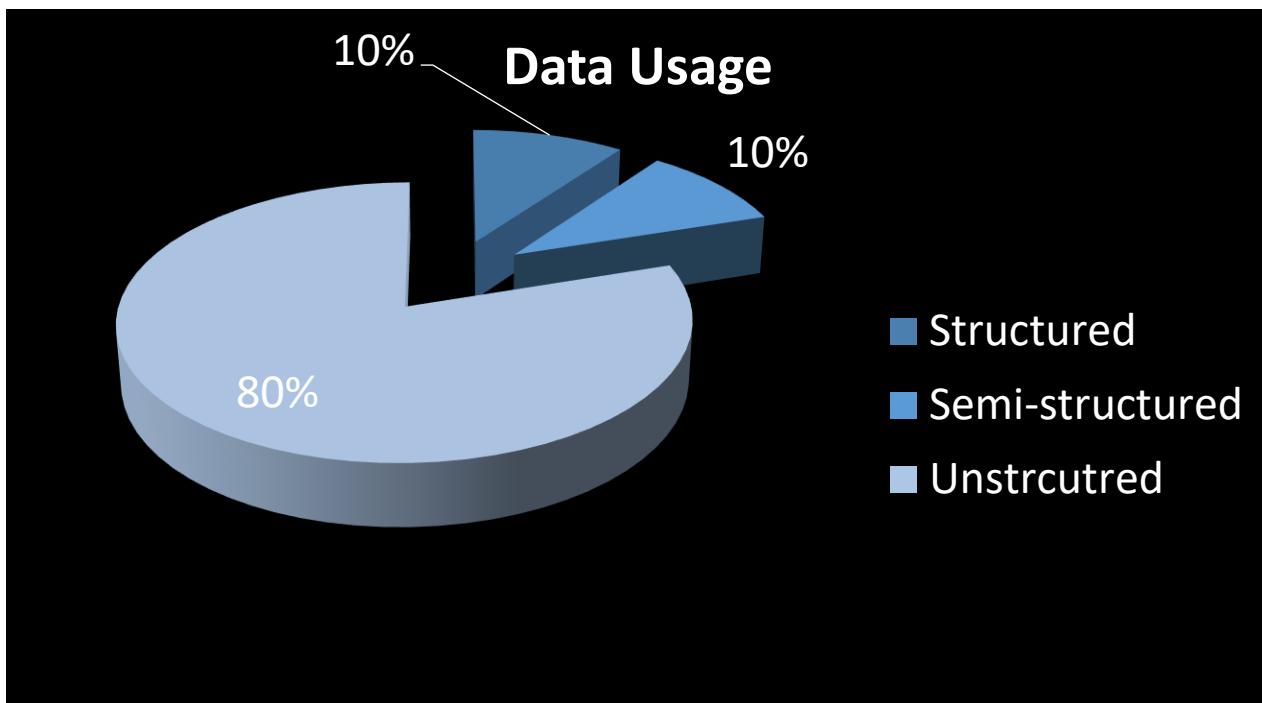
- Unstructured

Web pages
Images



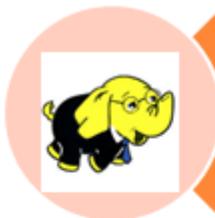
Data Usage Pattern

Usage Stats

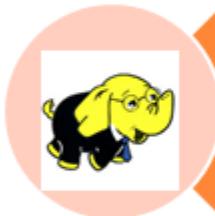


Big Data

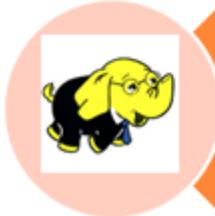
Defined



Anything beyond human and technical infrastructure needed to support storage, processing and analysis

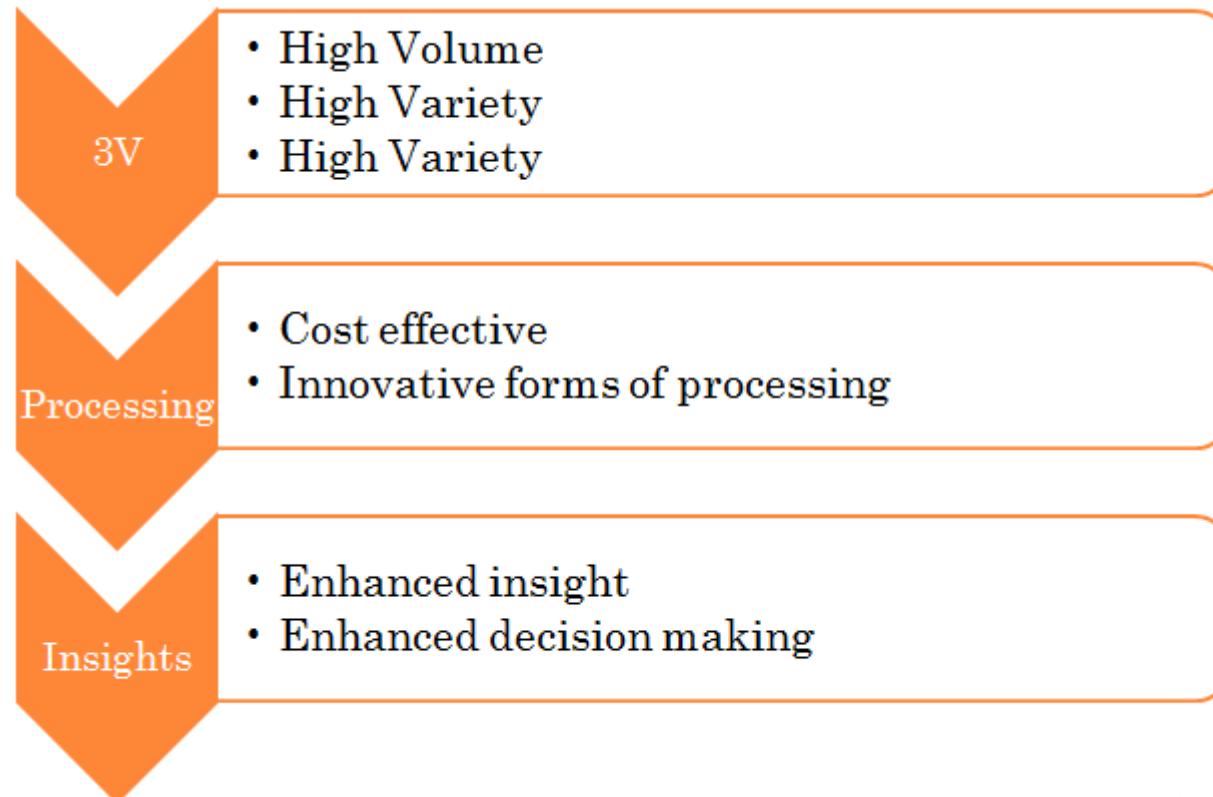


Today's BIG Data is tomorrow's NORMAL



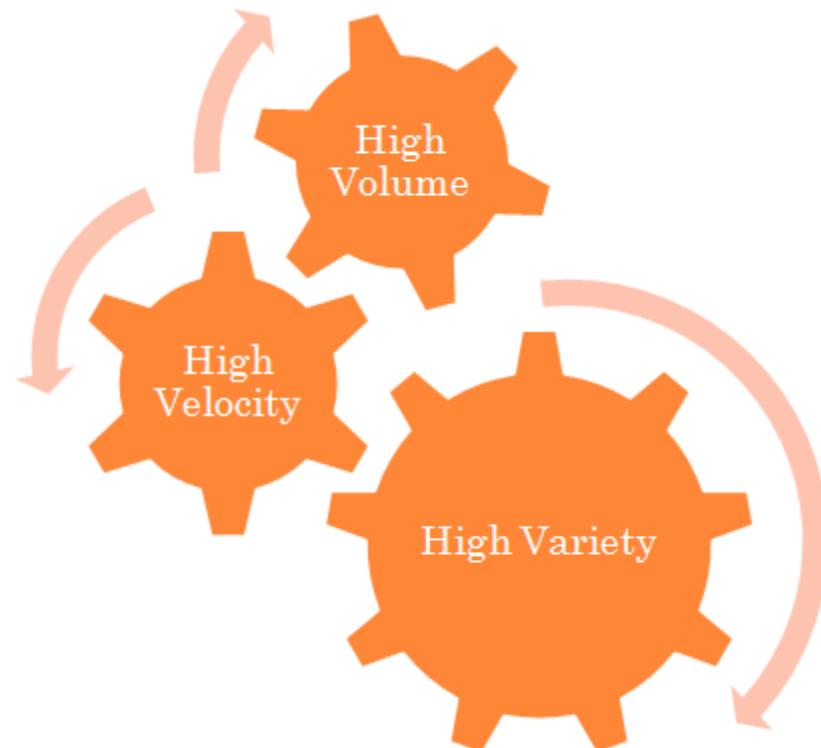
Terabytes or Petabytes or zettabytes of data

Big data – by Gartner



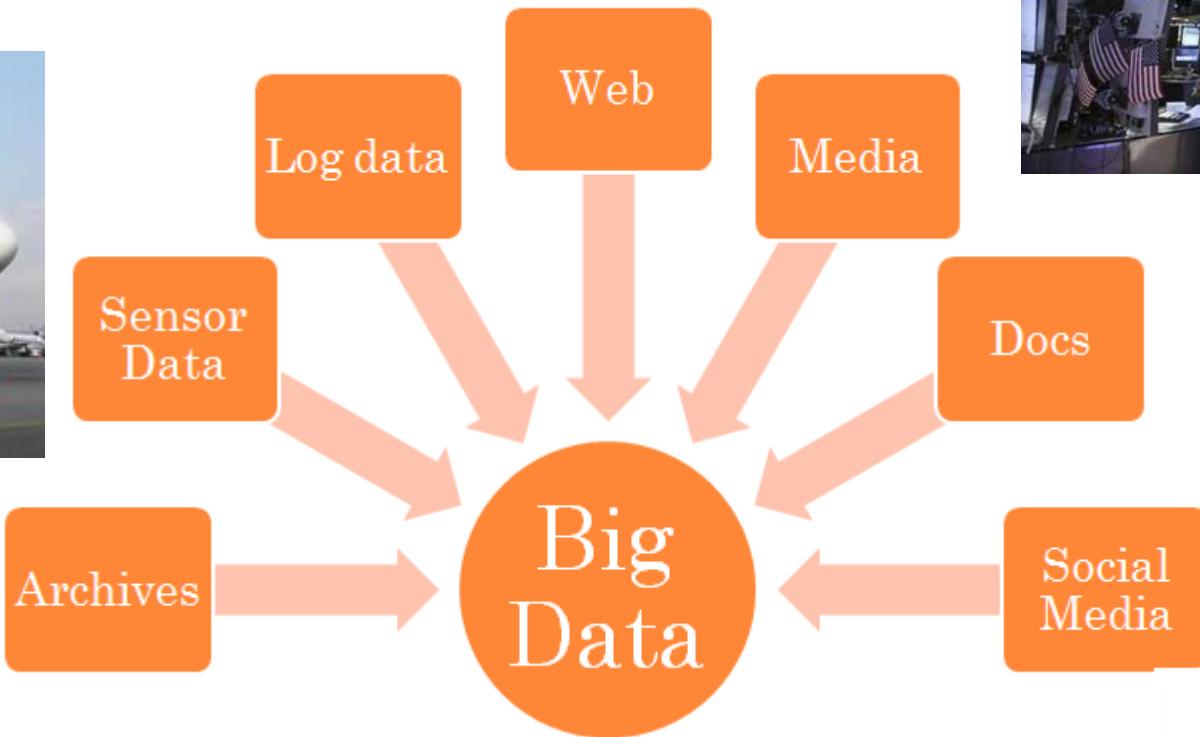
3 V's of Big Data

Volume, Velocity and Variety



Sources of Big Data

New Age Data Sources

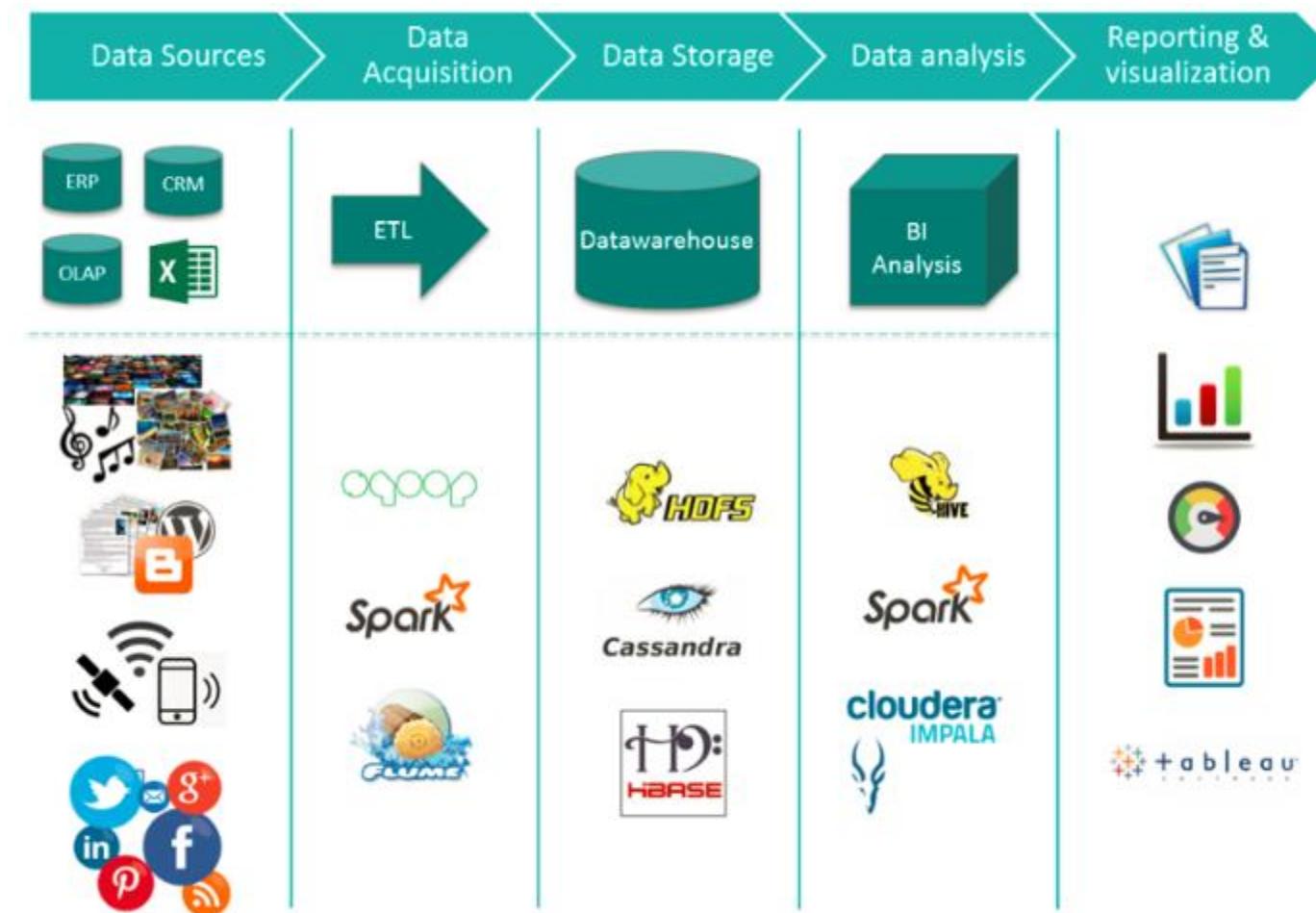


facebook

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

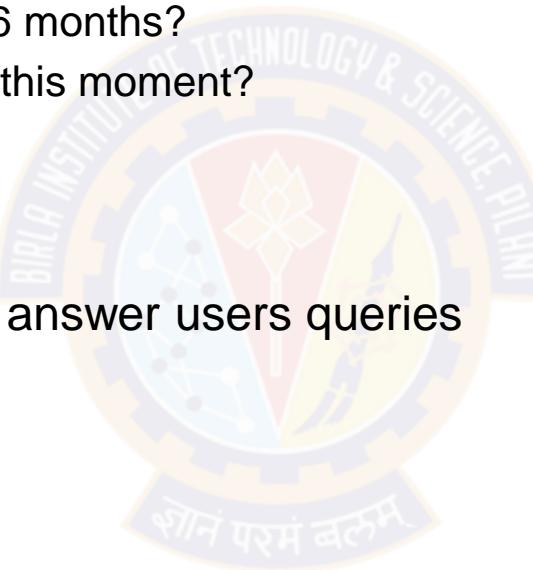
Desired Properties of Big Data Systems

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

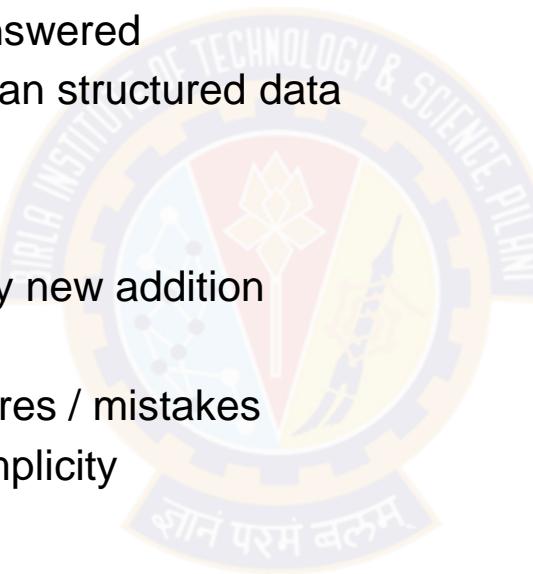
Data Model of Big Data Systems

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

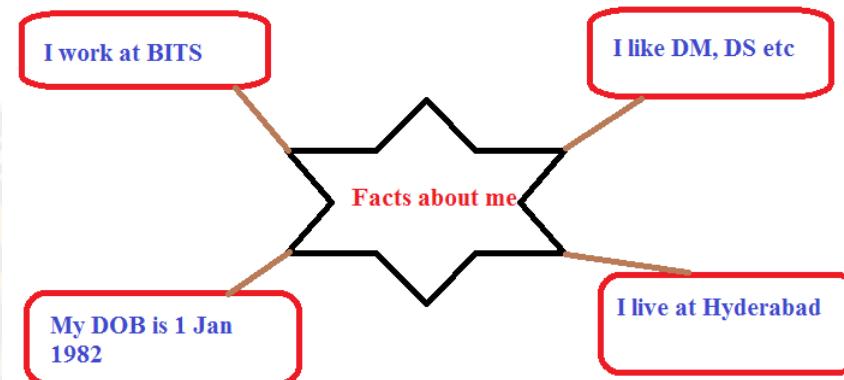
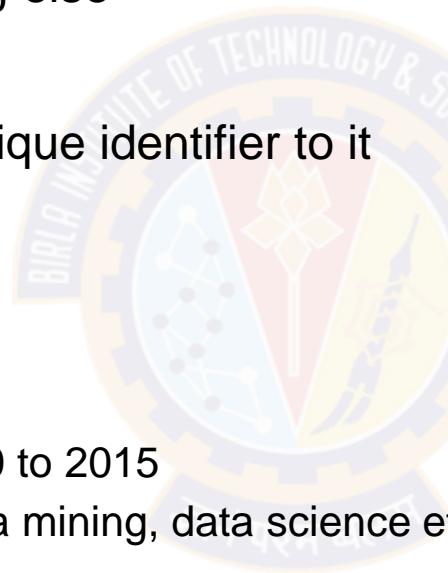


Source : Big Data , Nathan Marz

Fact based Model for Data

Facts

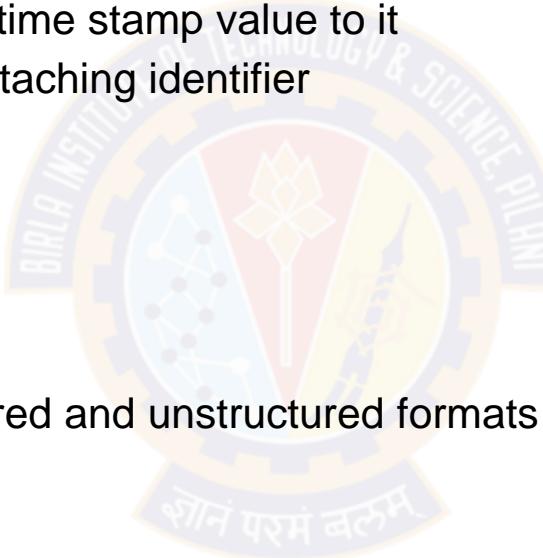
- Fundamental unit of data
- Data can not derived from anything else
- Fact is atomic and time stamped
- Can be made unique by adding unique identifier to it
- Example facts
 - ✓ I work for BITS Pilani
 - ✓ My DOB is 1 January 1982
 - ✓ I currently live in Hyderabad
 - ✓ I was living at Pune between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



Fact based Model for Data (2)

Benefits

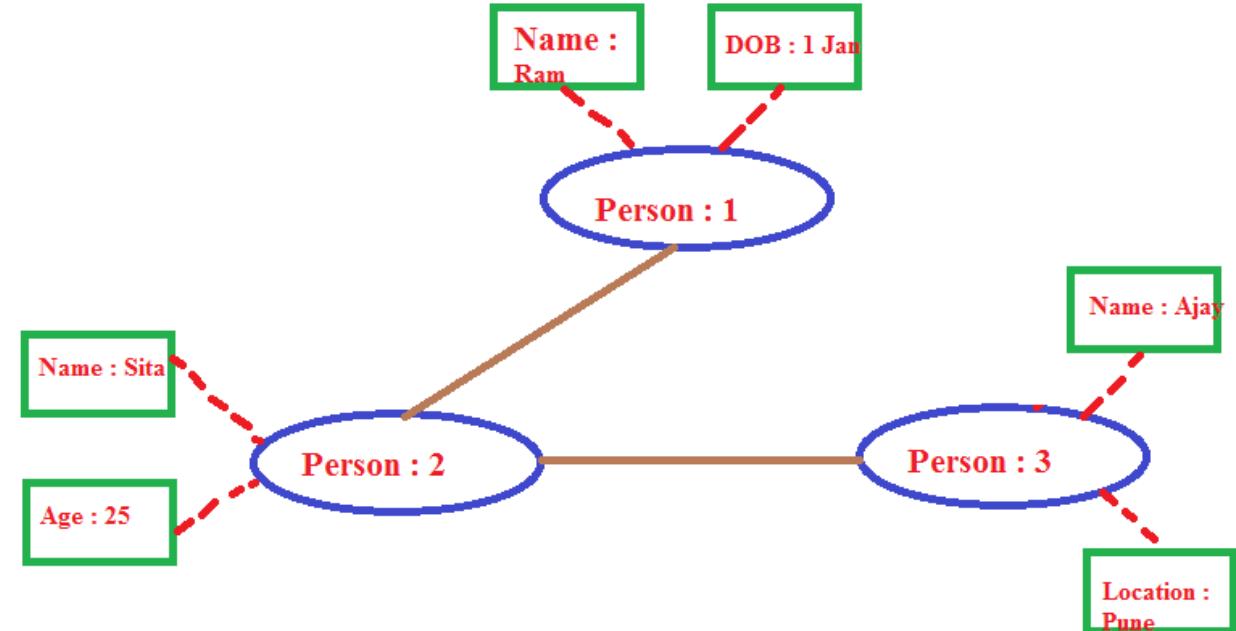
- Fact based model
 - ✓ Stores raw data as atomic facts
 - ✓ Makes facts immutable by adding time stamp value to it
 - ✓ Makes it uniquely identifiable by attaching identifier
- Benefits
 - ✓ Data is query able at any time
 - ✓ Data is tolerant to human errors
 - ✓ Data can be stored both in structured 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





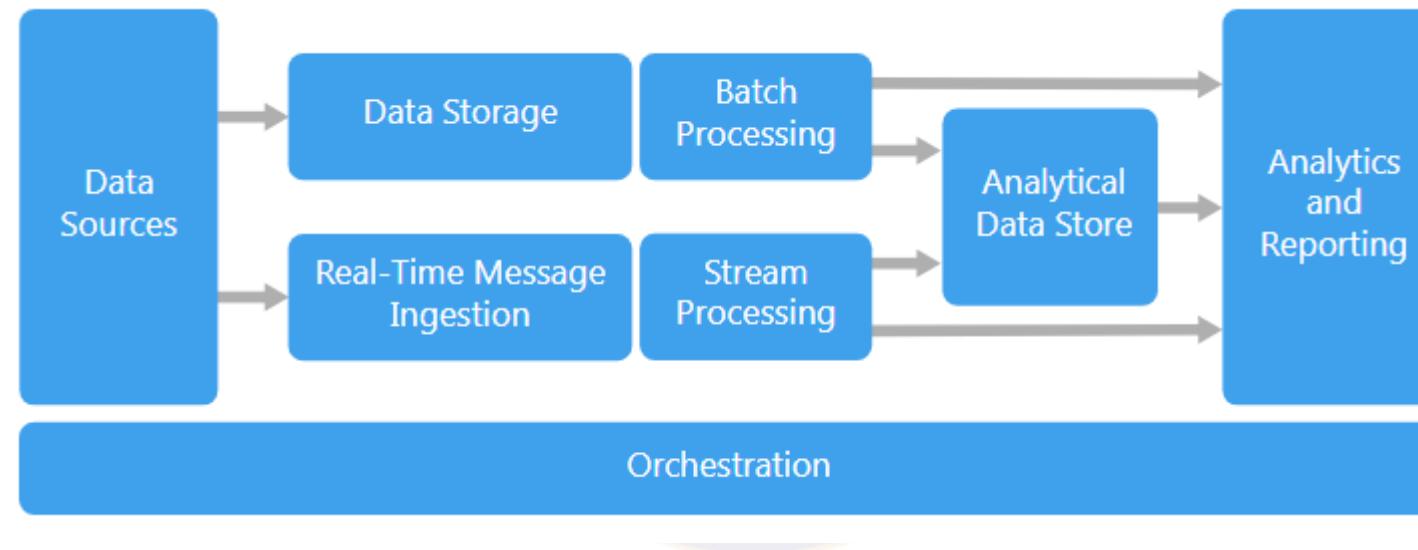
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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.

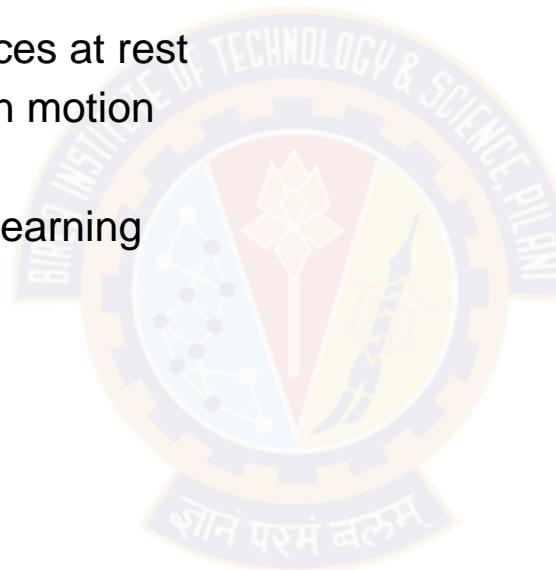


Source : <https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/big-data>

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

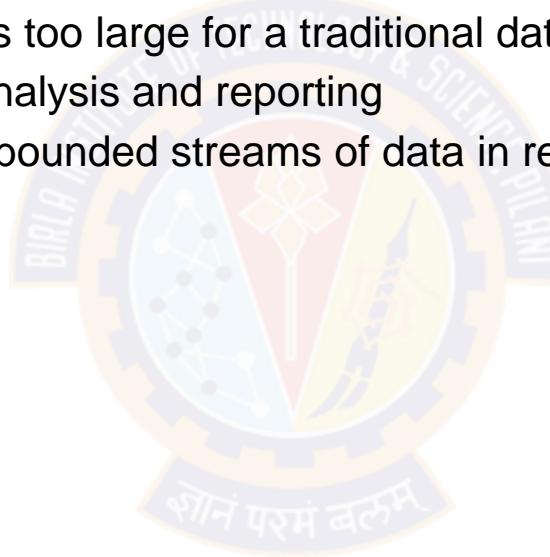
Components

- 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

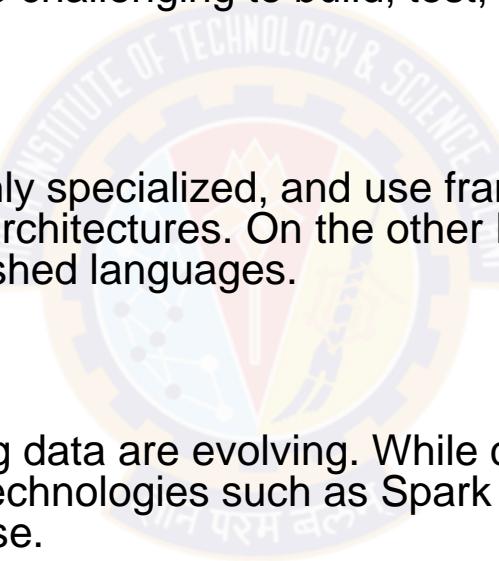
Advantages

- 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

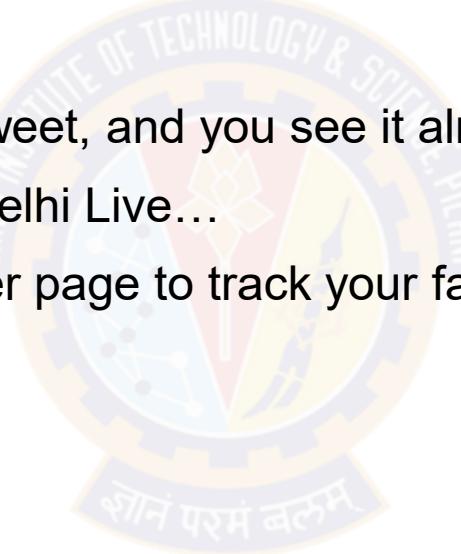
Real time systems

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...



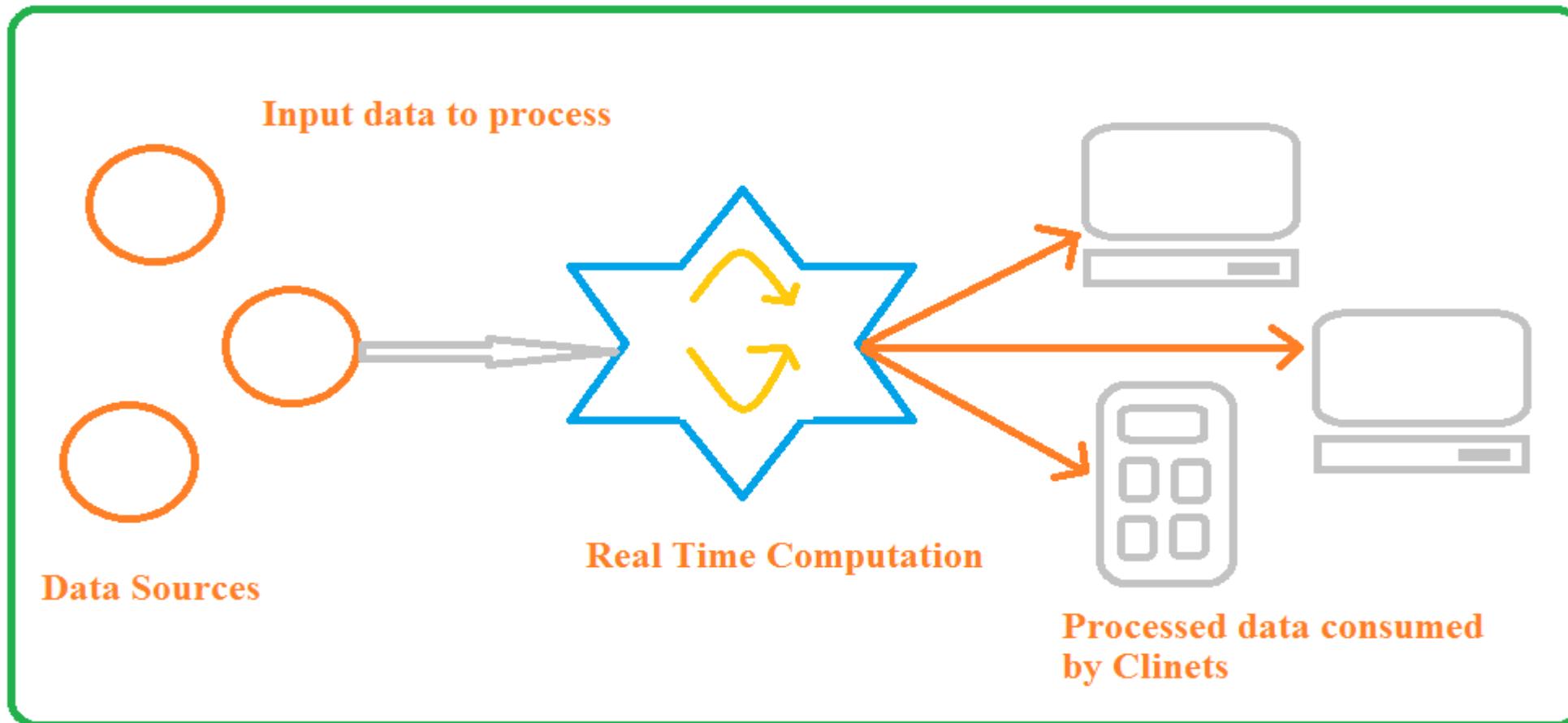
Classification of Real Time Systems

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

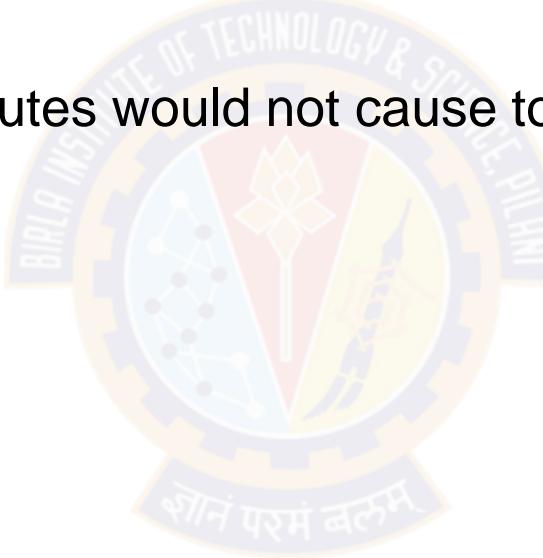


Source : Adapted from Streaming Data by Andrew Psaltis

A generic real-time system with consumers(2)

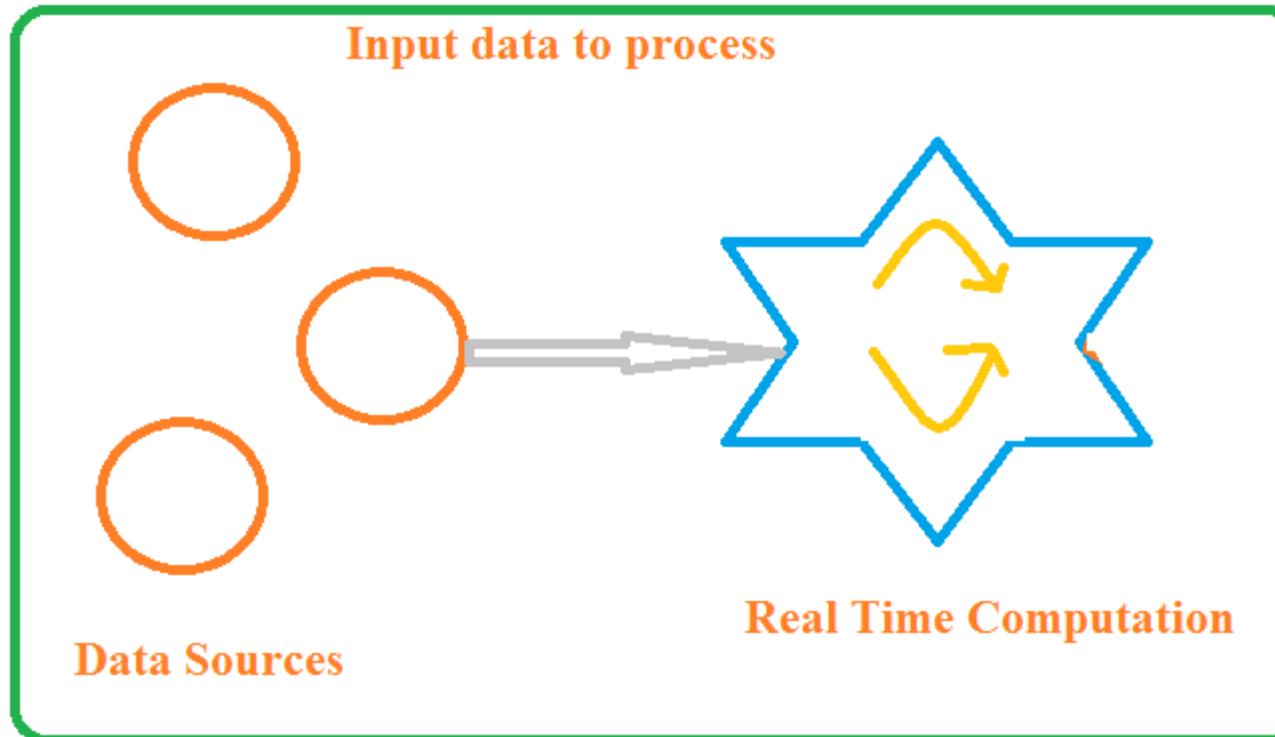
Examples discussion

- 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

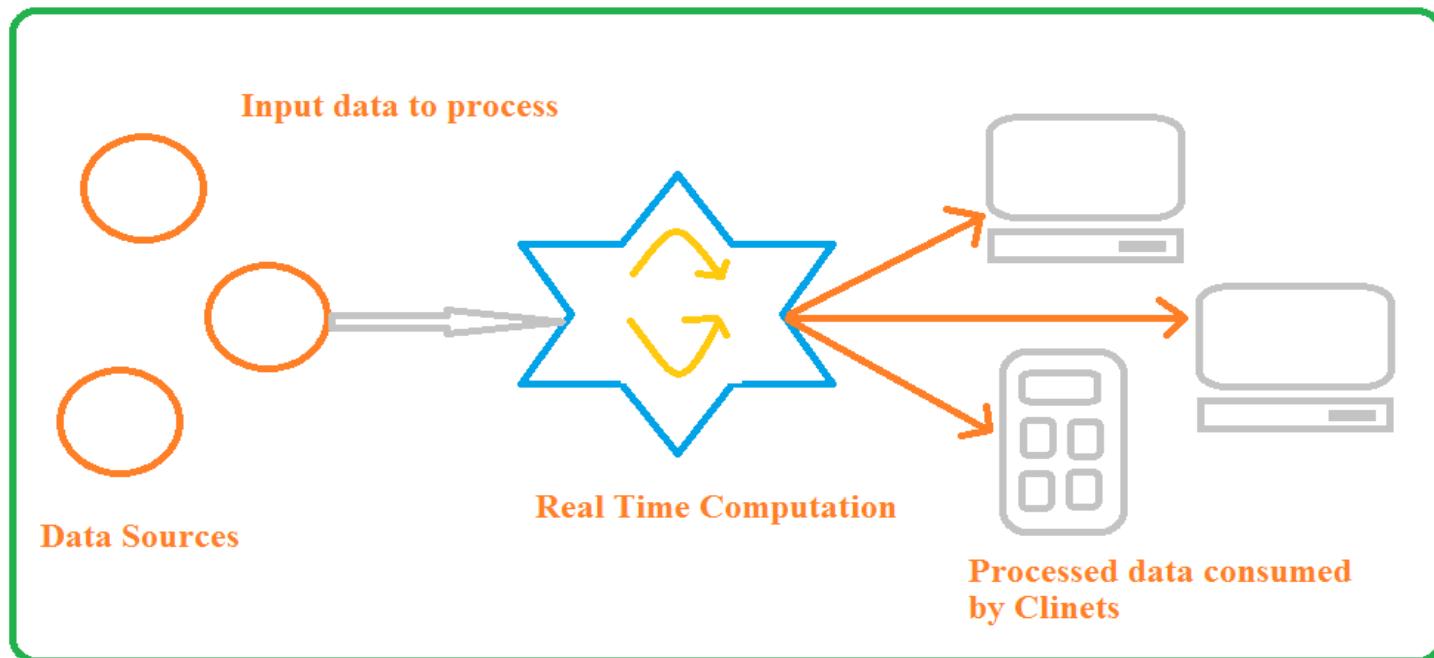
Difference between real-time and streaming systems

Pravin Y Pawar

Real-time systems again

Soft or near real time systems

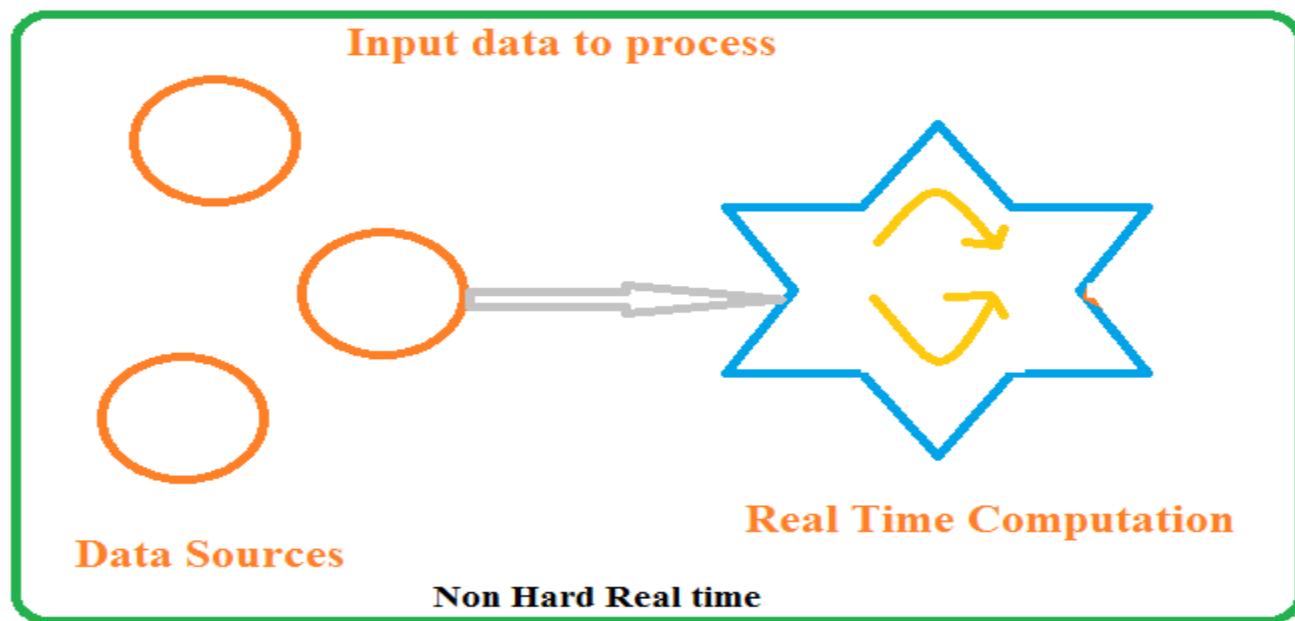
- Based on the delay experience by consumers – soft or near real time systems
- Difference between them is blurring , hard to distinguish
- Two parts are present in larger picture



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

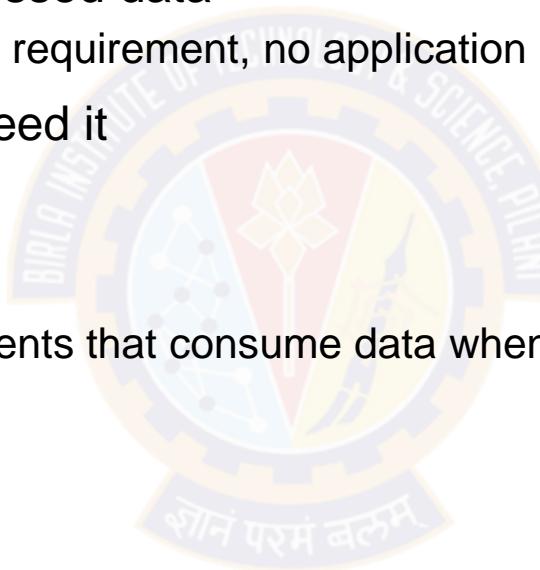


Source : Adapted from Streaming Data by Andrew Psaltis

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

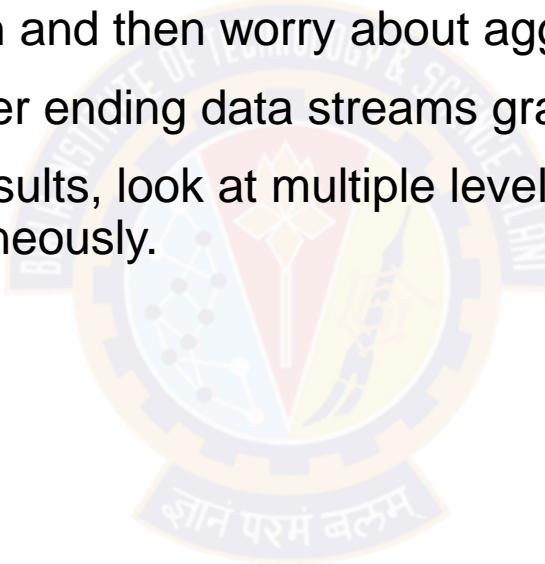
Streaming Data Applications

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.

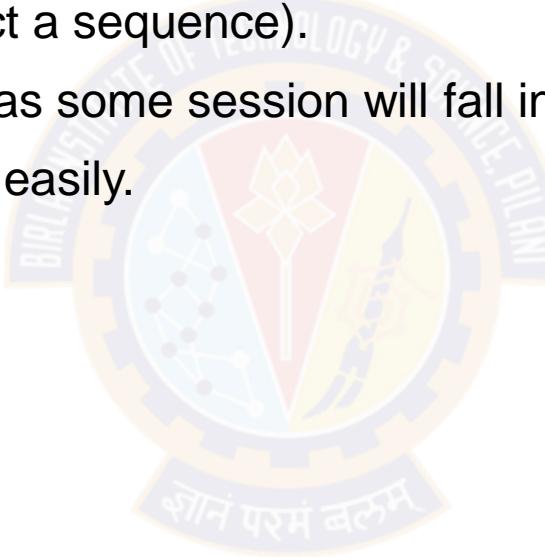


Source : <https://medium.com/stream-processing/what-is-stream-processing-1eadfca11b97>

Why is stream Processing needed? (2)

Reason 2

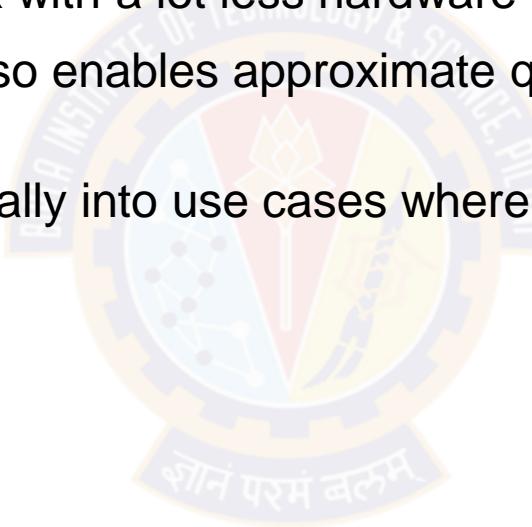
- Stream processing naturally fit with time series data and detecting patterns over time.
- For example, if you are trying to detect the length of a web session in a never-ending stream (this is an example of trying to detect a sequence).
- It is very hard to do it with batches as some session will fall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



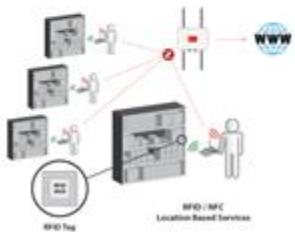
Smart Machinery



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

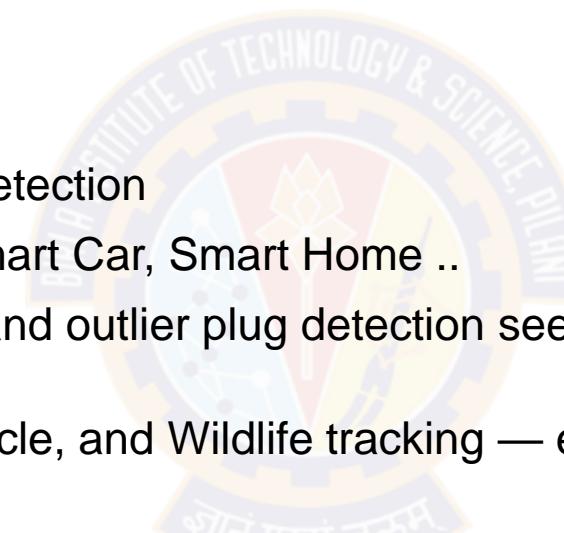
- Sensors in transportation vehicles, industrial equipment, and farm machinery send data to a streaming application. The application monitors performance, detects any potential defects in advance, and places a spare part order automatically preventing equipment down time.
- A financial institution tracks 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





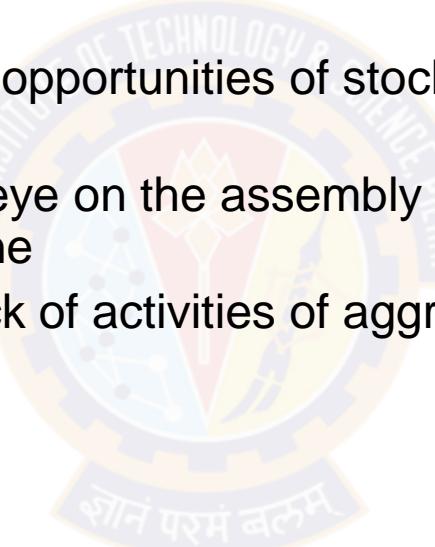
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

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.



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)

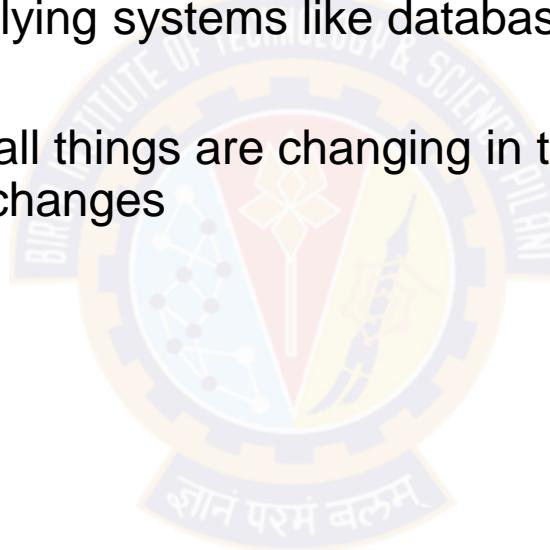
Stream Analytics

- 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

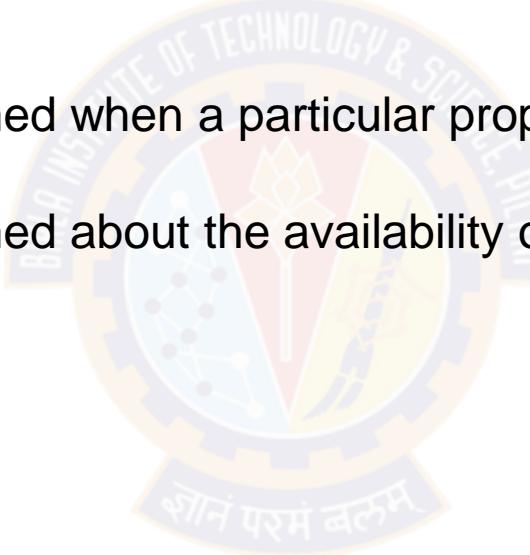
- Materialized views are kind of dependent applications like caches, search index, data warehouses etc.
- When something changes in underlying systems like databases, all dependents needs to be updated as well
- Can be achieved by tracking what all things are changing in the applications and derived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

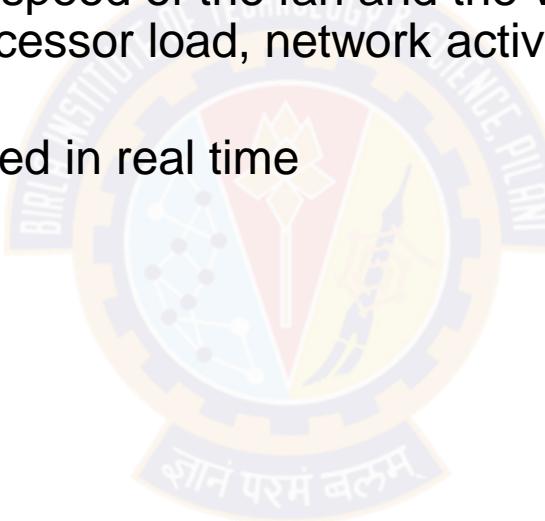
Sources of Streaming Data

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)

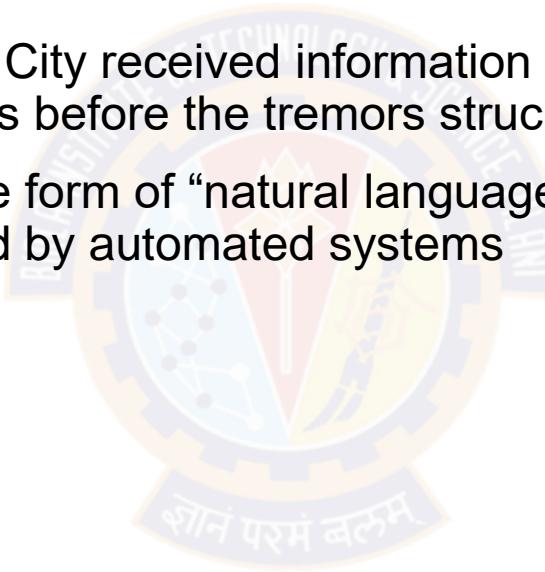
3. Online Advertising

- 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

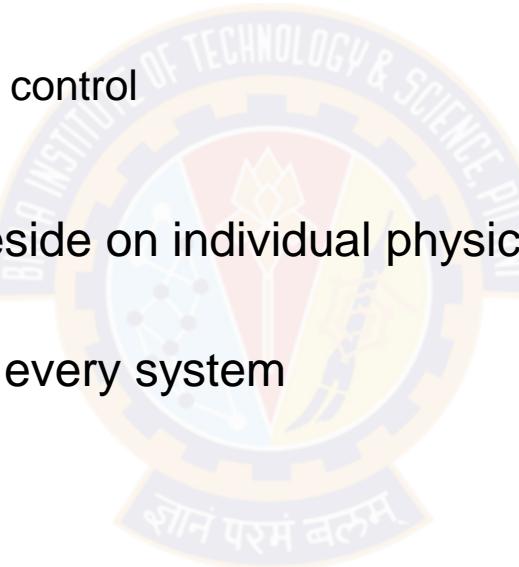
Generalized Streaming Data Architecture

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

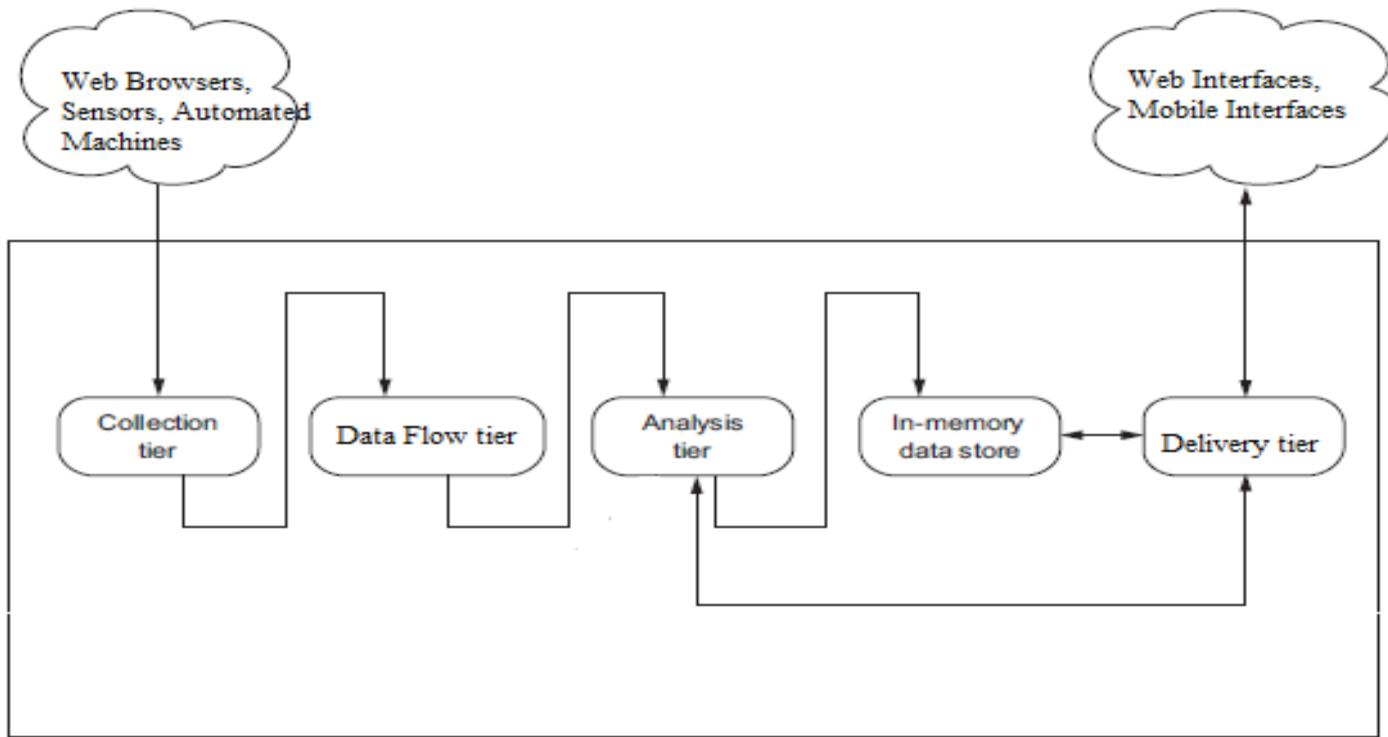


Streaming Data System Components

- 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

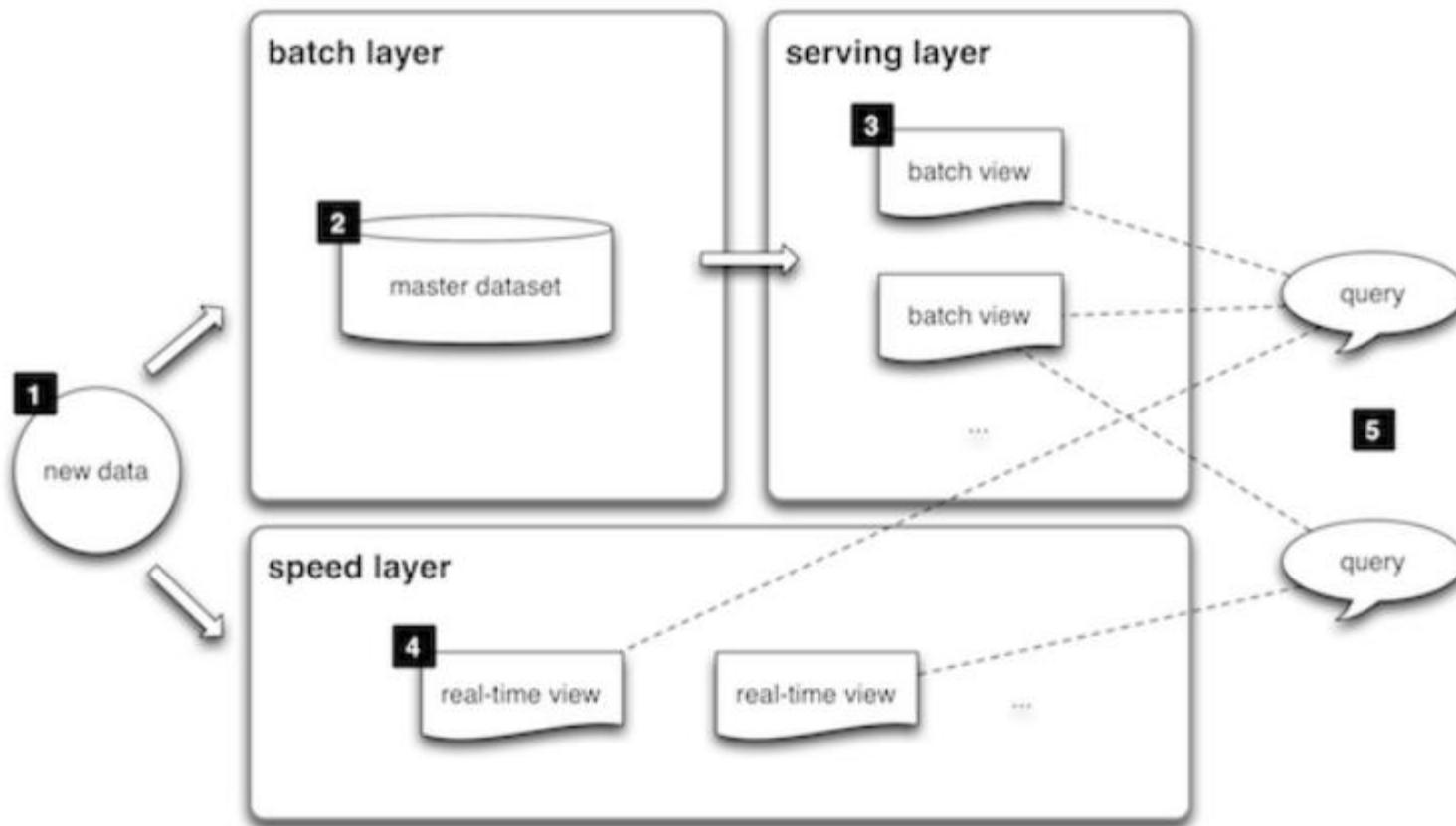
Lambda Architecture

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



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

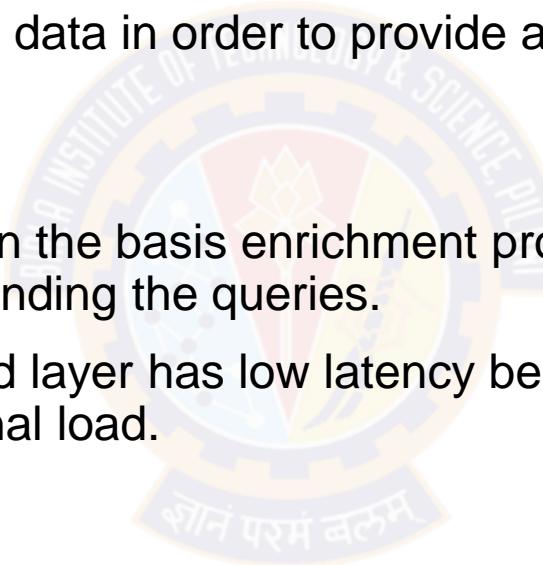
- New data comes continuously, as a feed to the data system.
- It gets fed to the batch layer and the speed layer simultaneously.
- It looks at all the data at once and eventually corrects the data in the stream layer.
- Here we can find lots of ETL and a traditional data warehouse.
- This layer is built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

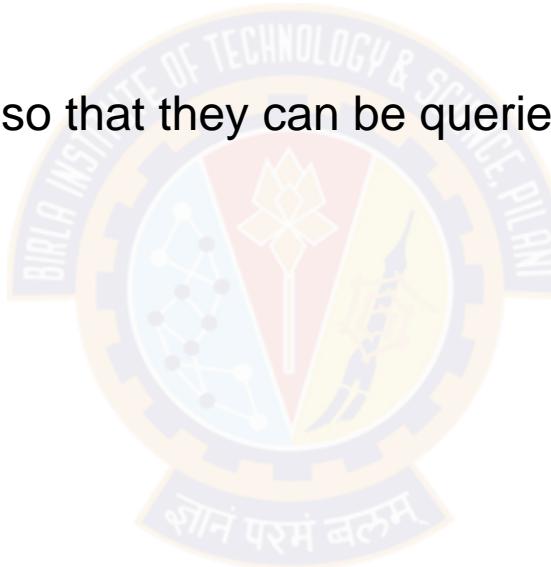
- This layer handles the data that are not already delivered in the batch view due to the latency of the batch layer.
- In addition, it only deals with recent data in order to provide a complete view of the data to the user by creating real-time views.
- Speed layer provides the outputs on the basis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

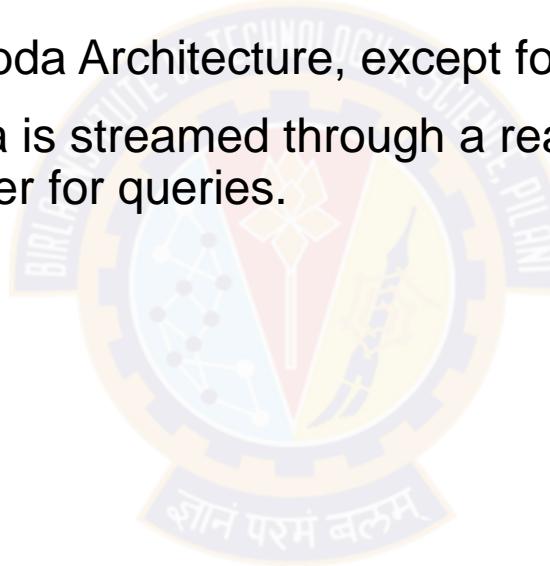
- The problem with the Lambda Architecture is that maintaining code that needs to produce the same result in two complex distributed systems is exactly as painful as it seems like it would be.
- Programming in distributed frameworks like Storm and Hadoop is complex. Inevitably, code ends up being specifically 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

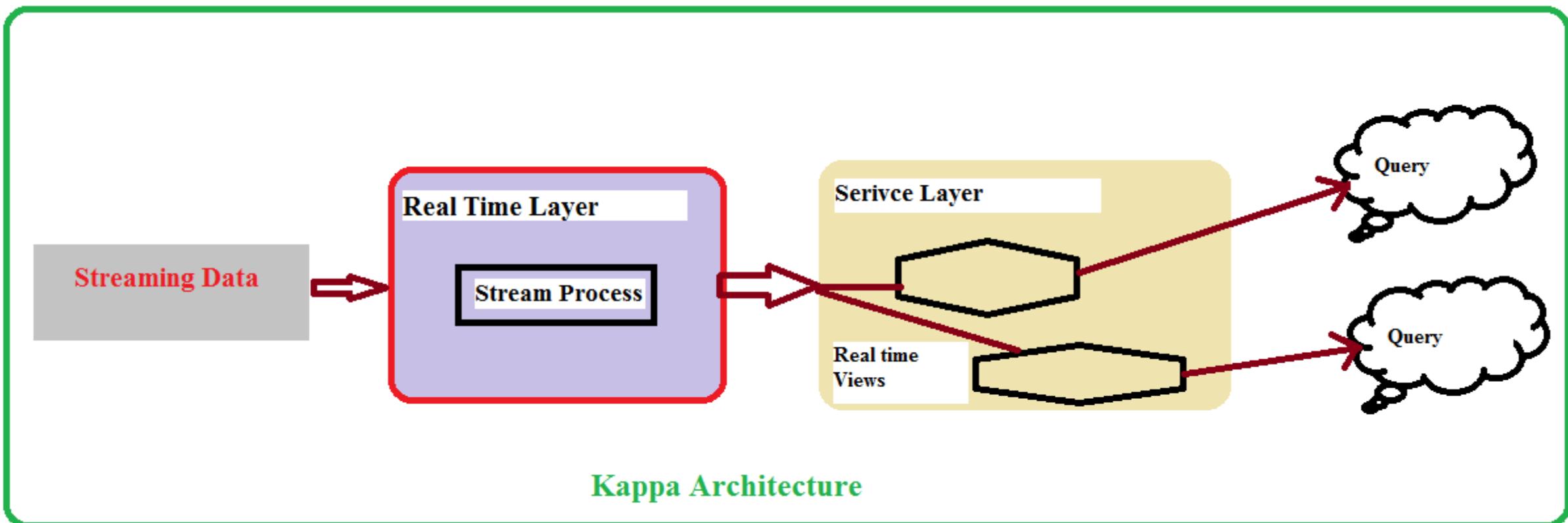
Defined

- 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.



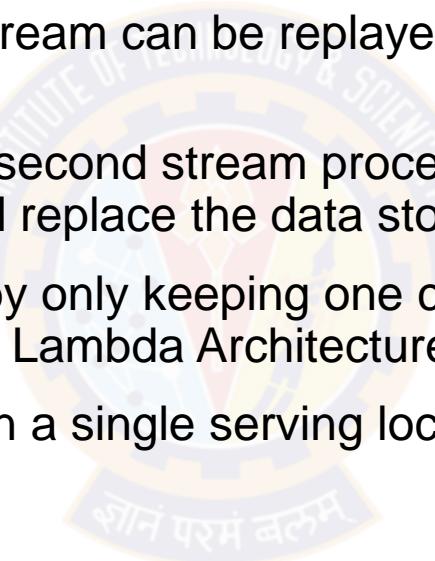
Kappa Architecture (2)

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->



BITS Pilani
Pilani Campus

Real Time System Characteristics

Pravin Y Pawar



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 sparse

- 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

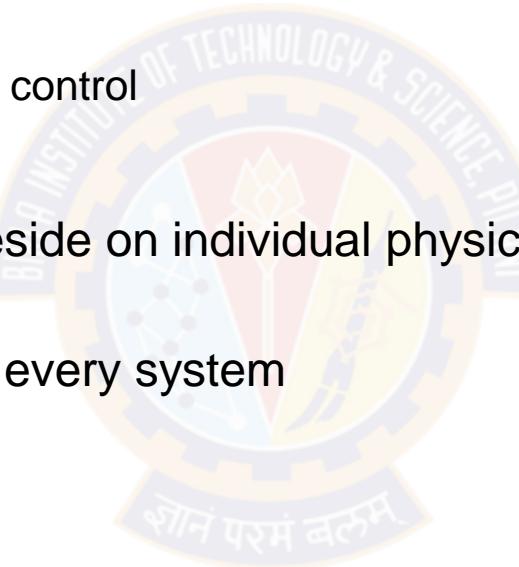
Generalized Streaming Data Architecture

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

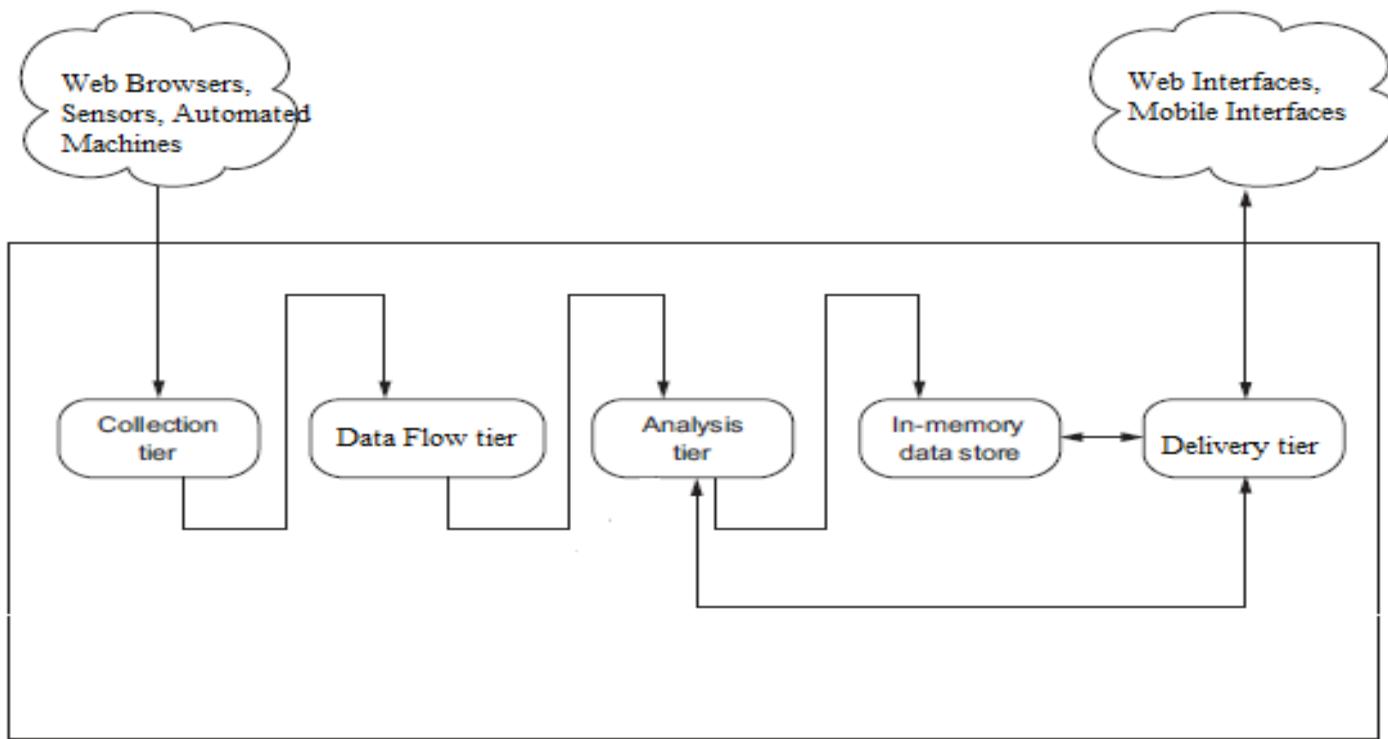


Streaming Data System Components

- 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

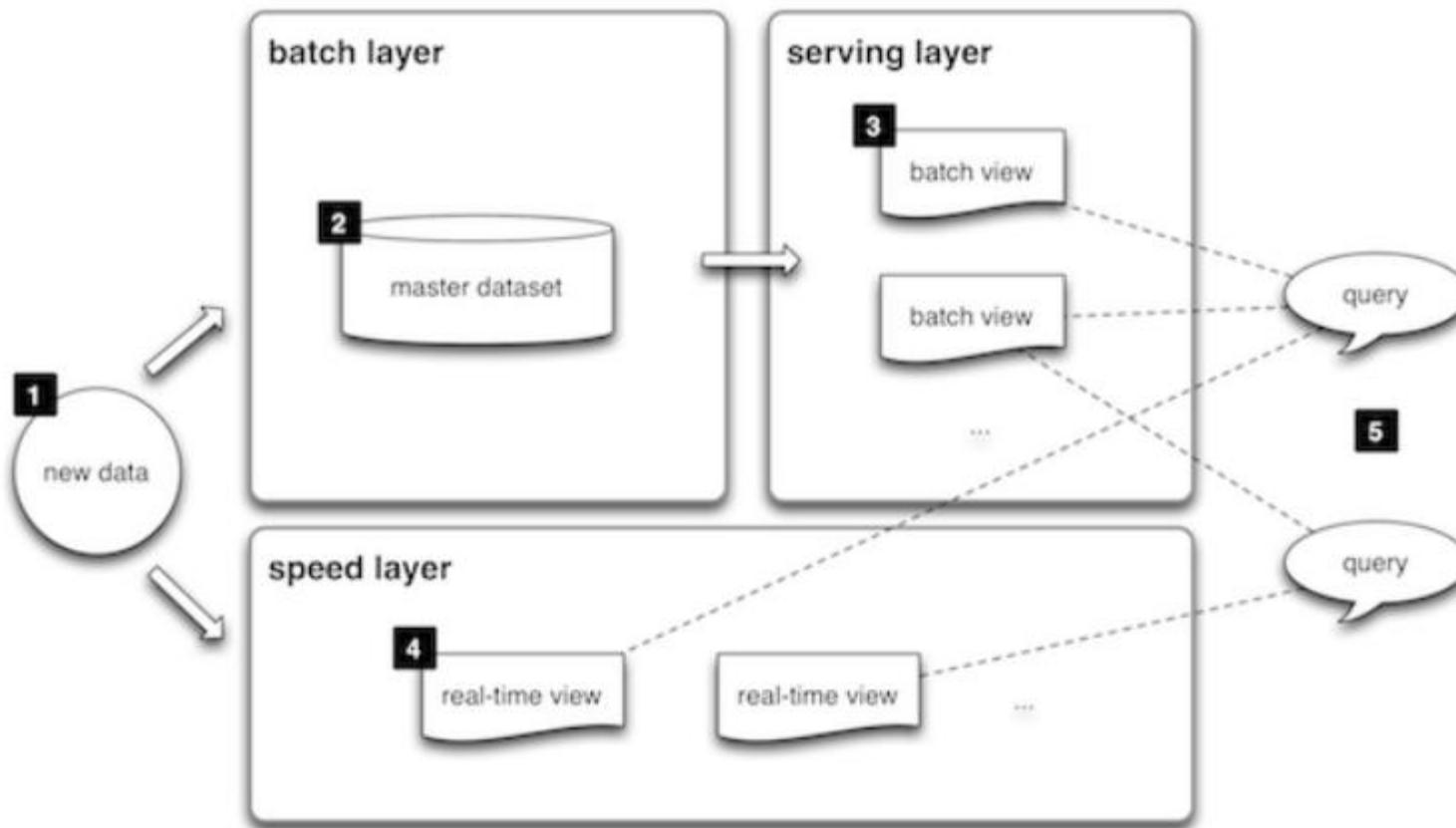
Lambda Architecture

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



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

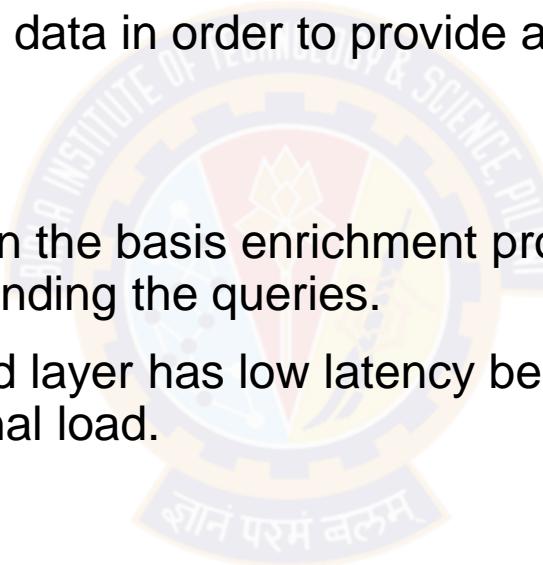
- New data comes continuously, as a feed to the data system.
- It gets fed to the batch layer and the speed layer simultaneously.
- It looks at all the data at once and eventually corrects the data in the stream layer.
- Here we can find lots of ETL and a traditional data warehouse.
- This layer is built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

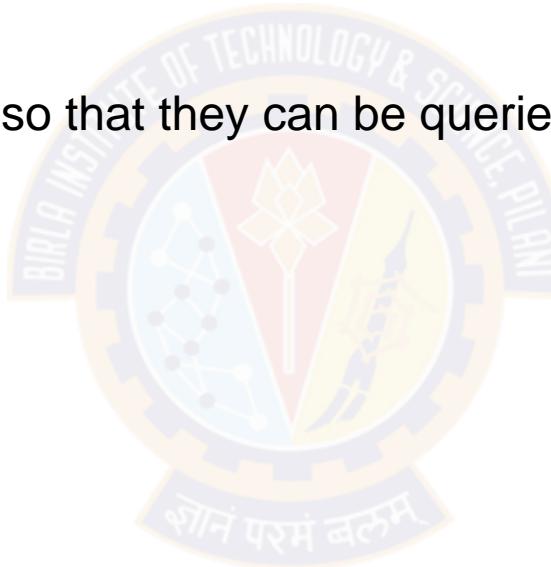
- This layer handles the data that are not already delivered in the batch view due to the latency of the batch layer.
- In addition, it only deals with recent data in order to provide a complete view of the data to the user by creating real-time views.
- Speed layer provides the outputs on the basis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

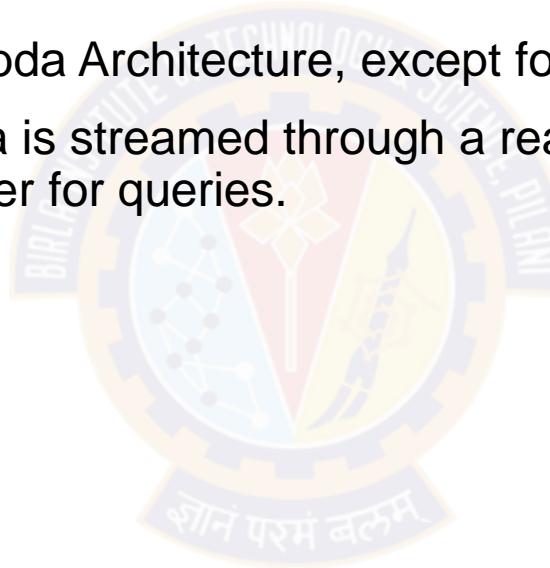
- The problem with the Lambda Architecture is that maintaining code that needs to produce the same result in two complex distributed systems is exactly as painful as it seems like it would be.
- Programming in distributed frameworks like Storm and Hadoop is complex. Inevitably, code ends up being specifically 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

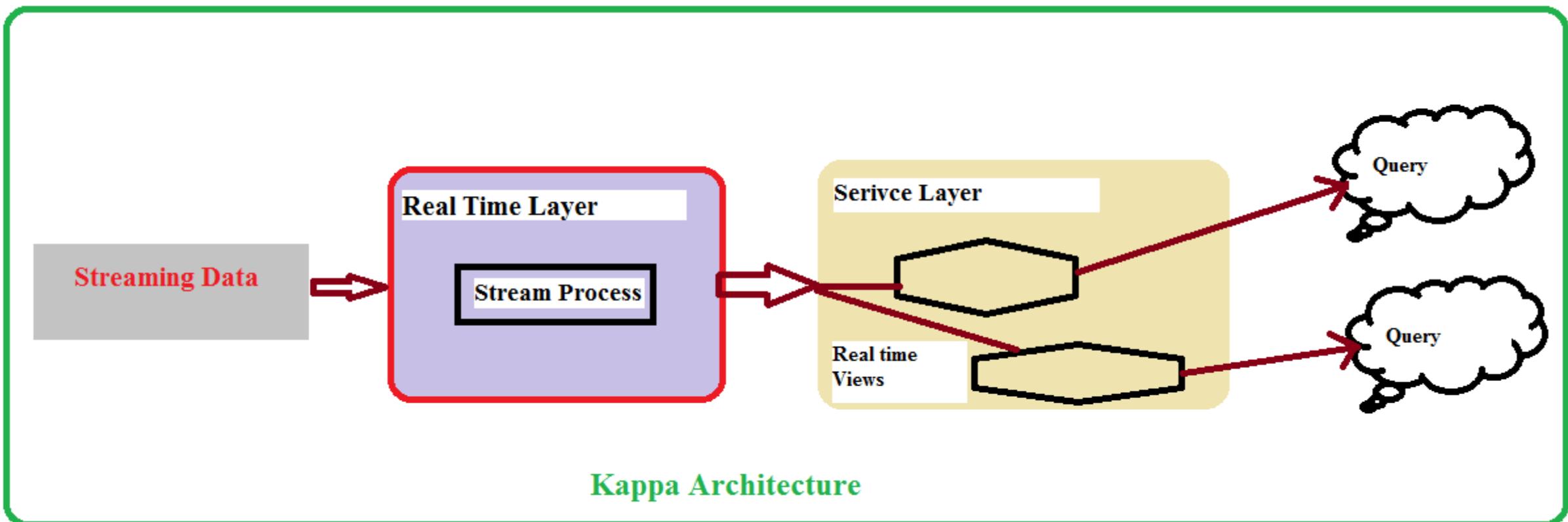
Defined

- 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.



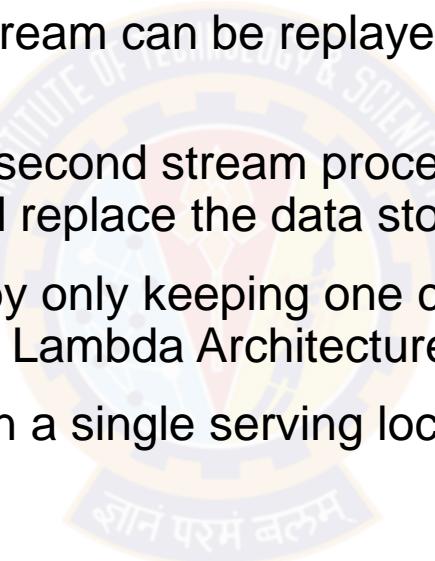
Kappa Architecture (2)

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->



BITS Pilani
Pilani Campus

Real Time System Characteristics

Pravin Y Pawar



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 sparse

- 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

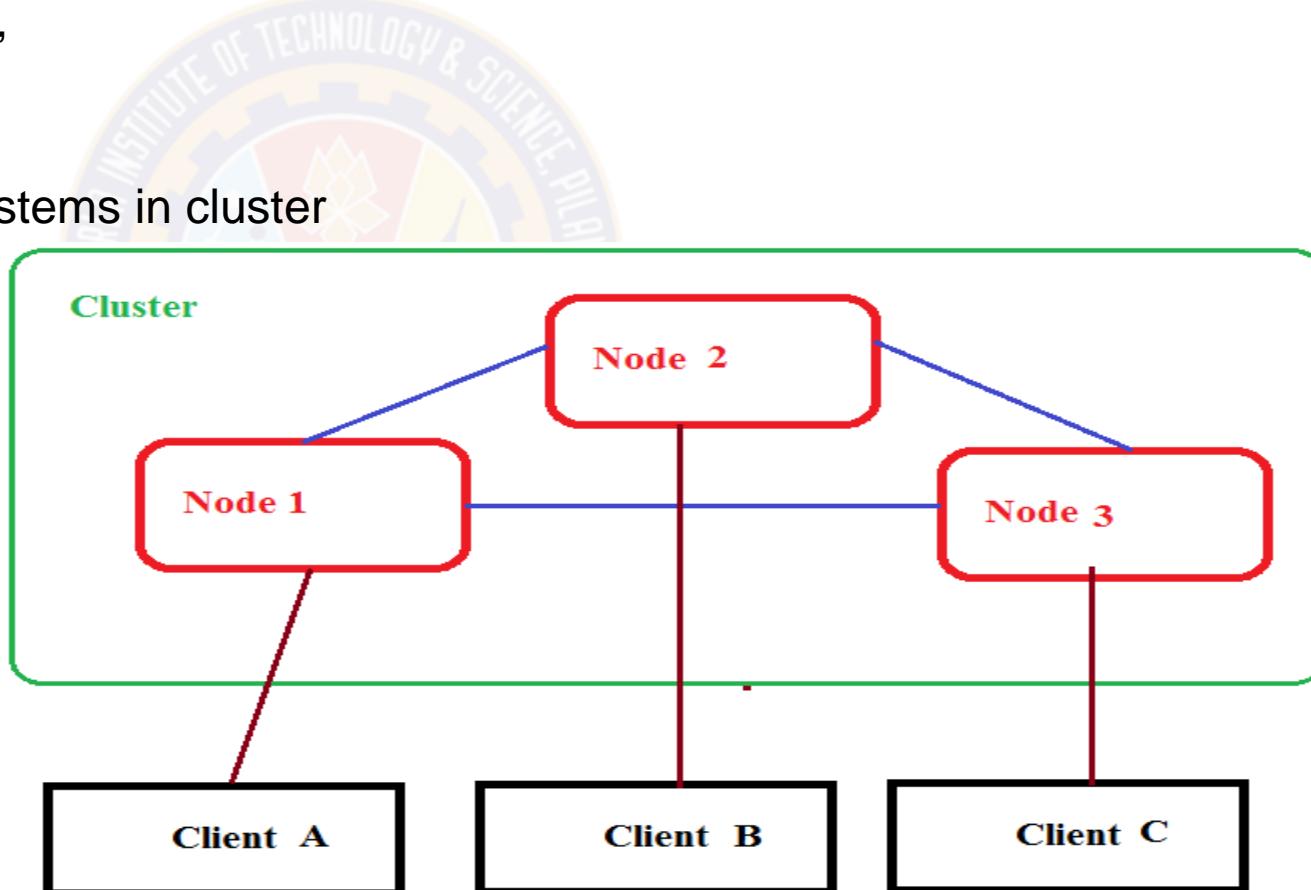
Service Configuration and Coordination Systems

Pravin Y Pawar

Distributed Applications

Overview

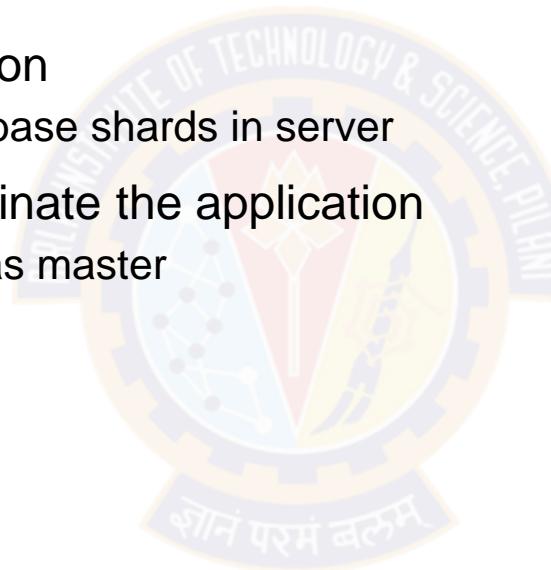
- Run on multiple systems in a network at a given time simultaneously
- Systems coordinates among themselves to carry out task in fast and efficient manner
- Collection of systems is “Cluster”
- System is termed as “Node”
- Client communicates with the systems in cluster



Motivation for Configuration and Coordination System

Need

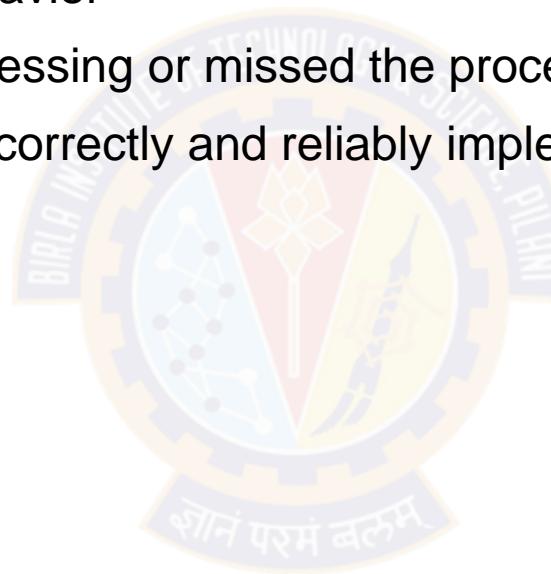
- Applications needs to share the metadata and state
- Metadata is configuration information
 - ✓ Example, location of various database shards in server
- System state is data used to coordinate the application
 - ✓ Example, server currently acting as master



Motivation for Configuration and Coordination System (2)

Need (Continued)

- Managing metadata and state is too difficult
- Often leads to incorrect server behavior
- Causes unacceptable 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

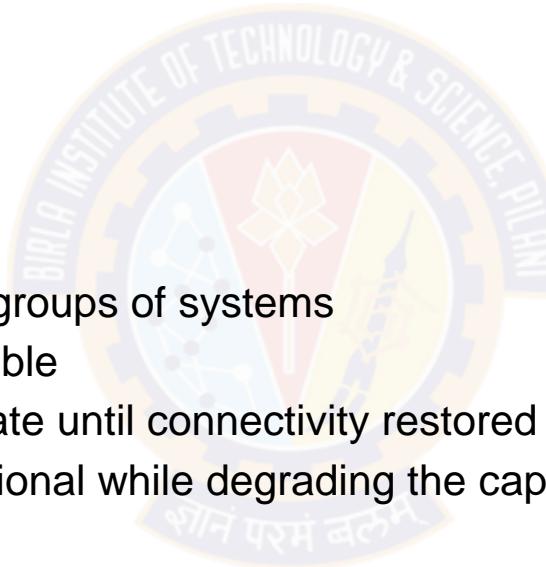
- Unreliable network connections
- Clock Synchronization
- Consistency in applications state



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

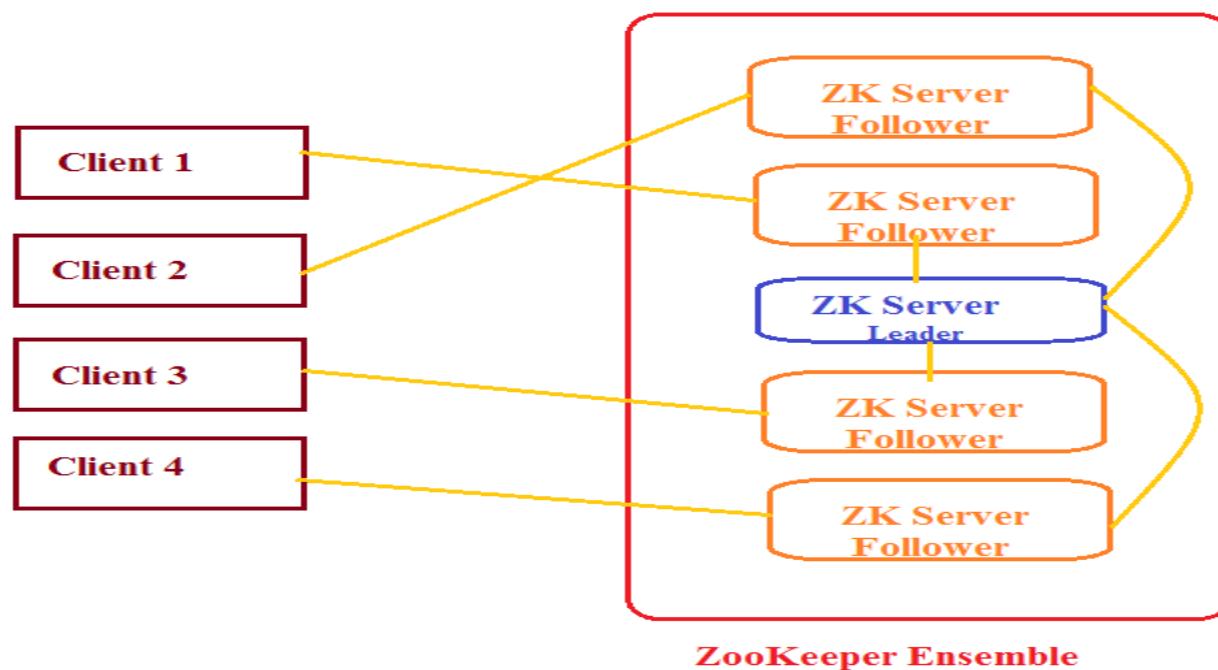
- State in distributed system has to be consistent in any case of failure
- Paxos algorithm or Multi-Paxos helps in maintaining the state
- But notoriously difficult to implement
- Recommended to use systems that already has implementations for the same



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





BITS Pilani
Pilani Campus

Apache ZooKeeper

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

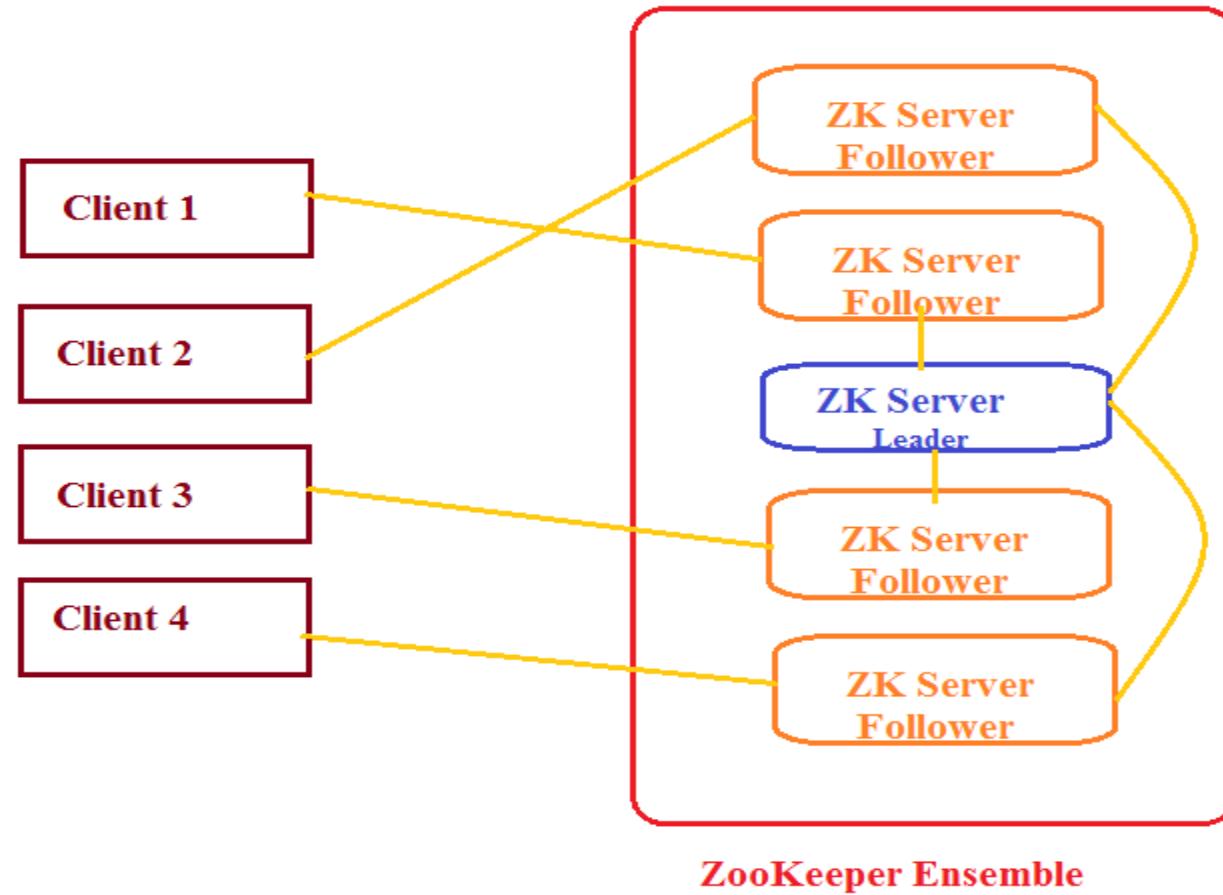
Apache ZooKeeper Fundamentals

➤ 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.

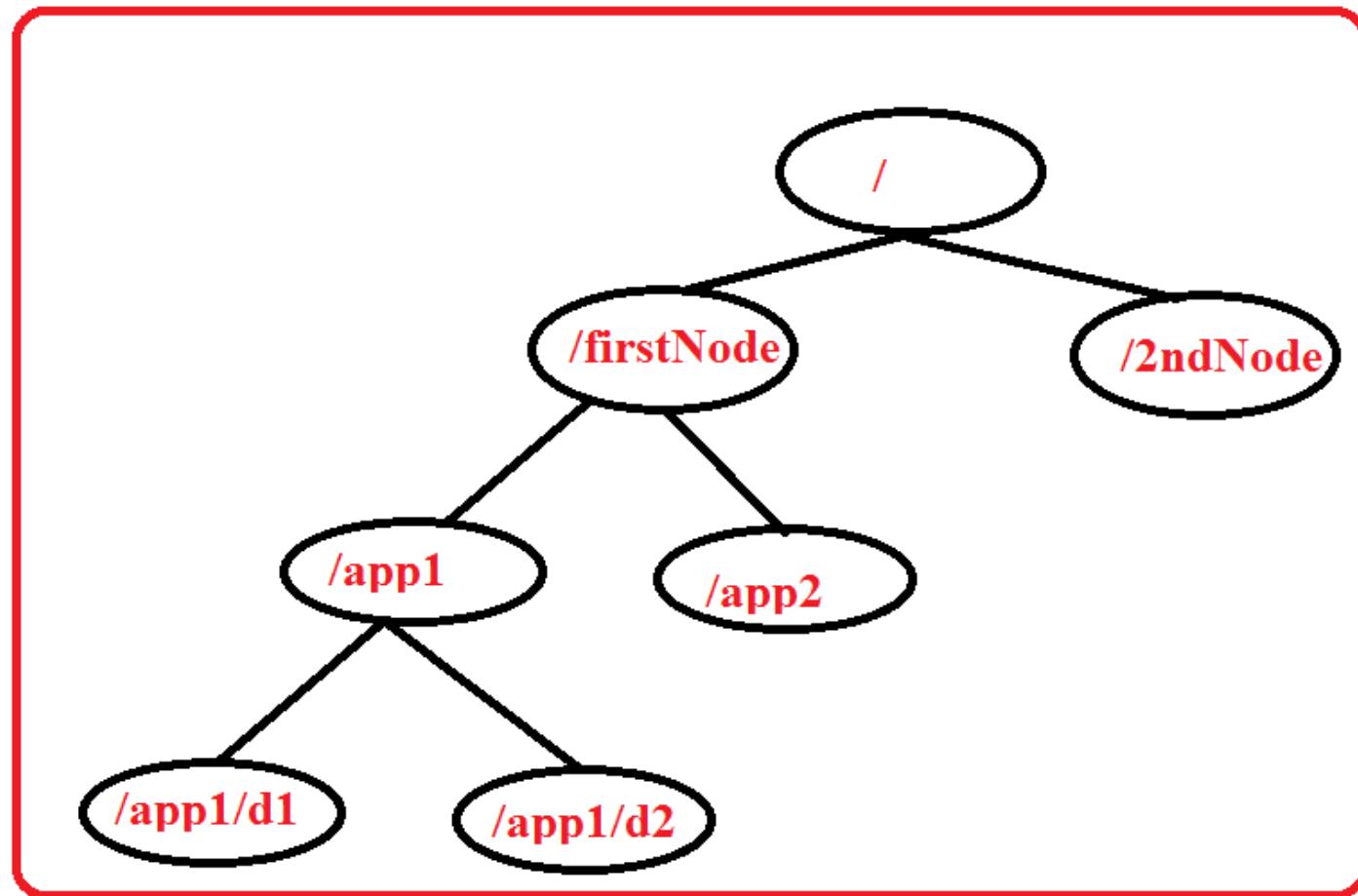
Apache ZooKeeper Fundamentals

➤ 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 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



BITS Pilani
Pilani Campus

Real Time System Characteristics

Pravin Y Pawar



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 sparse

- 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



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

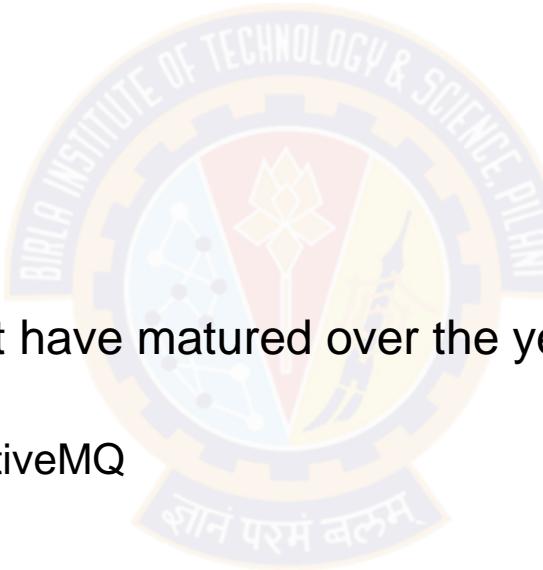
Data Flow Manager

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



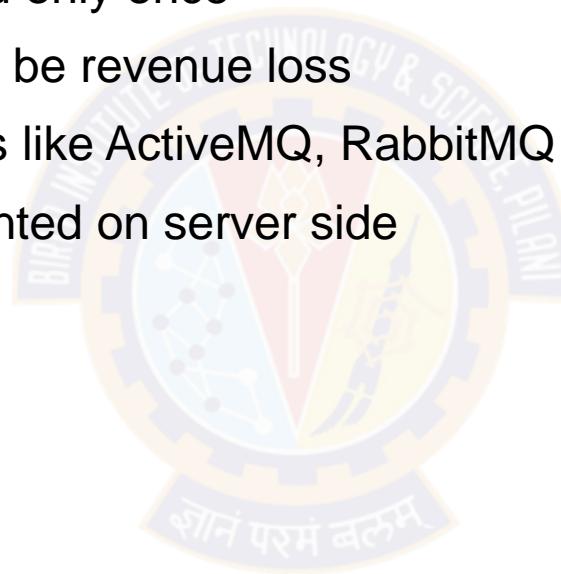
At most once delivery semantic

- 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



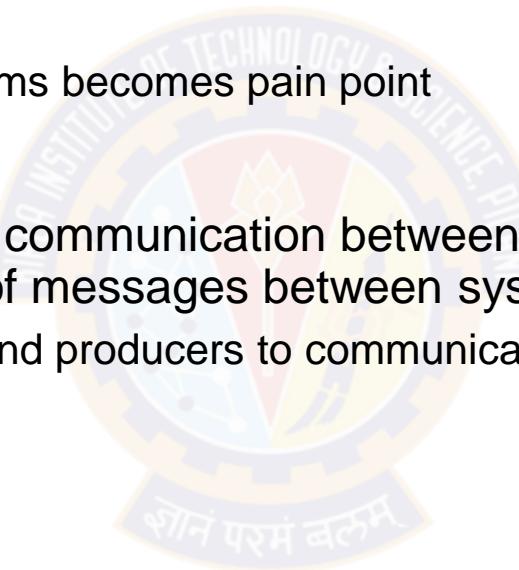
At least once delivery semantic

- 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



BITS Pilani
Pilani Campus

Apache Kafka

Pravin Y Pawar
CSIS-WILP



Agenda

- ❑ Messaging Systems
 - ❑ Kafka Architecture
 - ❑ Kafka Fundamentals
 - ❑ Kafka Workflow
 - ❑ Kafka Command Line Interface
 - ❑ Kafka Java APIs
-

Messaging System

- ❑ System to transfer data from one application to another
- ❑ Application focuses on its core features whereas messaging system takes care of data transfer / sharing

- ❑ Two types
 - ❑ Point to point
 - ❑ Pub-sub

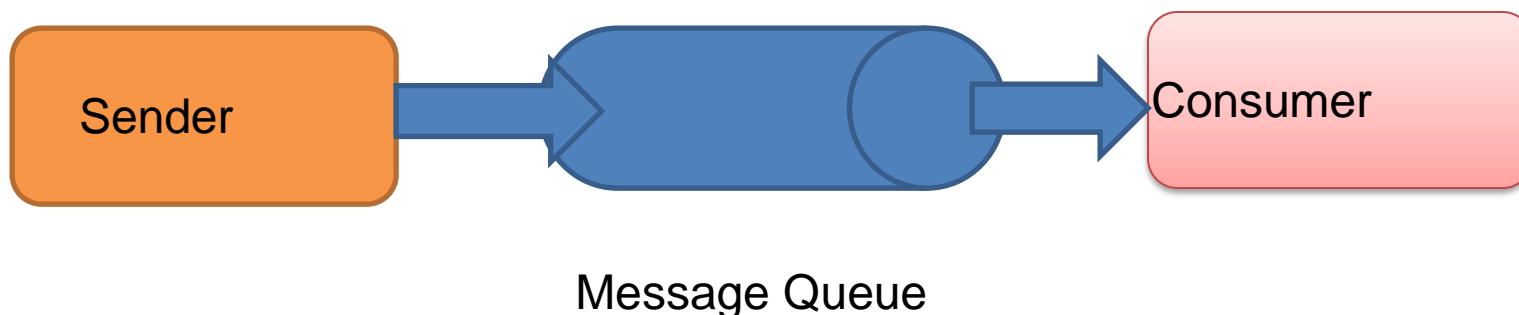
Point to Point messaging system

Messages are persisted in queue

One or more consumers can be connected to queue

But only one can consume the message

Message removed from queue, once any consumer reads it



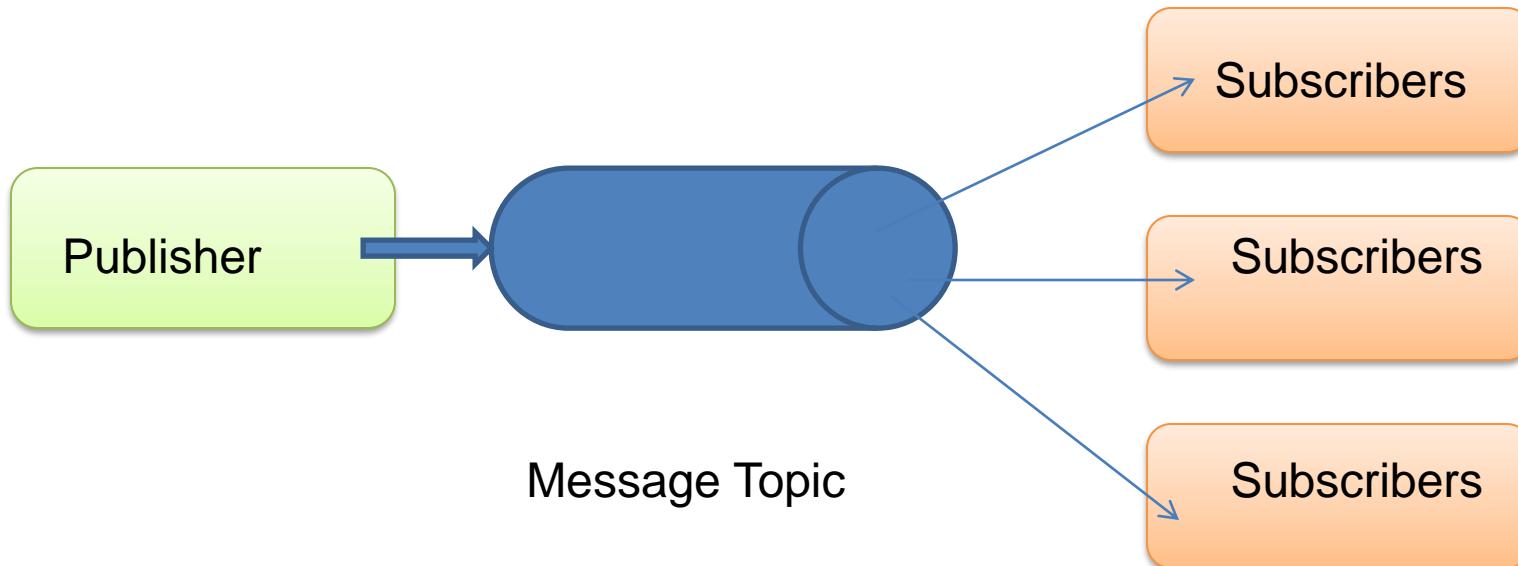
Pub-sub Messaging System

Messages are persisted in topic

Many consumers can consume messages at same time

Message producers are Publishers

Message consumers are Subscribers



Apache Kafka

- ❑ Distributed pub-sub and queuing messaging system
 - ❑ Designed for high throughput distributed systems
 - ❑ Better throughput
 - ❑ In-built partitioning mechanism
 - ❑ Replicas
 - ❑ Fault Tolerant
 - ❑ Good fit for large scale data processing
 - ❑ Suitable for both offline and online message transfer
 - ❑ Messages are persisted on disk and replicated across cluster to prevent message loss
 - ❑ Integrate with Storm and Spark easily
-

Kafka Benefits

Reliability

- distributed, partitioned, replicated and fault tolerant

Scalability

- scales easily without down time by addition of machines in cluster

Durability

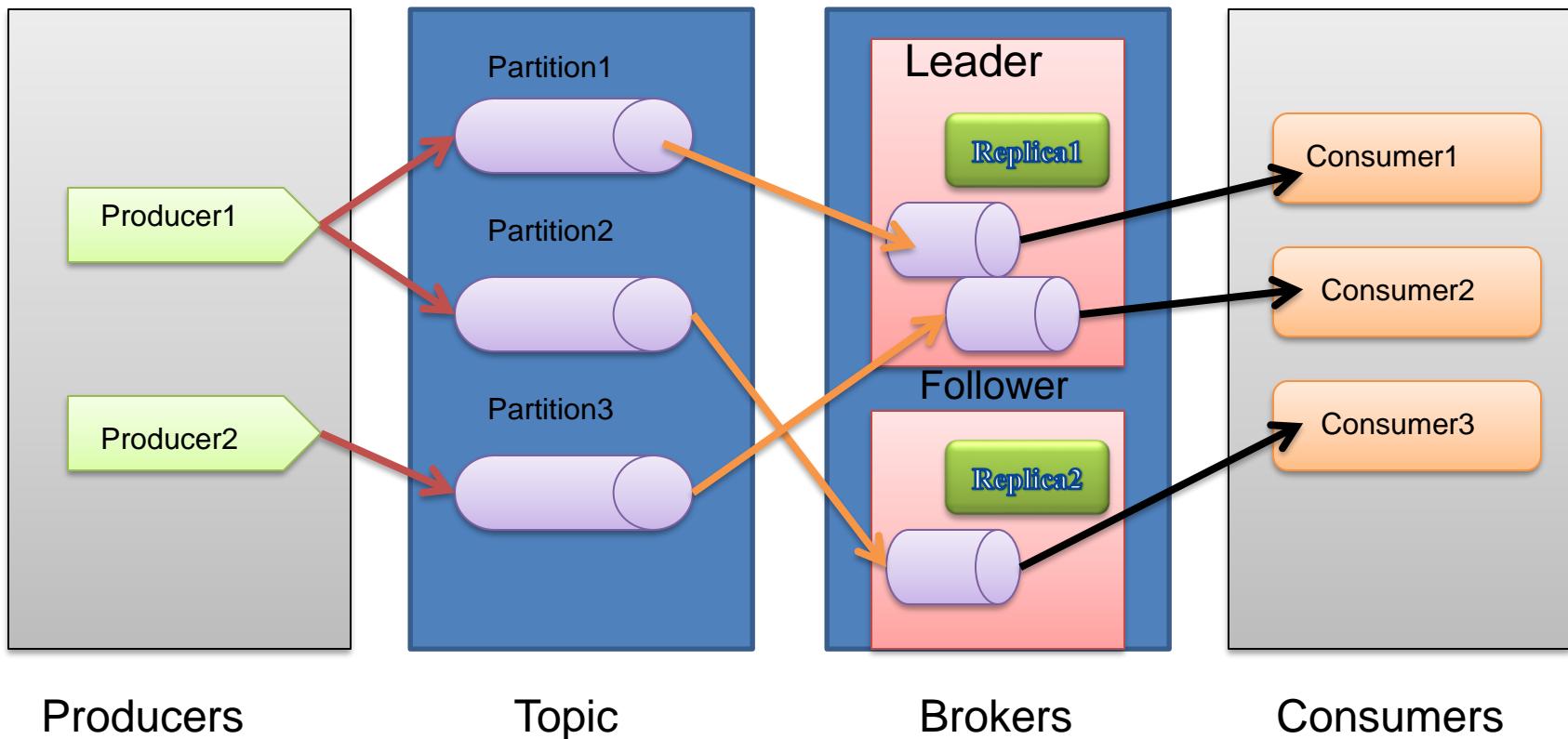
- messages persists on disk as fast as possible

Performance

- has high throughput for both publishing and subscribing messages

Kafka Fundamentals

Kafka Architecture



Producers

Topic

Brokers

Consumers

Topics

- ❑ A stream of messages belonging to a particular category
 - ❑ Data is stored in topic
 - ❑ Topics are split into partitions
 - ❑ For each topic, Kafka keeps a minimum of one partition
 - ❑ Each such partition contains messages in an immutable ordered sequence
 - ❑ A partition is implemented as a set of segment files of equal sizes
-

Partition

- ❑ Topics may have many partitions, so it can handle an arbitrary amount of data.
 - ❑ Each partitioned message has a unique sequence id called as "offset".
 - ❑ Replicas are nothing but "backups" of a partition.
 - ❑ Replicas are never read or write data. They are used to prevent data loss.
-

Brokers

- ❑ Simple system responsible for maintaining the published data
 - ❑ Each broker may have zero or more partitions per topic
 - ❑ Used to maintain load balance
 - ❑ are stateless, so they use ZooKeeper for maintaining their cluster state
- ❑ If there are N partitions in a topic and N number of brokers
 - ✓ each broker will have one partition
- ❑ If there are N partitions in a topic and more than N brokers
 - ✓ the first N broker will have one partition and the next M broker will not have any partition for that particular topic
- ❑ If there are N partitions in a topic and less than N brokers
 - ✓ each broker will have one or more partition sharing among them

Producers

- ❑ The publisher of messages to one or more Kafka topics
 - ❑ Sends data to Kafka brokers
 - ❑ Every time a producer publishes a message to a broker, the broker simply appends the message to the last segment file
 - ❑ Actually, the message will be appended to a partition
 - ❑ Producer can also send messages to a partition of their choice
 - ❑ Kafka producer doesn't wait for acknowledgements from the broker and sends messages as fast as the broker can handle.
-

Consumers

- ❑ Consumers read data from brokers.
- ❑ Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers.
- ❑ Since Kafka brokers are stateless, which means that the consumer has to maintain how many messages have been consumed by using partition offset.
 - ❑ If the consumer acknowledges a particular message offset, it implies that the consumer has consumed all prior messages.
 - ❑ The consumers can rewind or skip to any point in a partition simply by supplying an offset value.
 - ❑ Consumer offset value is notified by ZooKeeper.

Kafka Cluster

- ❑ Kafka's having more than one broker are called as Kafka cluster.
 - ❑ A Kafka cluster can be expanded without downtime.
 - ❑ These clusters are used to manage the persistence and replication of message data.
-

Leader and Follower node

□ Leader

- ✓ "Leader" is the node responsible for all reads and writes for the given partition.
- ✓ Every partition has one server acting as a leader.

□ Follower

- ✓ Node which follows leader instructions are called as follower.
- ✓ If the leader fails, one of the follower will automatically become the new leader.
- ✓ A follower acts as normal consumer, pulls messages and updates its own data store.

ZooKeeper

- ❑ ZooKeeper is used for managing and coordinating Kafka broker.
- ❑ ZooKeeper service is mainly used to notify producer and consumer about the presence of any new broker in the Kafka system or failure of the broker in the Kafka system.
 - ✓ As per the notification received by the Zookeeper regarding presence or failure of the broker then producer and consumer takes decision and starts coordinating their task with some other broker.

Kafka Workflow

- Kafka is simply a collection of topics split into one or more partitions.
 - A Kafka partition is a linearly ordered sequence of messages, where each message is identified by their index .
 - All the data in a Kafka cluster is the disjointed union of partitions.
 - Incoming messages are written at the end of a partition and messages are sequentially read by consumers.

 - Kafka provides both pub-sub and queue based messaging system in a fast, reliable, persisted, fault-tolerance and zero downtime manner.
-

Workflow of Pub-Sub Messaging

- ❑ Producers send message to a topic at regular intervals.
- ❑ Kafka broker stores all messages in the partitions configured for that particular topic. It ensures the messages are equally shared between partitions.
- ❑ Consumer subscribes to a specific topic.
- ❑ Once the consumer subscribes to a topic, Kafka will provide the current offset of the topic to the consumer and also saves the offset in the Zookeeper ensemble.
- ❑ Consumer will request the Kafka in a regular interval for new messages.
- ❑ Once Kafka receives the messages from producers, it forwards these messages to the consumers.
- ❑ Consumer will receive the message and process it.
- ❑ Once the messages are processed, consumer will send an acknowledgement to the Kafka broker.
- ❑ Once Kafka receives an acknowledgement, it changes the offset to the new value and updates it in the Zookeeper. Since offsets are maintained in the Zookeeper, the consumer can read next message correctly even during server outages.
- ❑ This above flow will repeat until the consumer stops the request.
- ❑ Consumer has the option to rewind/skip to the desired offset of a topic at any time and read all the subsequent messages.

Workflow of Queue Messaging / Consumer Group

- In a queue messaging system
 - Instead of a single consumer, a group of consumers having the same "Group ID" will subscribe to a topic.
 - Consumers subscribing to a topic with same "Group ID" are considered as a single group and the messages are shared among them.

Workflow of Queue Messaging / Consumer Group(2)



- ❑ Producers send message to a topic in a regular interval.
- ❑ Kafka stores all messages in the partitions configured for that particular topic.
- ❑ A single consumer subscribes to a specific topic.
- ❑ Kafka interacts with the consumer in the same way as Pub-Sub Messaging until new consumer subscribes the same topic.
- ❑ Once the new consumer arrives, Kafka switches its operation to share mode and shares the data between the two consumers.
- ❑ This sharing will go on until the number of consumers reach the number of partition configured for that particular topic.
- ❑ Once the number of consumer exceeds the number of partitions, the new consumer will not receive any further message until any one of the existing consumer unsubscribe.

Kafka Server Properties file

Config / server.properties

```
# The id of the broker. This must be set to a unique integer  
# for each broker.  
broker.id=1  
  
# The port the socket server listens on  
port=9092  
  
# A comma separated list of directories under which to  
# store log files  
log.dirs=/tmp/kafka-logs
```

Kafka CLI

-
- ❑ Server Management
 - ❑ Zookeeper Server
 - ✓ bin/zookeeper-server-start.sh config/zookeeper.properties
 - ❑ Kafka Server
 - ✓ bin/kafka-server-start.sh config/server.properties

Kafka CLI(2)

❑ Kafka Topic Lifecycle Management

- ❑ bin/kafka-topics.sh

❑ Create

- ✓ --create --zookeeper zkConnectionString --replication-factor count --partitions count
--topic topic-name

❑ List

- ✓ --list --zookeeper zkConnectionString

❑ Update

- ✓ --alter --zookeeper zkConnectionString --topic topic_name --partitions count

❑ Delete

- ✓ --delete --zookeeper zkConnectionString --topic topic_name

Kafka CLI(3)

❑ Kafka Producer

- ❑ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic topic-name

❑ Kafka Consumer

- ❑ bin/kafka-console-consumer.sh --zookeeper localhost:2181 —topic topic-name --from-beginning

Kafka Java API

❑ Kafka Producer API

- ✓ Applications intending to write into Kafka topics

❑ Kafka Consumer API

- ✓ Applications intending to read messages from Kafka topics

❑ Kafka Streaming API

- ✓ Applications intending to do computations on messages

Kafka Java API (2)

❑ Kafka Producer API

❑ Producer class

- ❑ The central part of the Producer API
- ❑ provides an option to connect Kafka broker in its constructor
- ❑ provides close method to close the producer pool connections to all Kafka brokers.

❑ KafkaProducer class

- ❑ Send - to send messages asynchronously to a topic
- ❑ Flush - to ensure all previously sent messages have been actually completed
- ❑ Metrics - helps in getting the partition metadata for a given topic

❑ ProducerRecord

- ❑ key/value pair that is sent to Kafka cluster
-

Kafka Java API (3)

- ❑ Kafka Consumer API
- ❑ KafkaConsumer class
 - used to consume messages from the Kafka Cluster
 - Subscribe - Subscribe to the given list of topics
 - Unsubscribe - Unsubscribe the topics from the given list of partitions
 - Poll - Fetch data for the topics or partitions specified using one of the subscribe/assign APIs
- ❑ ConsumerRecords class
 - acts as a container for ConsumerRecord
 - is used to keep the list of ConsumerRecord per partition for a particular topic
- ❑ ConsumerRecord class
 - used to create a consumer record with specific topic name, partition count and <key, value> pairs.

Reference



➤ Real-Time Analytics , Byron Ellis

- ❖ <https://kafka.apache.org/quickstart>
- ❖ <https://kafka.apache.org/intro.html>
- ❖ <http://tutorialspoint.com>



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Key features of Streaming Data Frameworks

Pravin Y Pawar

Key features of Streaming Data Frameworks

- Look for following when determining the framework
- Message Delivery Semantics
- State Management
- Fault Tolerance



Message Delivery Semantics

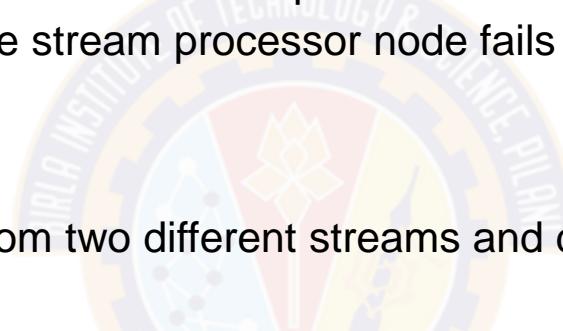
Types

- At most once
 - ✓ Message may get lost, never processed second time
 - ✓ Simplest delivery mechanism
 - ✓ Message may be lost or streaming processor fails to process message
- At least once
 - ✓ Never losses message, may be processed many times
 - ✓ Increased complexity as needs to keep track which is the message last processed
 - ✓ If message is not processed, sent back for processing
 - ✓ Every time same message is received, same output should be produced
- Exactly once
 - ✓ Message never lost and processed only once
 - ✓ Needs to keep track of messages processed and remove duplicates
 - ✓ Responds with a success or failure after message is produced

State Management

Where to maintain state?

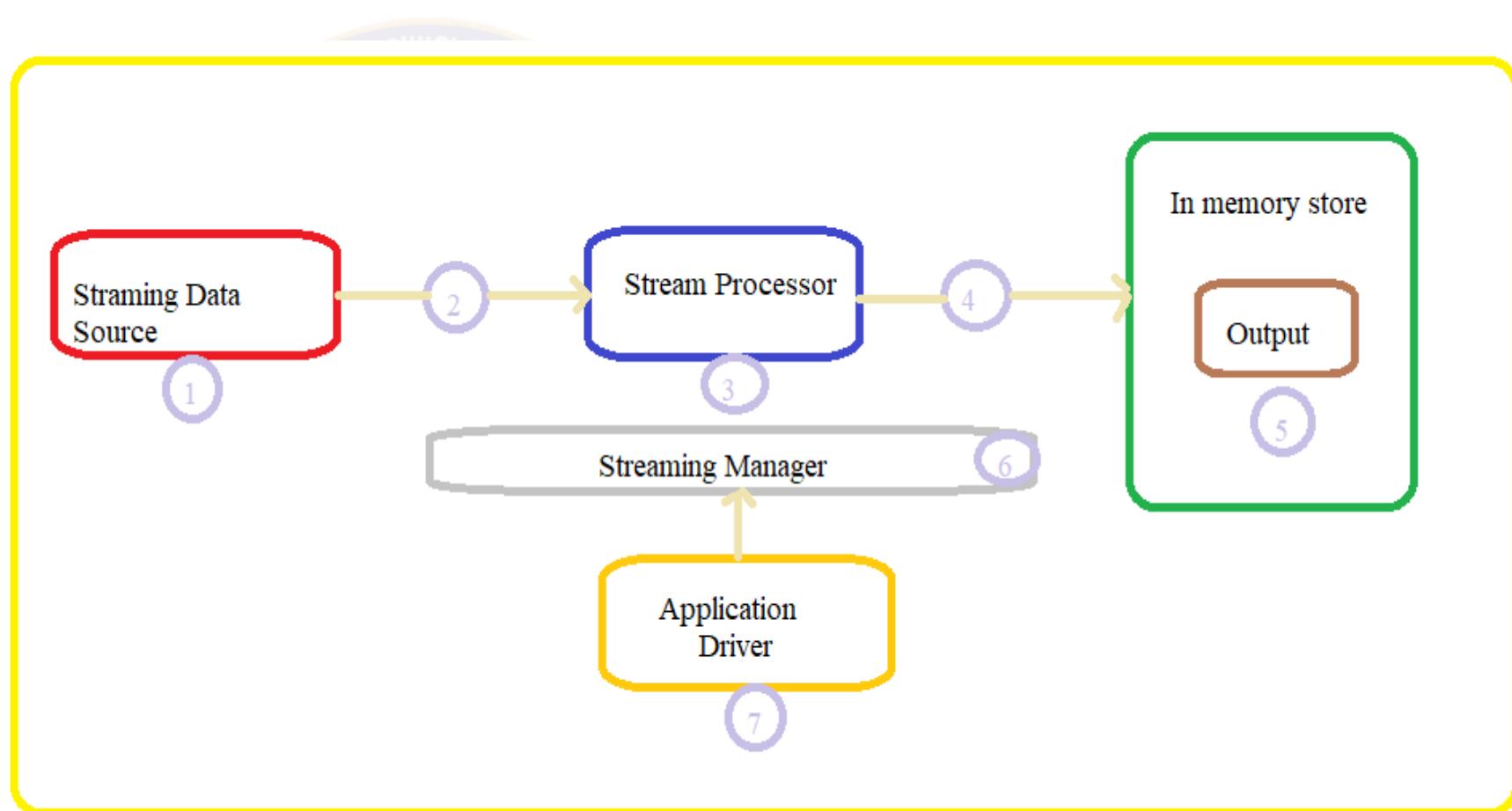
- In memory
 - ✓ For example, sum(sales) for last one hour
 - ✓ Data will be flushed out once one hour window expires
 - ✓ Potential of losing all the data if the stream processor node fails
- Persistent storage
 - ✓ For example, needs to join data from two different streams and data being produced at different rate



Fault Tolerance

Point of failures in Stream Processing data flow

- Incoming stream data
- Network carrying data
- Stream Processor
- Connection to output sink
- Output destination
- Streaming Manager
- Application Driver



Fault Tolerance(2)

- Data loss
 - ✓ Data loss over network, job crashes etc.
 - ✓ Replication and Coordination
 - ❖ replicate the state of computation on multiple nodes
 - ❖ In case of failures, streaming manager interacts with replicas
 - ❖ Can predefine number of simultaneous failures
 - ❖ K- fault tolerant where k is number of simultaneous failures
 - ✓ State machine approach
 - ❖ Streaming job is replicated on multiple nodes
 - ❖ Replicas are coordinated by sending same input in same order to all
 - ❖ Allows for quick failover , resulting into little disruption
 - ✓ Rollback recovery approach
 - ❖ Stream processor periodically packages the state of computation into checkpoint
 - ❖ Copies checkpoint to different nodes
 - ❖ In case of failure, stream manager fetches the computation from last saved checkpoint and reschedules it



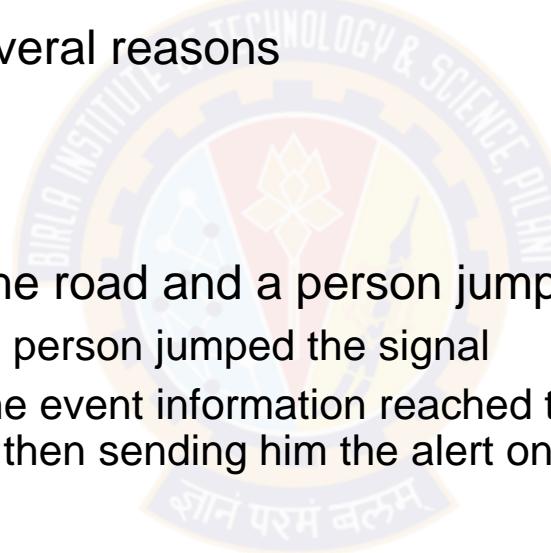
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Timing Concepts

Pravin Y Pawar

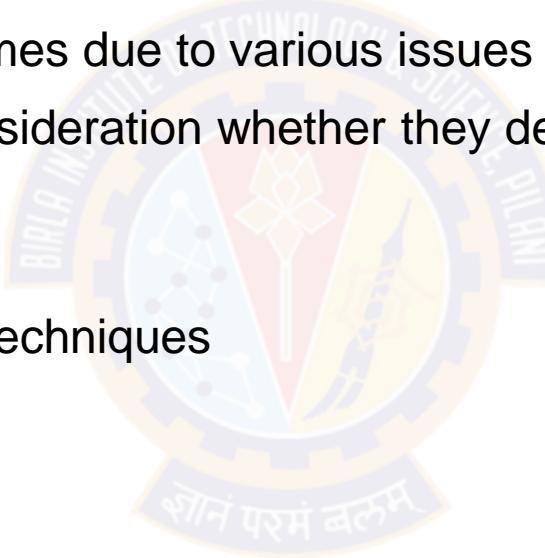
Event Time vs. Stream Time

- Event time is time at which the event occurs
 - Stream time is time at which event enters the streaming system
 - Stream time will lag a bit due to several reasons
-
- For example,
 - If we are monitoring the traffic on the road and a person jumps the signal, then
 - ✓ event time is the time at which the person jumped the signal
 - ✓ stream time is the time at which the event information reached to streaming platform for processing like snapping the fine on the user and then sending him the alert on the mobile



Time Skew

- Stream time is often lagging behind the event time
- The difference between these two times is “time skew”
- Variance can be significant sometimes due to various issues
- Applications needs to take into consideration whether they depend on the event time or stream time
- Needs to bother about windowing techniques





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

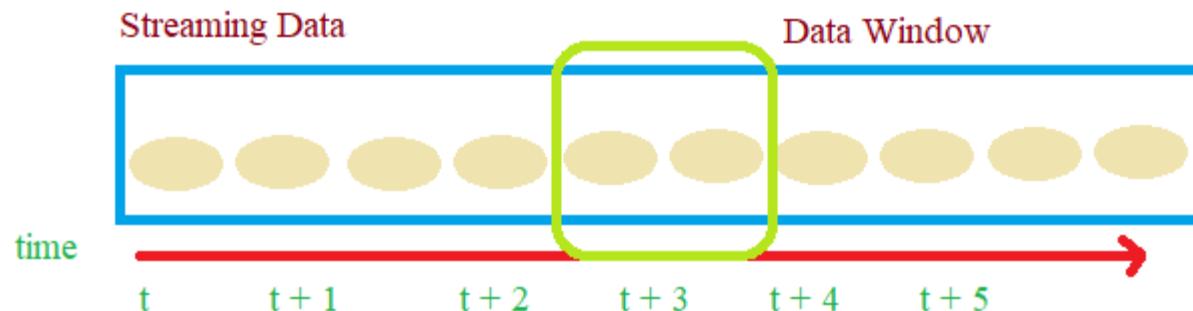
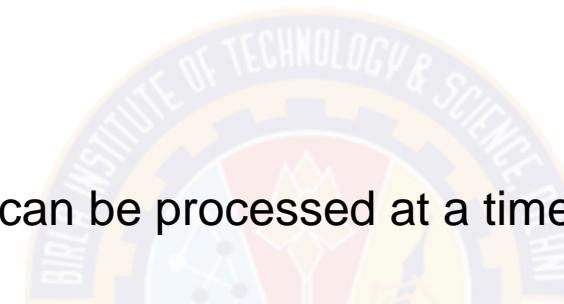
Windowing

Pravin Y Pawar

Windows

Defined

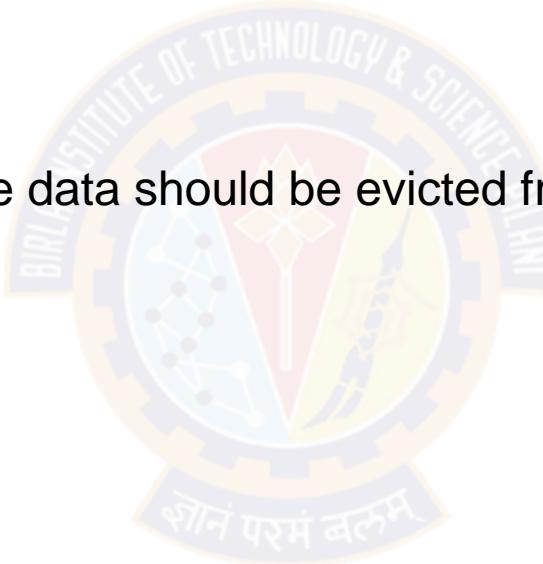
- Streaming data can not be completely stored in the memory
- Then how to process it ?
- Windows of data
 - ✓ A certain amount of data that can be processed at a time



Windowing (2)

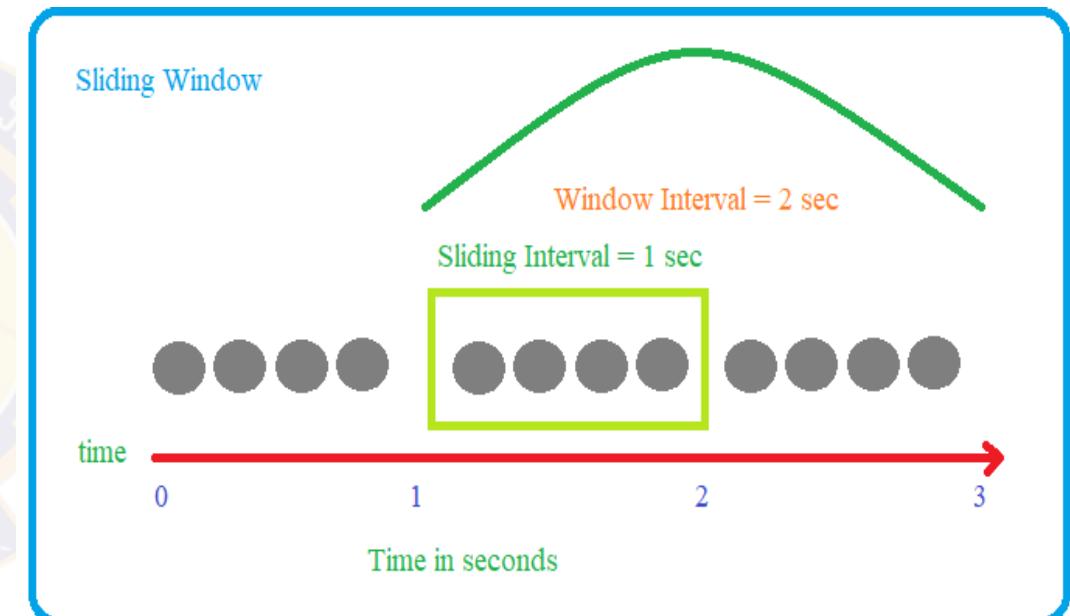
Policies and Types

- Trigger policy
 - ✓ Rules for determining when the code should be executed
- Eviction policy
 - ✓ Rules for determining when the data should be evicted from the window
- Can be time based or count based
- Types
 - ✓ Sliding window
 - ✓ Tumbling window



Sliding window

- Uses the windowing technique based on time
- Window length specifies the eviction policy
 - ✓ Time duration for which data is available for processing
- Sliding interval specifies the trigger policy
 - ✓ Time duration after which code will be triggered
 - ✓ If Window length is 2 seconds, then data older than 2 seconds will be evicted
 - ✓ If sliding interval is 1 second, after every second the code will be executed



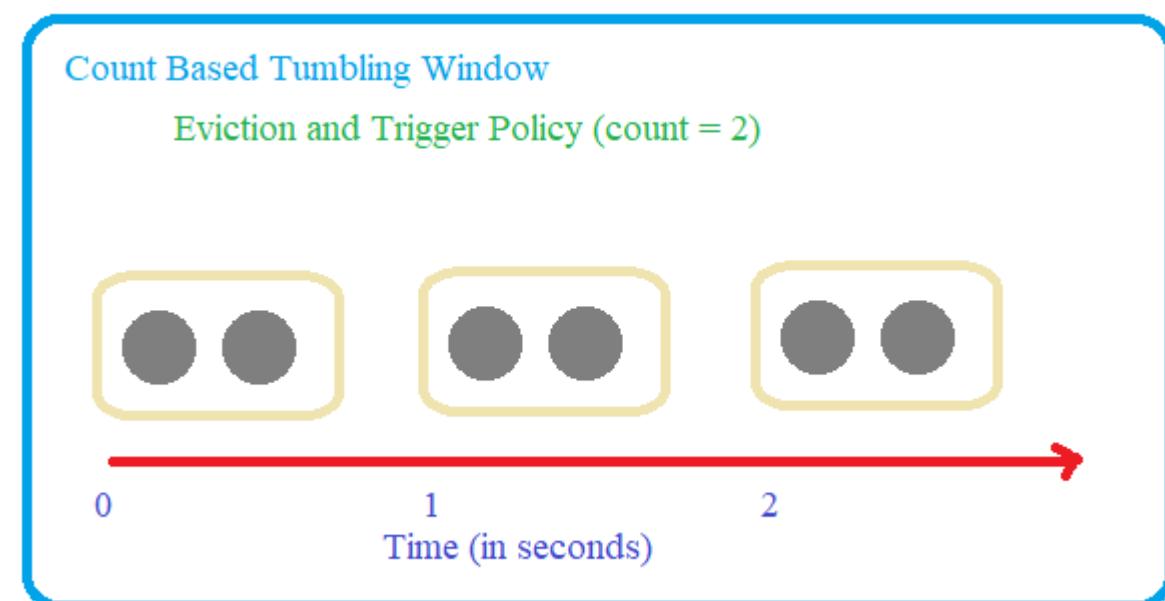
Tumbling Window

- Eviction policy based on window being full
- Trigger policy based on time or number of events in window
- Types
 - ✓ Count based Tumbling
 - ✓ Temporal based Tumbling



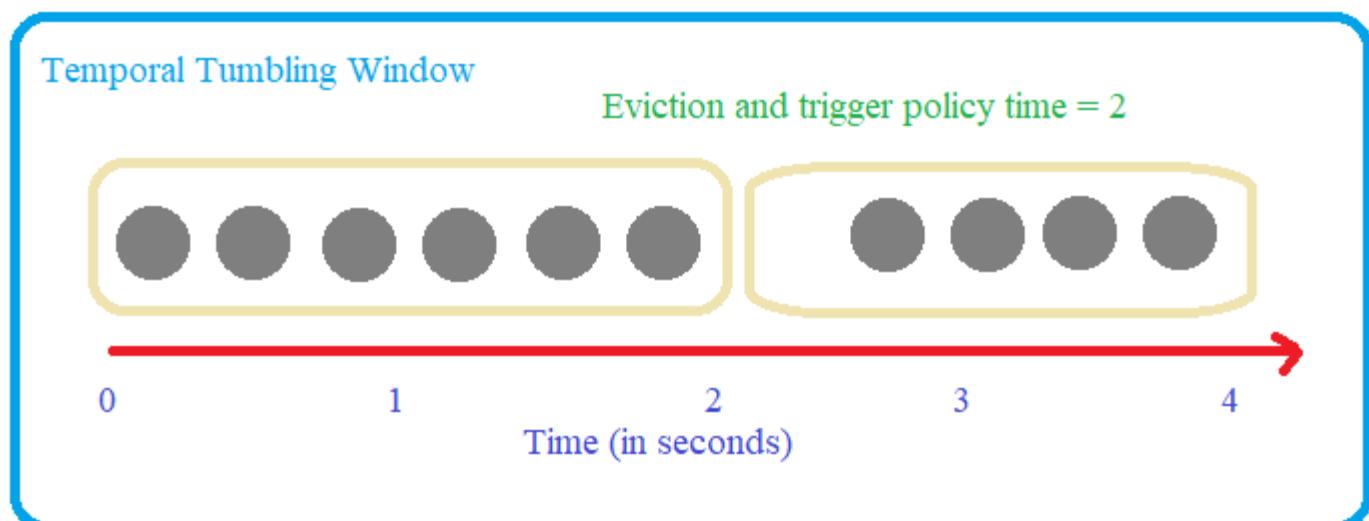
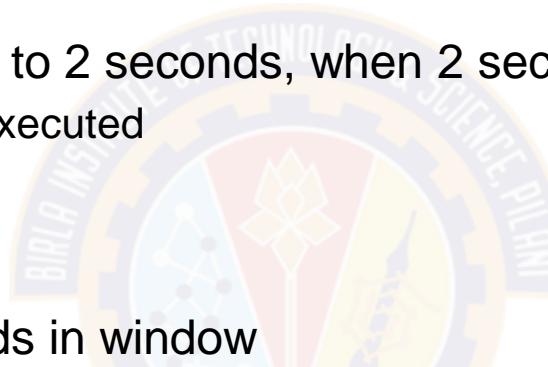
Count based Tumbling Window

- Trigger and eviction policies will be executed when window is full
- If the eviction and trigger policies are two , when two events are accumulated in window:
 - ✓ Trigger will be fired , code will be executed
 - ✓ Window will be drained
- No time constraints for window to be filled



Temporal Tumbling Window

- Based on time
- If the eviction and trigger policy set to 2 seconds, when 2 seconds are over :
 - ✓ Trigger will be fired, code will be executed
 - ✓ Window will be drained
- No constraints on number of records in window





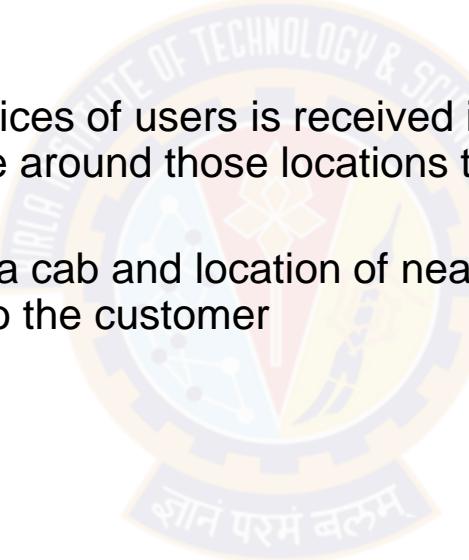
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Stream Joins

Pravin Y Pawar

Stream Joins

- Similar to joins in batch processing, multiple streams can also be joined
- Sources of those streams can be same or different
- For example,
 - ✓ Location coordinates of mobile devices of users is received in real time and can be joined with the traffic updates available in real time around those locations to suggest an alternative route in case of congestion
 - ✓ Location of the user trying to book a cab and location of nearby cabs can be joined to determine which can be made available at earliest to the customer
- Types
 - ✓ Stream- stream join
 - ✓ Stream – table join



Stream – stream join (windowing)

- Event streams from two or more sources are joined in real time
- Usually takes events occurred in certain time period i.e. time window is attached
- Correlate the events happened in last 5 minutes
- For example,
 - ✓ User is searching for a product on a ecommerce portal , all his search queries can be tracked and sourced as stream
 - ✓ At the same time, whatever clicks, the same user is producing on the product related web pages can be tracked and sourced as another stream
 - ✓ Those two event streams can be joined to offer a discount to the user to increase the chances of purchase of those items
- The state needs to be maintained by streaming application like what events arrived since user started a session

Stream – table join (enrichment)

- Streaming event and data stored in persistent storage like database is joined together
- Usually information from the database is fetched to add more value to the streaming data
- For example,
 - ✓ Users location is tracked near a mall in real time along with identity of user and sourced as stream
 - ✓ At the same time, profile data of user is fetched from database, his / her interests are observed
 - ✓ Based on the interests and outlets located in the malls, discount coupons can be sent on the users devices
- Lookup in database can be time consuming in case of too many records of users are available
- Local copy of users data can be loaded in stream processor
 - ✓ Needs to keep it updated



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

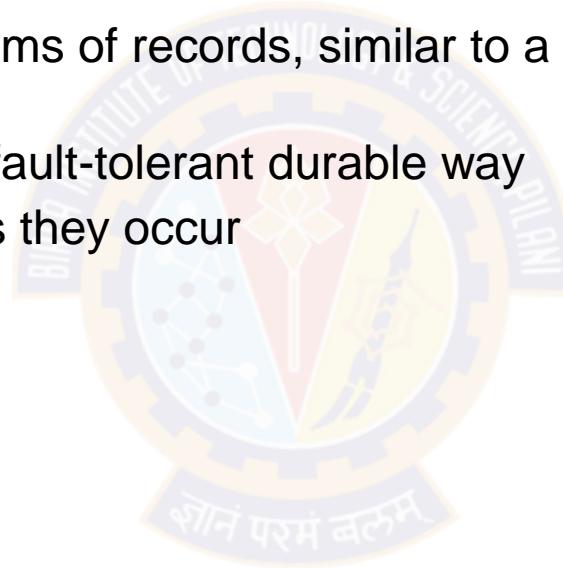
Apache Kafka Streaming

Pravin Y Pawar

Streaming Platform

Features

- A streaming platform has three key capabilities:
 - ✓ Publish and subscribe to streams of records, similar to a message queue or enterprise messaging system
 - ✓ Store streams of records in a fault-tolerant durable way
 - ✓ Process streams of records as they occur



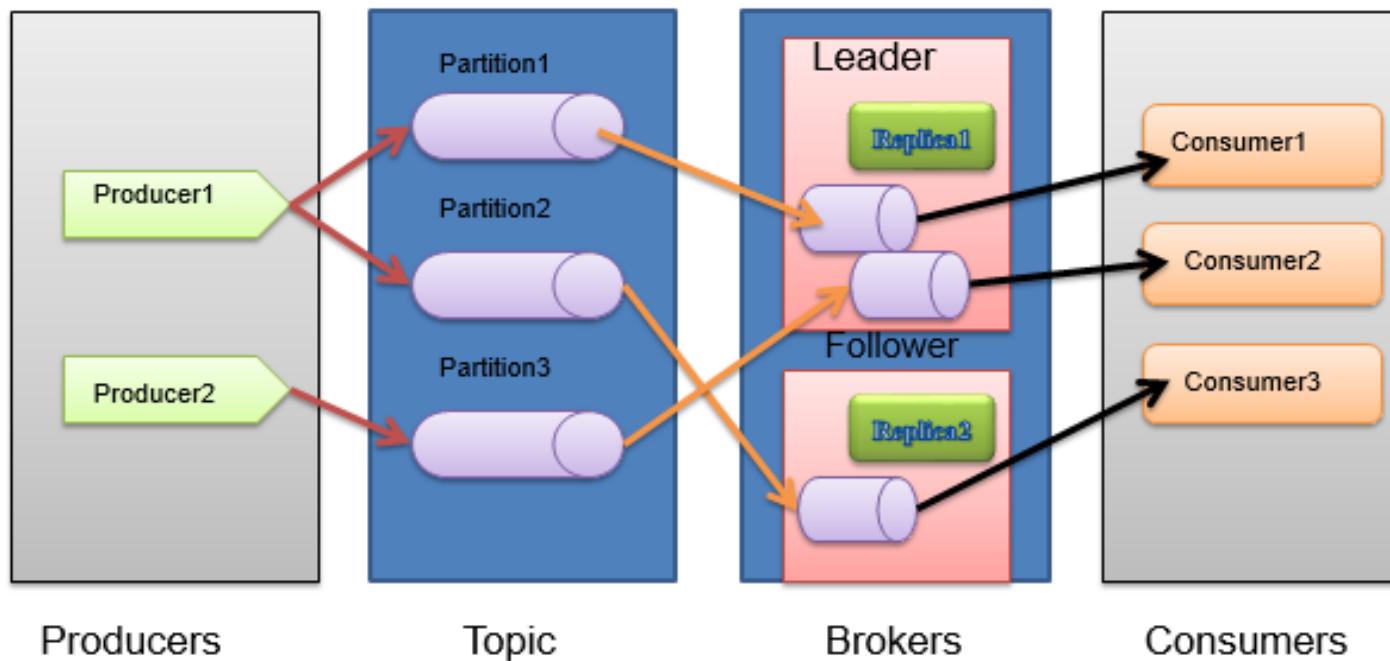
Apache Kafka

- Distributed pub-sub and queuing messaging system
- Designed for high throughput distributed systems
 - ✓ Better throughput
 - ✓ In-built partitioning mechanism
 - ✓ Replicas
 - ✓ Fault Tolerant
- Good fit for large scale data processing
- Suitable for both offline and online message transfer
- Messages are persisted on disk and replicated across cluster to prevent message loss
- Integrate with Storm and Spark easily



Kafka Fundamentals

- Kafka Architecture



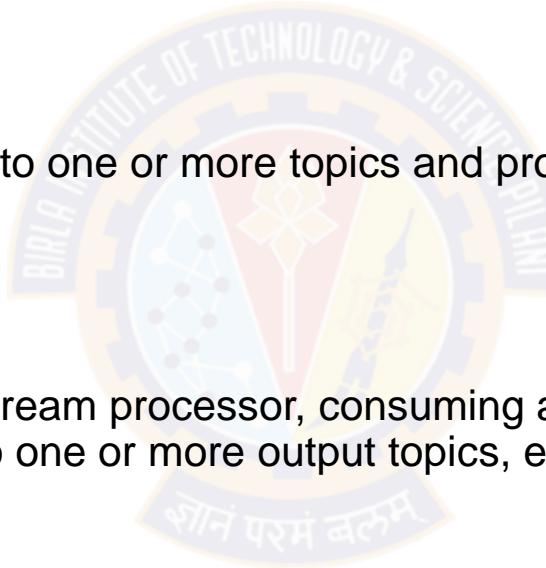
Kafka Concepts

- Kafka is run as a cluster on one or more servers that can span multiple datacenters.
- The Kafka cluster stores streams of records in categories called topics.
- Each record consists of a key, a value, and a timestamp.



Kafka APIs

- The Producer API
 - ✓ allows an application to publish a stream of records to one or more Kafka topics.
- The Consumer API
 - ✓ allows an application to subscribe to one or more topics and process the stream of records produced to them.
- The Streams API
 - ✓ allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
- The Connector API
 - ✓ allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.



Kafka for Stream Processing

- It isn't enough to just read, write, and store streams of data, the purpose is to enable real-time processing of streams.
- In Kafka a stream processor is anything that
 - ✓ takes continual streams of data from input topics
 - ✓ performs some processing on this input
 - ✓ produces continual streams of data to output topics
- For example, a retail application might take in input streams of sales and shipments, and output a stream of reorders and price adjustments computed off this data.
- It is possible to do simple processing directly using the producer and consumer APIs. However for more complex transformations Kafka provides a fully integrated Streams API.
- The streams API builds on the core primitives Kafka provides
 - ✓ it uses the producer and consumer APIs for input
 - ✓ uses Kafka for stateful storage
 - ✓ uses the same group mechanism for fault tolerance among the stream processor instances

Kafka Streams API

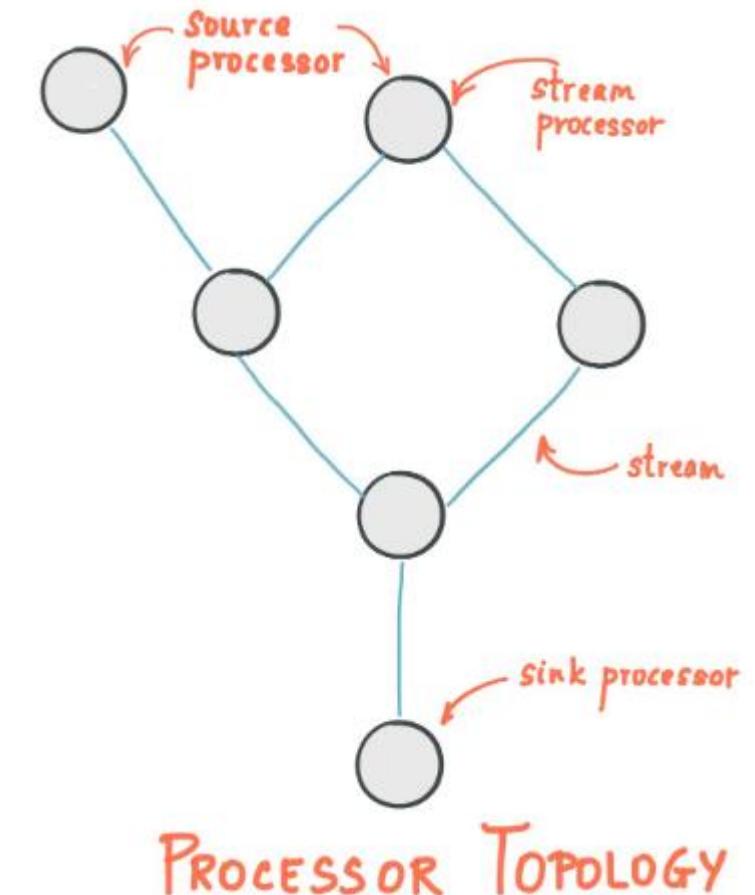
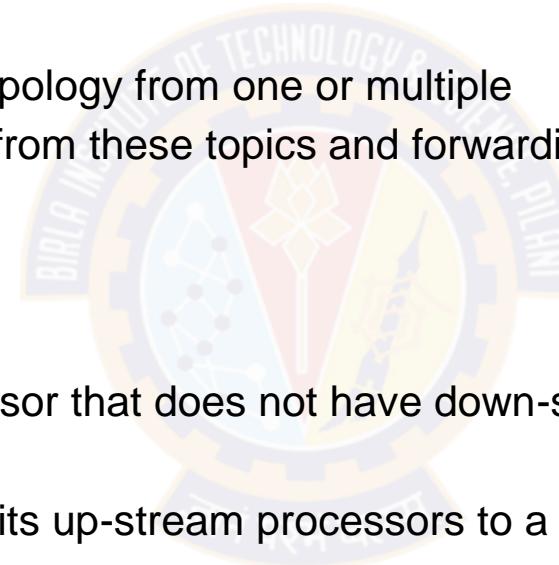
- Can build highly scalable, elastic, distributed, and fault-tolerant application
- Provides Stateful and stateless processing
- Can perform Event-time processing with windowing, joins, and aggregations
- Can use the already-defined most common transformation operation using Kafka Streams DSL or the lower-level processor API, which allow us to define and connect custom processors
- Low barrier to entry
 - ✓ does not take much configuration and setup to run a small scale trial of stream processing; the rest depends on your use case
- No separate cluster requirements for processing (integrated with Kafka)
- Employs one-record-at-a-time processing to achieve millisecond processing latency
 - ✓ supports event-time based windowing operations with the late arrival of records.
- Supports Kafka Connect to connect to different applications and databases.

Stream Processing Topology

- A stream
 - ✓ is the most important abstraction provided by Kafka Streams
 - ✓ represents an unbounded, continuously updating data set
 - ✓ is an ordered, replayable, and fault-tolerant sequence of immutable data records, where a data record is defined as a key-value pair.
- A stream processing application
 - ✓ is any program that makes use of the Kafka Streams library
 - ✓ defines its computational logic through one or more processor topologies, where a processor topology is a graph of stream processors (nodes) that are connected by streams (edges).
- A stream processor
 - ✓ is a node in the processor topology
 - ✓ represents a processing step
 - to transform data in streams by receiving one input record at a time from its upstream processors in the topology
 - applying its operation to it
 - may subsequently produce one or more output records to its downstream processors

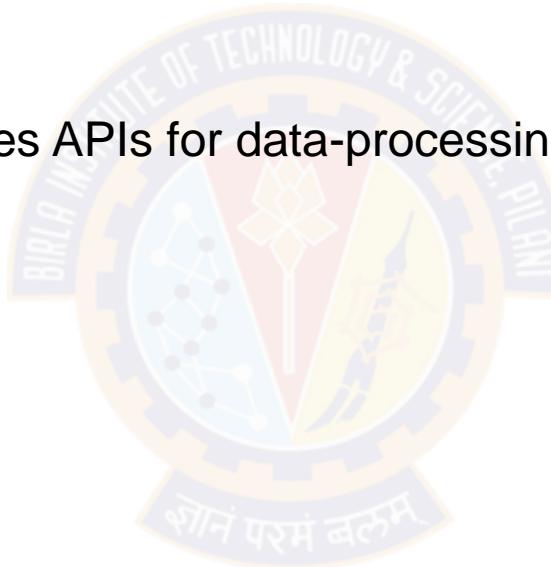
Kafka Stream Processors

- Source Processor
 - ✓ is a special type of stream processor that does not have any upstream processors
 - ✓ produces an input stream to its topology from one or multiple Kafka topics by consuming records from these topics and forwarding them to its down-stream processors
- Sink Processor
 - ✓ is a special type of stream processor that does not have down-stream processors
 - ✓ sends any received records from its up-stream processors to a specified Kafka topic



Processing in Kafka Streams

- Two options available for processing stream data:
- High-level Kafka Streams DSL
- A lower-level processor that provides APIs for data-processing, composable processing, and local state storage



Processing in Kafka Streams(2)

High-Level DSL

- Has already implemented methods ready to use
- Composed of two main abstractions: KStream and KTable or GlobalKTable.
- KStream
 - ✓ is an abstraction of record stream where each data is a simple key value pair in the unbounded dataset
 - ✓ provides us many functional ways to manipulate stream data like
 - map
 - mapValue
 - flatMap
 - flatMapValues
 - filter
 - provides joining methods for joining multiple streams and aggregation methods on stream data
- KTable or GlobalKTable
 - ✓ is an abstraction of a changelog stream
 - ✓ every data record is considered an Insert or Update (Upsert) depending upon the existence of the key as any existing row with the same key will be overwritten

Processing in Kafka Streams(3)

Processor API

- The low-level Processor API
 - ✓ provides a client to access stream data
 - ✓ to perform our business logic on the incoming data stream
 - ✓ send the result as the downstream data
- allows extending the abstract class AbstractProcessor and overriding the process method which contains our logic
 - ✓ this process method is called once for every key-value pair
- The high-level DSL provides ready to use methods with functional style
- The low-level processor API provides you the flexibility to implement processing logic according to your need
 - ✓ The trade-off is just the lines of code you need to write for specific scenarios



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Apache Spark Streaming

Pravin Y Pawar

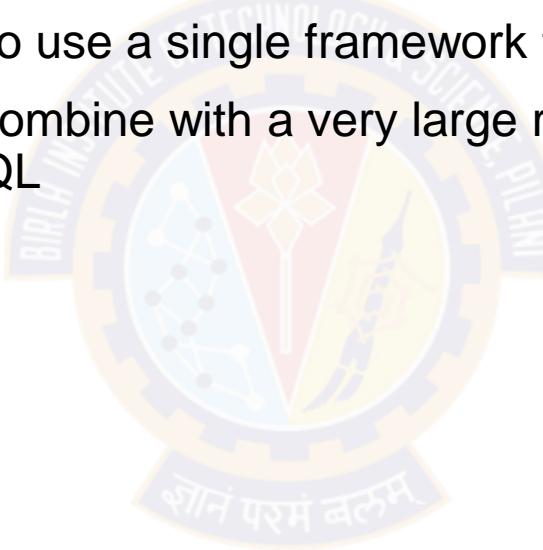
Continuous operator model

Working

- A simple and natural model
 - ✓ Streaming data is received from data sources (e.g. live logs, system telemetry data, IoT device data, etc.) into some data ingestion system like Apache Kafka, Amazon Kinesis, etc.
 - ✓ The data is then processed in parallel on a cluster.
 - ✓ Results are given to downstream systems like HBase, Cassandra, Kafka, etc.
- ✓ Data is received from ingestion systems via Source operators and given as output to downstream systems via sink operators
- ✓ Cluster has set of worker nodes, each of which runs one or more continuous operators
- ✓ Each continuous operator processes the streaming data one record at a time and forwards the records to other operators in the pipeline

Why Spark Streaming?

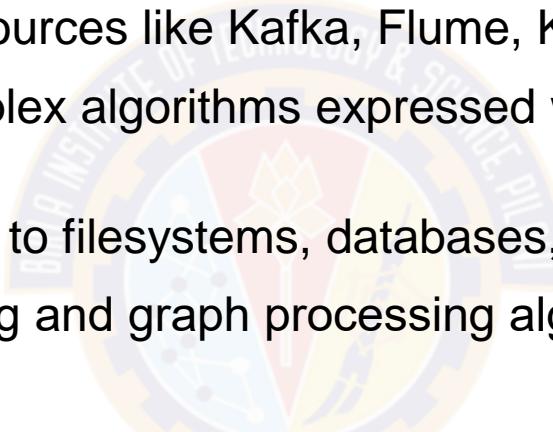
- Provides the unification of disparate data processing capabilities
- Has a unified engine that natively supports both batch and streaming workloads
- Makes it very easy for developers to use a single framework to satisfy all the processing needs
- Data from streaming sources can combine with a very large range of static data sources available through Apache Spark SQL



Spark Streaming

Overview

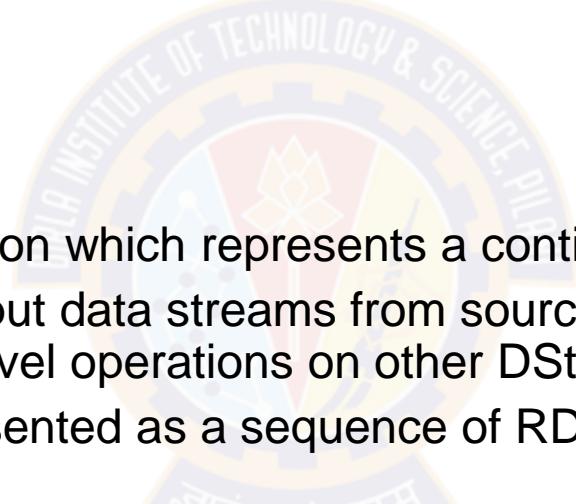
- An extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams
- Data can be ingested from many sources like Kafka, Flume, Kinesis, or TCP sockets
- Data can be processed using complex algorithms expressed with high-level functions like map, reduce, join and window
- Processed data can be pushed out to filesystems, databases, and live dashboards
- Can apply Spark's machine learning and graph processing algorithms on data streams



Spark Streaming (2)

High Level Working

- Spark Streaming receives live input data streams and divides the data into batches
- These batches are then processed by the Spark engine to generate the final stream of results in batches.
- DStreams
 - Provides a high-level abstraction which represents a continuous stream of data
 - Can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams
 - Internally, a DStream is represented as a sequence of RDDs



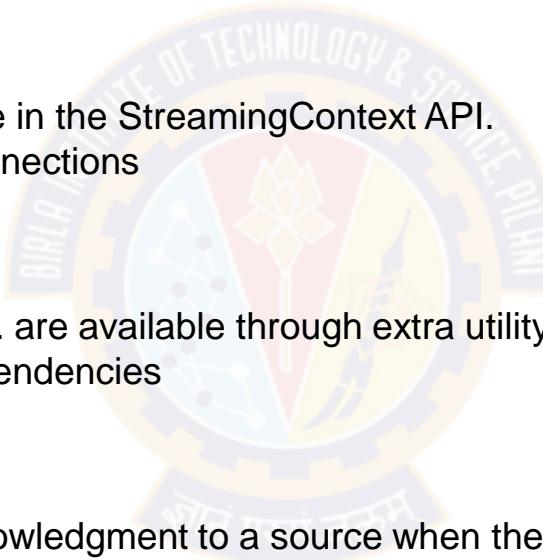
Spark Streaming (3)

Micro-batching

- Instead of processing the streaming data one record at a time, Spark Streaming discretizes the data into tiny, sub-second micro-batches
 - Spark Streaming receivers accept data in parallel and buffer it in the memory of Spark's workers nodes
 - The latency-optimized Spark engine runs short tasks to process the batches and output the results to other systems.
-
- Spark tasks are assigned to the workers dynamically on the basis of data locality and available resources.
 - ✓ Enables better load balancing and faster fault recovery
 - Each batch of data is a Resilient Distributed Dataset (RDD) in Spark, which is the basic abstraction of a fault-tolerant dataset in Spark
 - ✓ Allows the streaming data to be processed using any Spark code or library

Spark Streaming Sources & Receivers

- Every input DStream (except file stream) associate with a Receiver object which receives the data from a source and stores it in Spark's memory for processing.
- Basic sources
 - ✓ These are the sources directly available in the StreamingContext API.
 - ✓ Examples: file systems, and socket connections
- Advanced sources
 - ✓ Sources like Kafka, Flume, Kinesis, etc. are available through extra utility classes
 - ✓ These require linking against extra dependencies
- Reliable Receiver
 - ✓ Is the one that correctly sends an acknowledgment to a source when the data receives and stores in Spark with replication
- Unreliable Receiver
 - ✓ Is the one that does not send an acknowledgment to a source



Spark Streaming Operations

Transformation Operations

- Spark transformations allow modification of the data from the input Dstream

Transformation	Meaning
<code>map(func)</code>	Return a new DStream by passing each element of the source DStream through a function <i>func</i> .
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items.
<code>filter(func)</code>	Return a new DStream by selecting only the records of the source DStream on which <i>func</i> returns true.
<code>repartition(numPartitions)</code>	Changes the level of parallelism in this DStream by creating more or fewer partitions.
<code>union(otherStream)</code>	Return a new DStream that contains the union of the elements in the source DStream and <i>otherDStream</i> .
<code>count()</code>	Return a new DStream of single-element RDDs by counting the number of elements in each RDD of the source DStream.
<code>reduce(func)</code>	Return a new DStream of single-element RDDs by aggregating the elements in each RDD of the source DStream using a function <i>func</i> (which takes two arguments and returns one). The function should be associative and commutative so that it can be computed in parallel.
<code>countByValue()</code>	When called on a DStream of elements of type K, return a new DStream of (K, Long) pairs where the value of each key is its frequency in each RDD of the source DStream.

Spark Streaming Operations(2)

Output Operations

- Output operations allow DStream's data to be pushed out to external systems like a database or a file systems

Output Operation	Meaning
<code>print()</code>	Prints the first ten elements of every batch of data in a DStream on the driver node running the streaming application. This is useful for development and debugging. <small>Python API</small> This is called <code>pprint()</code> in the Python API.
<code>saveAsTextFiles(prefix, [suffix])</code>	Save this DStream's contents as text files. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ".
<code>saveAsObjectFiles(prefix, [suffix])</code>	Save this DStream's contents as <code>SequenceFiles</code> of serialized Java objects. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ". <small>Python API</small> This is not available in the Python API.
<code>saveAsHadoopFiles(prefix, [suffix])</code>	Save this DStream's contents as Hadoop files. The file name at each batch interval is generated based on <code>prefix</code> and <code>suffix</code> : " <code>prefix-TIME_IN_MS[.suffix]</code> ". <small>Python API</small> This is not available in the Python API.
<code>foreachRDD(func)</code>	The most generic output operator that applies a function, <code>func</code> , to each RDD generated from the stream. This function should push the data in each RDD to an external system, such as saving the RDD to files, or writing it over the network to a database. Note that the function <code>func</code> is executed in the driver process running the streaming application, and will usually have RDD actions in it that will force the computation of the streaming RDDs.



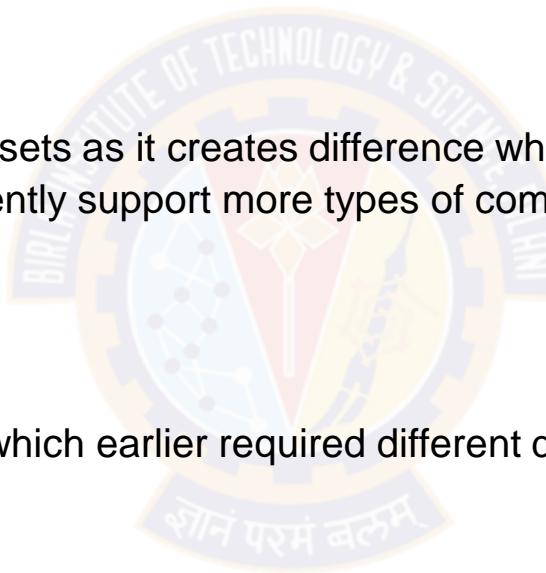
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Apache Spark

Pravin Y Pawar

Spark Overview

- Cluster computing platform
 - ✓ Designed to be fast and general purpose
- Speed
 - ✓ Is important in processing large datasets as it creates difference when data is being explored
 - ✓ Extends MapReduce model to efficiently support more types of computations
 - ✓ Runs computations in memory
- Generality
 - ✓ Covers a wide variety of workloads which earlier required different distributed systems
 - ✓ Including
 - ❖ Batch applications
 - ❖ Iterative algorithms
 - ❖ Interactive queries
 - ❖ Streaming
 - ✓ Easy and inexpensive to combine different processing types in data pipelines



Spark Overview (2)

- Spark handles highly accessible, simple APIs in

- ✓ Java
- ✓ Python
- ✓ Scala
- ✓ SQL



- Integrates easily with other big data tools like
 - ✓ Hadoop (HDFS) and other ecosystem tools
 - ✓ Kafka
 - ✓ AWS



Unified Stack

Spark SQL
structured data

Spark Streaming
real-time

MLib
machine
learning

GraphX
graph
processing

Spark Core

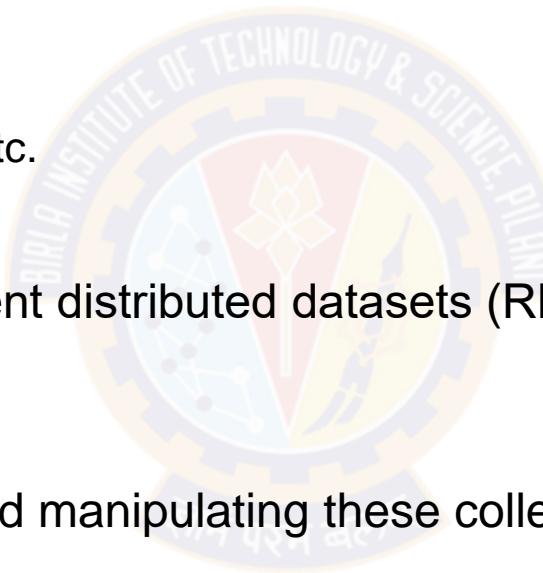
Standalone Scheduler

YARN

Mesos

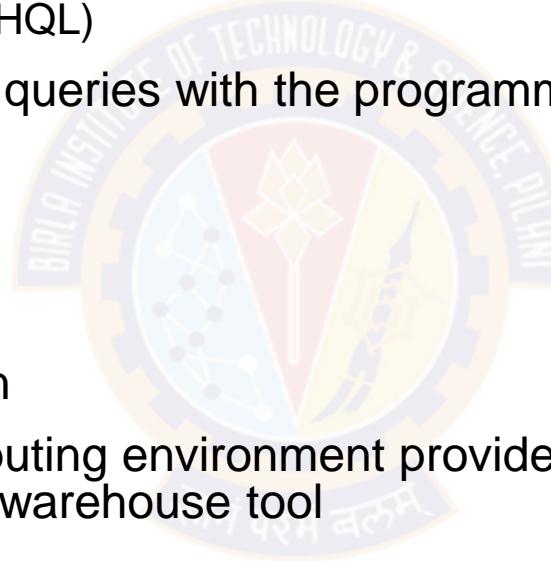
Spark Core

- Contains the basic functionality of Spark, including components for
 - ✓ task scheduling
 - ✓ memory management
 - ✓ fault recovery
 - ✓ interacting with storage systems etc.
- Home to the API that defines resilient distributed datasets (RDDs), which are Spark's main programming abstraction
- Provides many APIs for building and manipulating these collections



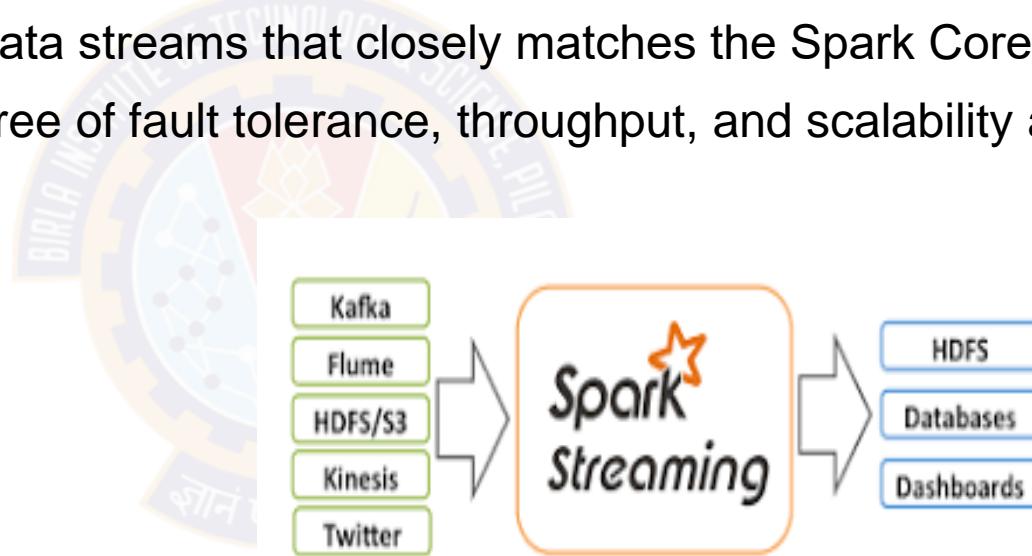
Spark SQL

- Spark's package for working with structured data
- Allows querying data via SQL as well as the Apache Hive variant of SQL
 - ✓ called the Hive Query Language (HQL)
- Allows developers to intermix SQL queries with the programmatic data manipulations
 - supported by RDDs in
 - ✓ Python
 - ✓ Java
 - ✓ Scala
 - all within a single application
- Tight integration with the rich computing environment provided by Spark makes Spark SQL unlike any other open source data warehouse tool



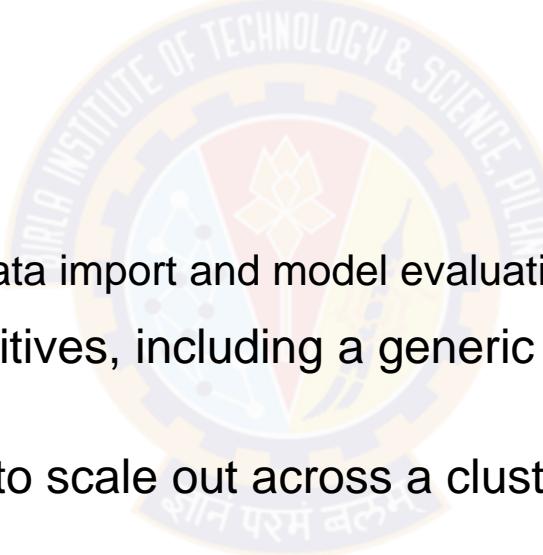
Spark Streaming

- Spark component that enables processing of live streams of data
- Examples of data streams include web servers log files
- Provides an API for manipulating data streams that closely matches the Spark Core's RDD API
- Designed to provide the same degree of fault tolerance, throughput, and scalability as Spark Core



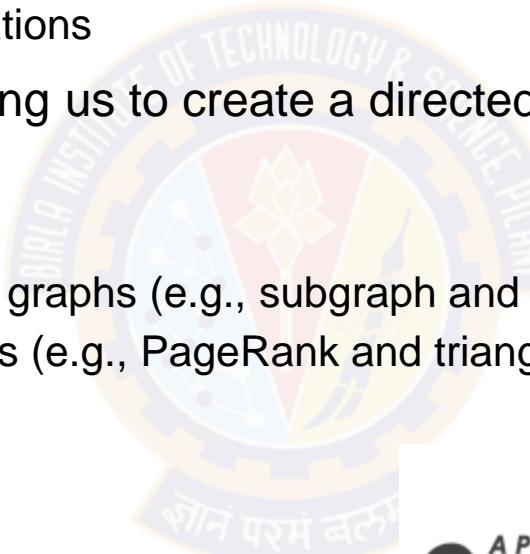
MLlib

- Built in library containing common machine learning (ML) functionality
- Provides multiple types of machine learning algorithms, including
 - ✓ Classification
 - ✓ Regression
 - ✓ Clustering
 - ✓ Collaborative filtering
 - ✓ Supporting functionality such as data import and model evaluation
- Provides some lower-level ML primitives, including a generic gradient descent optimization algorithm
- All of these methods are designed to scale out across a cluster



GraphX

- Library for
 - ✓ manipulating graphs (e.g. a social network's relations graph)
 - ✓ performing graph-parallel computations
- Extends the Spark RDD API, allowing us to create a directed graph with arbitrary properties attached to each vertex and edge
- Provides
 - ✓ various operators for manipulating graphs (e.g., subgraph and mapVertices)
 - ✓ library of common graph algorithms (e.g., PageRank and triangle counting)



APACHE  GraphX

Cluster Managers

- Spark is designed to efficiently scale up from one to many thousands of compute nodes
- Can run over a variety of cluster managers, including
 - ✓ Hadoop YARN
 - ✓ Apache Mesos
 - ✓ Simple cluster manager included in Spark itself called the Standalone Scheduler
 - ✓ Kubernetes
- If you are just installing Spark on an empty set of machines
 - ✓ the Standalone Scheduler provides an easy way to get started
- If you already have a Hadoop YARN or Mesos cluster,
 - ✓ Spark's support for these cluster managers allows your applications to also run on them





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark – Getting Started

Pravin Y Pawar

Getting Started

- Spark can be used from Python, Java or Scala
- Spark is written in Scala and runs on JVM
- Spark can be run in local mode (standalone) or cluster mode
- Spark can be run on both Windows and Unix like systems
- Requirements
 - ✓ It's easy to run locally on one machine — all you need is to have java installed on your system PATH, or the JAVA_HOME environment variable pointing to a Java installation
 - ✓ Spark runs on Java 8, Python 2.7+/3.4+ and R 3.1+. For the Scala API, Spark 2.4.5 uses Scala 2.12. You will need to use a compatible Scala version (2.12.x).

Getting Started (2)

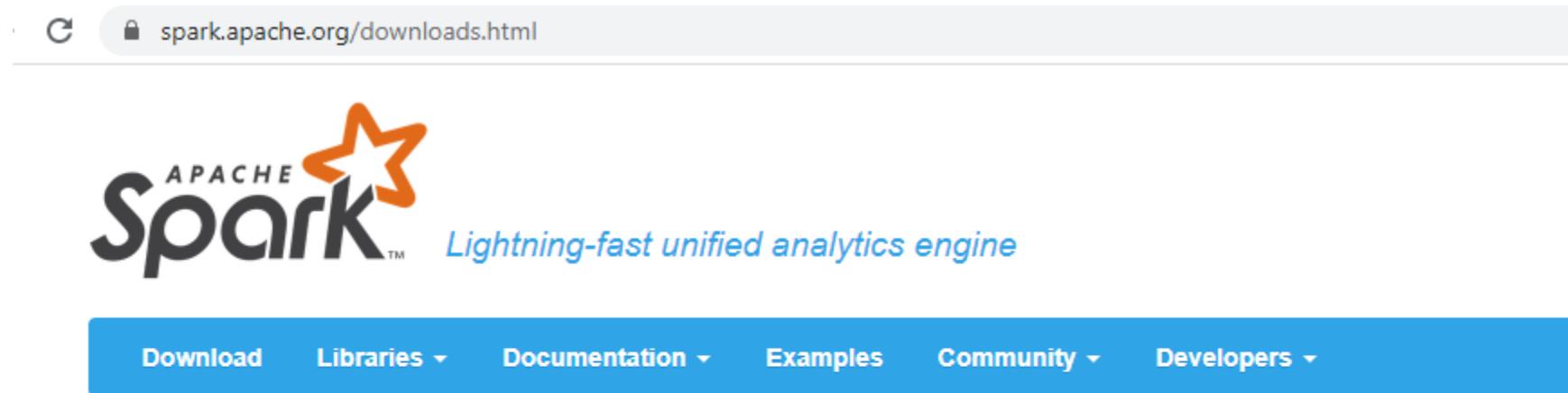
Download

- Needs to download and unpack the tarball
- Go to the Spark homepage to download the Spark release
 - ✓ <https://spark.apache.org/downloads.html>
- Steps -
 - ✓ Choose a Spark release:
 - ✓ Choose a package type:
 - ✓ Download Spark: spark-3.0.0-preview2-bin-hadoop2.7.tgz
 - ✓ Verify this release using the 3.0.0-preview2 signatures, checksums and project release KEYS.



Getting Started (3)

Download



The screenshot shows the Apache Spark download page at spark.apache.org/downloads.html. The page features the Apache Spark logo and tagline "Lightning-fast unified analytics engine". A blue navigation bar at the top includes links for Download, Libraries, Documentation, Examples, Community, and Developers. Below the navigation bar, a section titled "Download Apache Spark™" provides step-by-step instructions for downloading the software. Step 1: Choose a Spark release (dropdown menu set to 3.0.0-preview2 (Dec 23 2019)). Step 2: Choose a package type (dropdown menu set to Pre-built for Apache Hadoop 2.7). Step 3: Download Spark: [spark-3.0.0-preview2-bin-hadoop2.7.tgz](#). Step 4: Verify this release using the 3.0.0-preview2 [signatures](#), [checksums](#) and [project release KEYS](#). A note at the bottom states: "Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12."

1. Choose a Spark release: 3.0.0-preview2 (Dec 23 2019) ▾

2. Choose a package type: Pre-built for Apache Hadoop 2.7 ▾

3. Download Spark: [spark-3.0.0-preview2-bin-hadoop2.7.tgz](#)

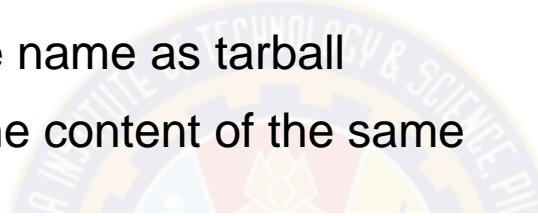
4. Verify this release using the 3.0.0-preview2 [signatures](#), [checksums](#) and [project release KEYS](#).

Note that, Spark is pre-built with Scala 2.11 except version 2.4.2, which is pre-built with Scala 2.12.

Getting Started (4)

Unpack

- The tarball .tgz will get downloaded
- Open the terminal and unzip the tar ball
- Result into new directory with same name as tarball
- Change into the directory and list the content of the same



```
Applications Places Terminal Wed 12:35
[csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7]
File Edit View Search Terminal Help
[csishydlab@apache-spark ~]$ ls Downloads/
CentOS-7-x86_64-DVD-1908.torrent      Spark-DataFrame(1).html
CentOS-7-x86_64-Everything-1908.iso   Spark-DataFrame.html
kafka_2.12-2.4.0.tgz                 Spark-DataFrame.ipynb
kafka_2.12-2.4.1.tgz                 spark-streaming-kafka-0-8-assembly_2.11-2.4.4(1).jar
[csishydlab@apache-spark ~]$ cd ~
[csishydlab@apache-spark ~]$ ls
data      kafka_2.12-2.4.0    output      scala-2.10.1.tgz
Desktop   kafka_2.12-2.4.0.tgz Pictures    spark-2.4.4-bin-hadoop2.7
Documents  logs                Public     spark-2.4.4-bin-hadoop2.7.tgz
Downloads Music               python_code  spark_kafka_demo.logs
[csishydlab@apache-spark ~]$ cd spark-2.4.4-bin-hadoop2.7/
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ ls
bin      data      jars      LICENSE  NOTICE  R      regression_metrics_example.py  sample_linear_regression_data.txt  yarn
conf    examples  kubernetes  licenses  python  README.md  RELEASE
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$
```

Getting Started (5)

Spark Shells

- Interactive shell available for interactive data analysis
 - ✓ Similar to other shells like R , Bash or Windows command prompt
 - Allows to interact with data that is distributed on disk or in memory across many machines
 - Provides both Python and Scala shells
 - ✓ Python – execute bin/pyspark
 - ✓ Scala – execute bin/spark-shell

is distributed on disk or in memory across many machines

shells



```
Applications Places Terminal
csishydlab@apache-spark:~/spark-2.4.4-bin-hadoop2.7
File Edit View Search Terminal Help
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ bin/pyspark
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
20/04/15 12:40:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to

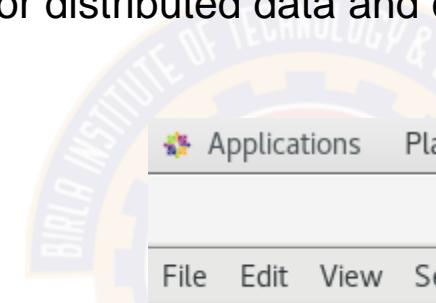
    / \
   /   \
  /     \
 /       \
/         \
 \       /
  \     /
   \   /
    \ /
     \
version 2.4.4

Using Python version 2.7.5 (default, Aug 7 2019 00:51:29)
SparkSession available as 'spark'.
>>>
```

Getting Started (6)

Executing in shells

- Computations are expressed on collections distributed across the cluster
 - ✓ Termed as Resilient Distributed Datasets (RDD)
 - ✓ Sparks fundamental abstraction for distributed data and computation
- Example
 - ✓ Create RDD from local text file
 - ✓ Do some very simple analysis on it
 - ✓ Output the result
 - ✓ Exit the shell , Ctrl - D



```
Applications Places Terminal
File Edit View Search Terminal Help
>>> lines = sc.textFile("README.md")
>>> lines.count()
105
>>> lines.first()
u'# Apache Spark'
>>>
```



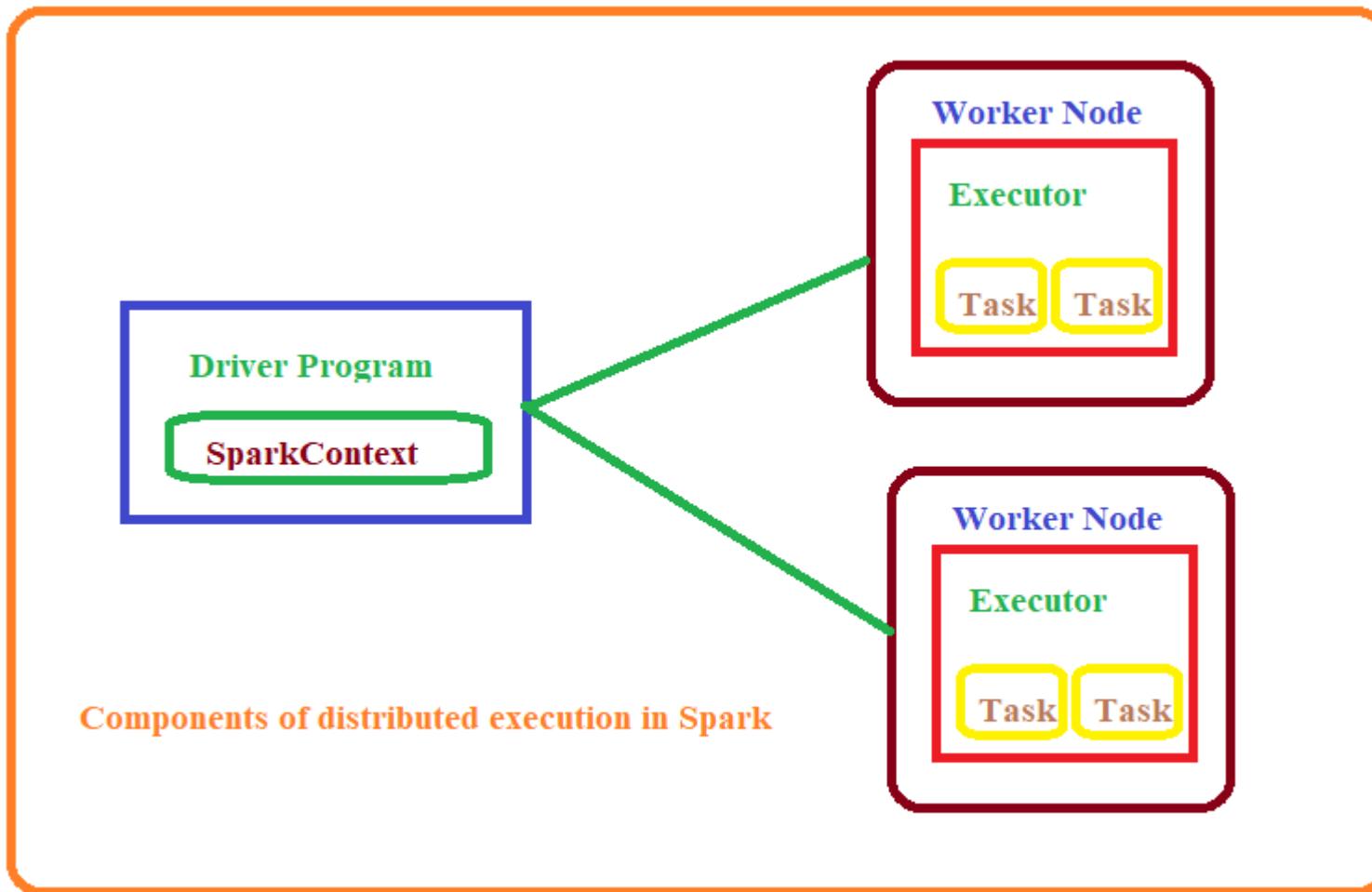
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Standalone Application

Pravin Y Pawar

Core Spark Concepts

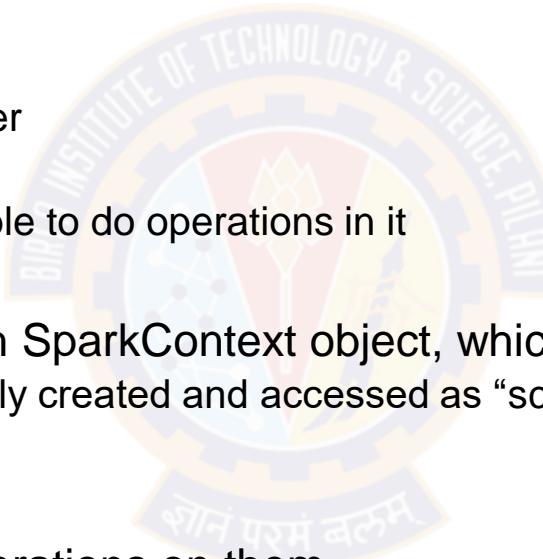
Components of distributed execution



Core Spark Concepts (2)

Driver Program

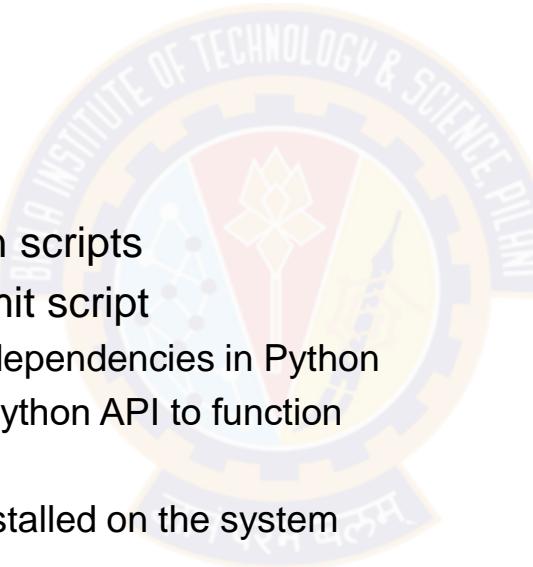
- Every Spark application consists of driver program that launches various parallel operations on a cluster
- Driver program
 - ✓ contains main function
 - ✓ defines distributed datasets on cluster
 - ✓ then applies actions to them
 - Shell is the driver program, hence able to do operations in it
- Driver program access Spark through SparkContext object, which is connection to the cluster
 - ✓ In shell, SparkContext is automatically created and accessed as “sc”
 - ✓ SparkContext is used to build RDDs
- RDDs can be used to run various operations on them
- To run these operations, driver program manage number of nodes called executors
 - ✓ Executors will be present on the different nodes



Standalone Applications

Using Python

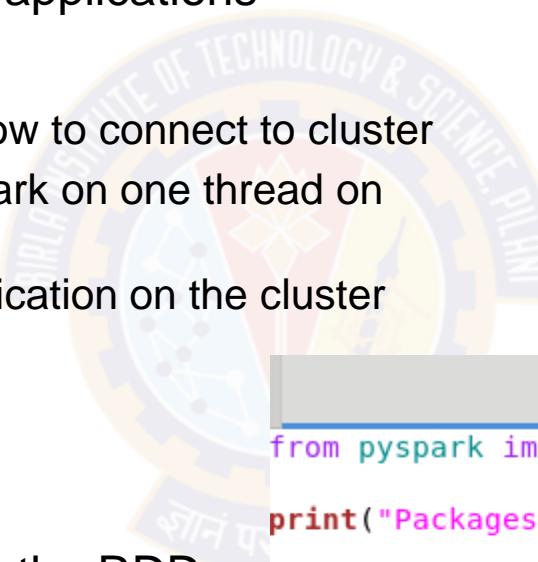
- Spark can be linked into standalone applications in Java, Scala or Python
 - ✓ Need to initialize the SparkContext in program
 - ✓ After that API is same!
- In Python,
 - ✓ Simply write applications as Python scripts
 - ✓ Run them using the bin/spark-submit script
 - ❖ Spark-submit includes the Spark dependencies in Python
 - ❖ Sets up environment for Spark's Python API to function
 - ❖ Need to have pyspark package installed on the system



Standalone Applications(2)

Initializing and Using SparkContext

- Import the packages in program
- Create SparkConf – to configure the applications
- Create SparkContext
 - ✓ A cluster URL, which tells Spark how to connect to cluster
 - ✓ Local is special value that runs Spark on one thread on local machine
 - ✓ Application name – to identify application on the cluster managers UI



- Use methods to create RDDs
- Use APIs to carry out operations on the RDDs
- Call stop() method SparkContext object to stop execution of application

```
my_script.py
from pyspark import SparkConf, SparkContext
print("Packages imported!")
conf = SparkConf().setMaster("local").setAppName("MyScript")
sc = SparkContext(conf = conf)
print('Able to contact Spark!')
lines = sc.textFile("README.md")
print('*****')
print(lines.count())
print('*****')
```

Standalone Applications(3)

Executing program

- Using spark-submit , execute the program

```
File Edit View Search Terminal Help
```

```
[csishydlab@apache-spark spark-2.4.4-bin-hadoop2.7]$ spark-submit my_script.py
```

```
20/04/15 13:31:51 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 0 (PythonRDD[2] at count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12) (first 15 tasks are for partitions Vector(0))
20/04/15 13:31:51 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks
20/04/15 13:31:51 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, executor driver, partition 0, PROCESS_LOCAL, 7917 bytes)
20/04/15 13:31:51 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
20/04/15 13:31:51 INFO HadoopRDD: Input split: file:/home/csishydlab/spark-2.4.4-bin-hadoop2.7/README.md:0+3952
20/04/15 13:31:52 INFO PythonRunner: Times: total = 315, boot = 266, init = 49, finish = 0
20/04/15 13:31:52 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1547 bytes result sent to driver
20/04/15 13:31:52 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 428 ms on localhost (executor driver) (1/1)
20/04/15 13:31:52 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
20/04/15 13:31:52 INFO PythonAccumulatorV2: Connected to AccumulatorServer at host: 127.0.0.1 port: 58581
20/04/15 13:31:52 INFO DAGScheduler: ResultStage 0 (count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12) finished in 0.486 s
20/04/15 13:31:52 INFO DAGScheduler: Job 0 finished: count at /home/csishydlab/spark-2.4.4-bin-hadoop2.7/my_script.py:12, took 0.539315 s
105
*****
```



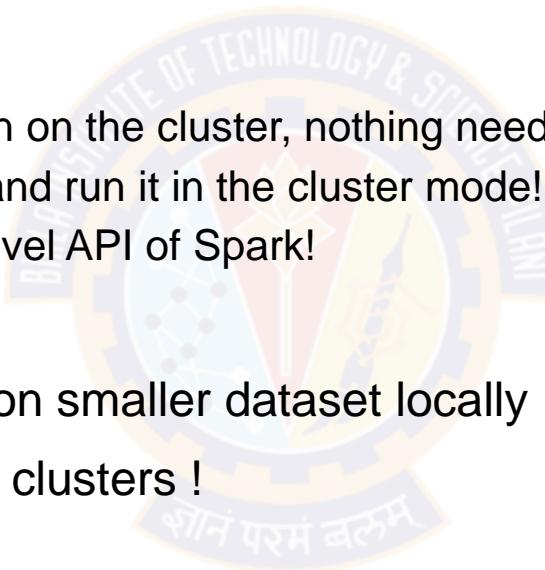
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Runtime Architecture

Pravin Y Pawar

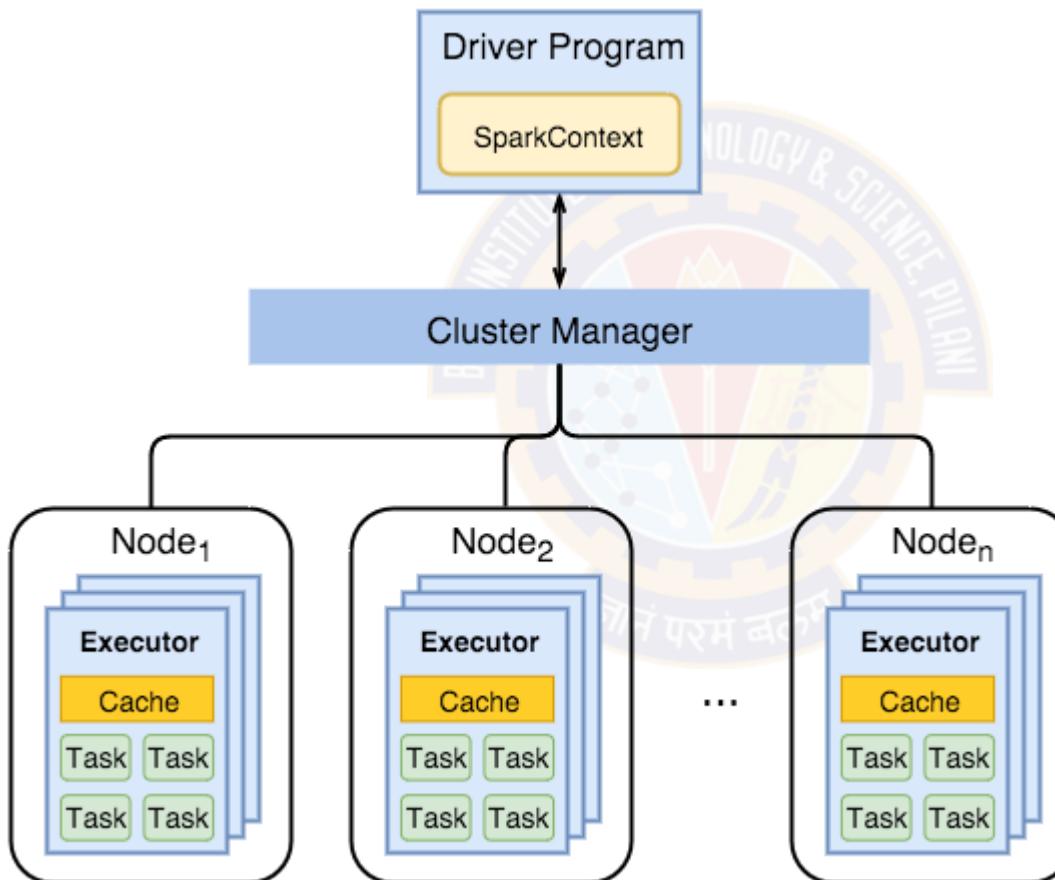
Motivation

- Till now executed applications on the local (standalone) mode
- How to move to the cluster?
 - ✓ If same application needs to be run on the cluster, nothing needs to be changed!
 - ✓ Just need to add more machines and run it in the cluster mode!
 - ✓ All this possible because of high level API of Spark!
- Can rapidly prototype applications on smaller dataset locally
- Run unmodified code on very large clusters !



Spark Runtime Architecture

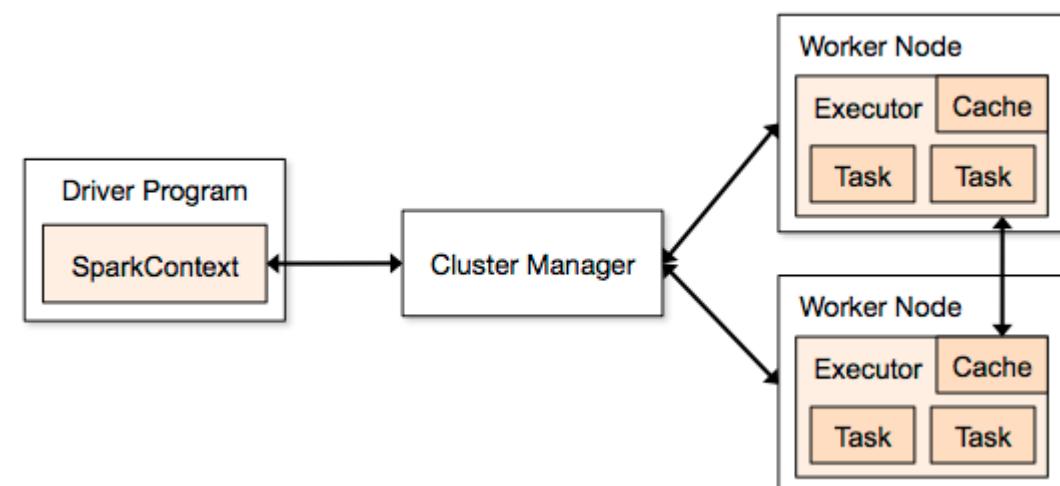
Components of Distributed Spark Application



Source : [Spark intro blog](#)

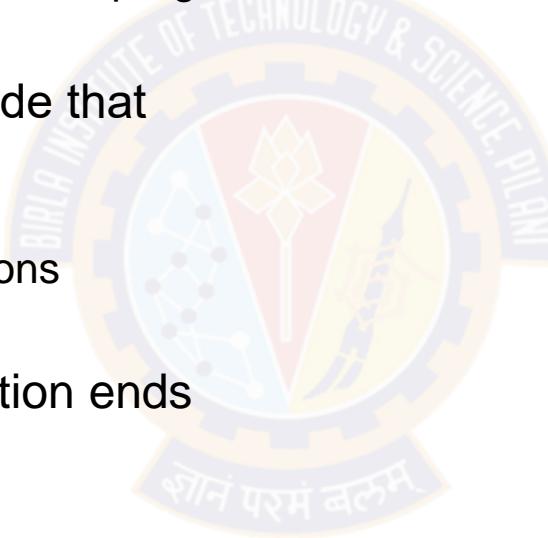
Spark Runtime Architecture (2)

- Uses Master/Slave architecture with one central coordinator and many distributed workers
 - ✓ Central coordinator – driver
 - ✓ Driver communicates with large number of workers – executors
 - ✓ Both runs in their separate Java processes
 - ✓ Driver and Executor together termed as Spark Application!
- Spark application launched on set of machines using external service called cluster manager
 - ✓ Spark has built-in standalone cluster manager
 - ✓ Also supports Hadoop YARN, Apache Mesos as cluster managers



Driver

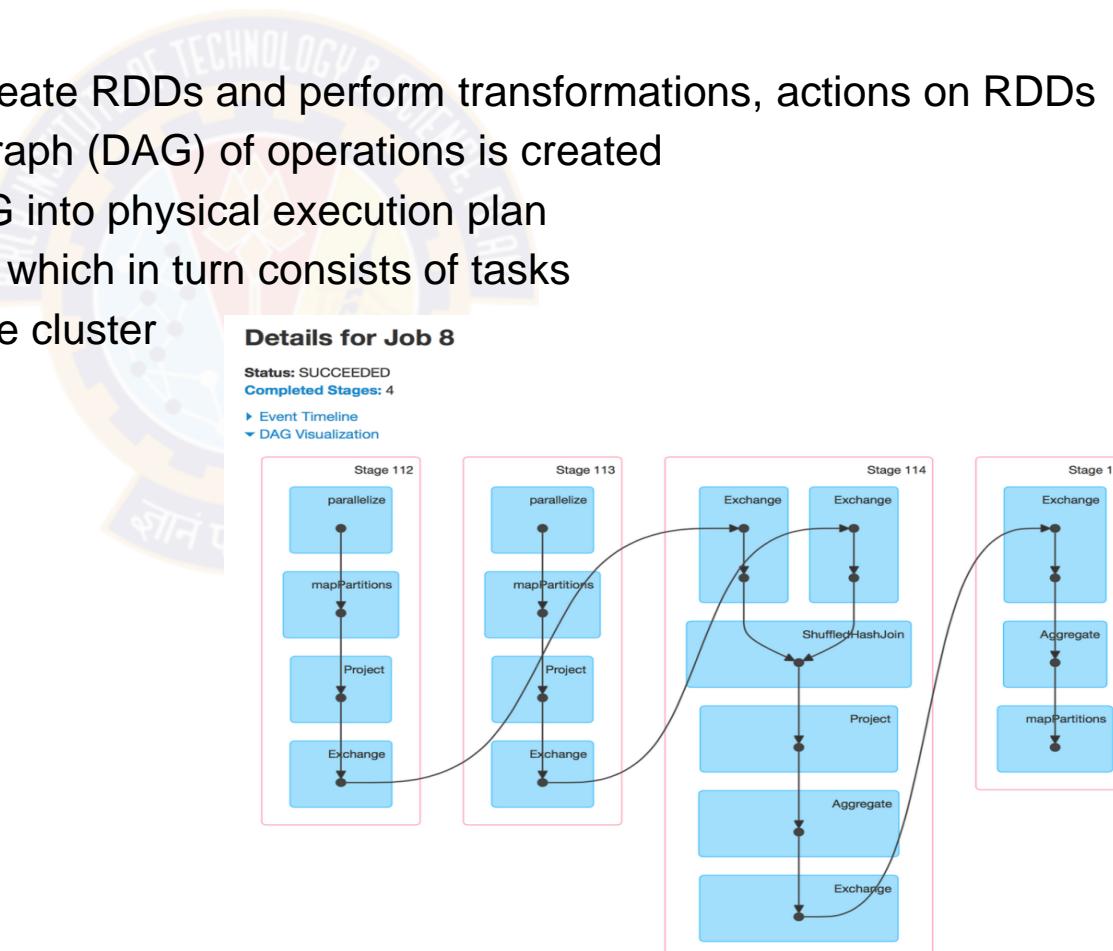
- Driver is the central coordinator for Spark application
 - ✓ Runs in separate Java process
 - ✓ It's the process where main() method of program runs
- It's the process running the user code that
 - ✓ Creates SparkContext
 - ✓ Creates RDDs
 - ✓ Performs transformations and actions
- Once the driver terminates, application ends
- Two responsibilities
 - ✓ Converting user program into tasks
 - ✓ Scheduling tasks on executors



Driver (2)

Two Duties - Converting user program into tasks

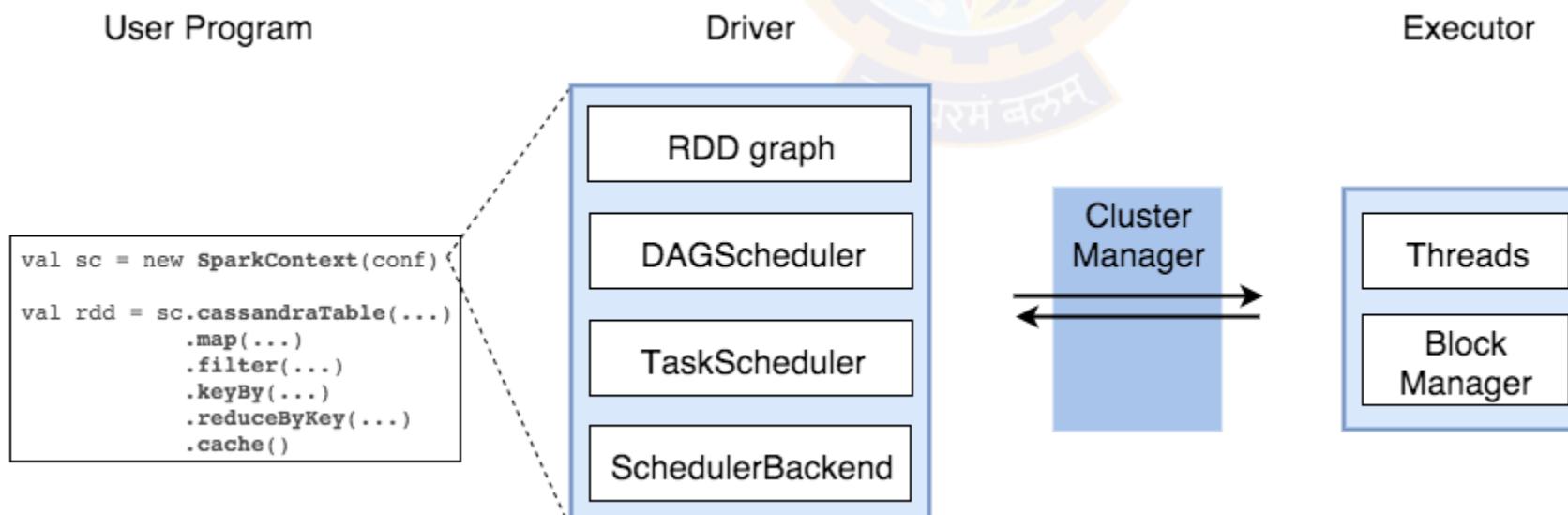
- Converting user program into tasks
 - ✓ Physical execution unit is called as task
 - ✓ Smallest unit of work – task!
 - ✓ Programs creates SparkContext, Create RDDs and perform transformations, actions on RDDs
 - ✓ Implicitly a logical directed acyclic graph (DAG) of operations is created
 - ✓ When driver runs, it converts the DAG into physical execution plan
 - ✓ Converts execution plan into stages which in turn consists of tasks
 - ✓ Tasks are bundled up and sent to the cluster



Driver (3)

Two Duties – Tasks Scheduling

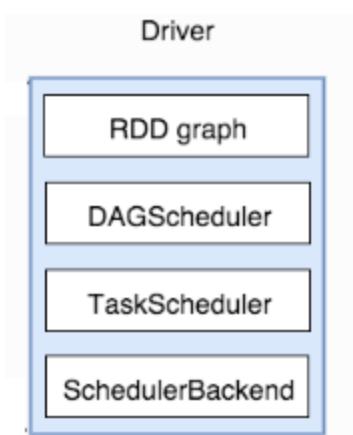
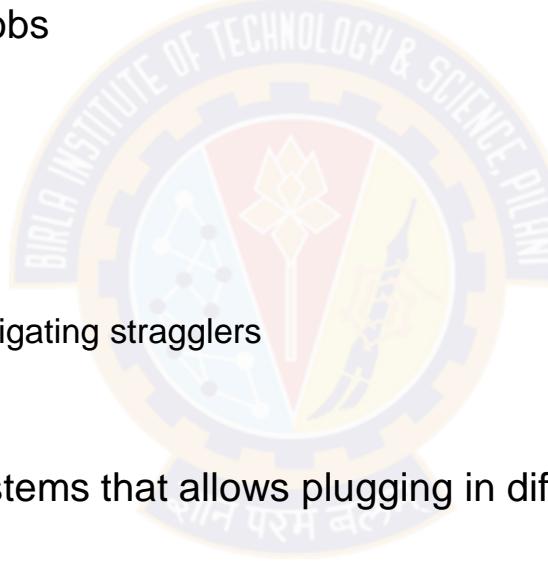
- Scheduling the tasks on executors
 - ✓ With execution plan, Driver needs to coordinate scheduling of tasks on executors
 - ✓ When executor starts, it gets registered with driver, so driver has complete info about them
 - ✓ Executor is process capable of running tasks and storing RDD data
- Based on data placement, Spark will identify the executors to schedule the tasks
 - ✓ When tasks completed on executor, they might have cached data
 - ✓ Driver uses that cached data to schedule future tasks based on that data



Driver (4)

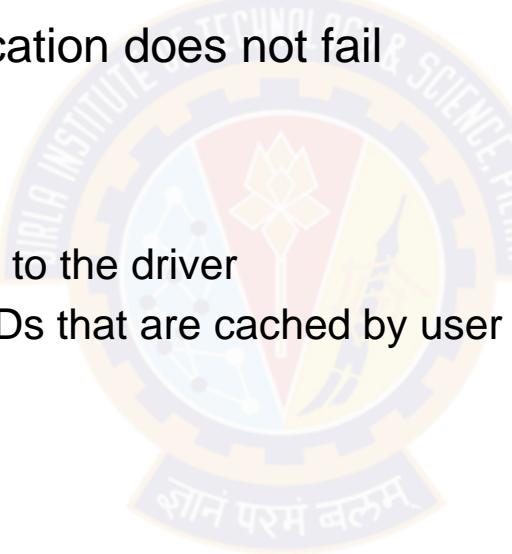
Program Translation into Tasks

- DAGScheduler
 - ✓ computes a DAG of stages for each job and submits them to TaskScheduler
 - ✓ determines preferred locations for tasks (based on cache status or shuffle files locations)
 - ✓ finds minimum schedule to run the jobs
- TaskScheduler
 - ✓ responsible for
 - ✓ sending tasks to the cluster
 - ✓ running them,
 - ✓ retrying if there are failures, and mitigating stragglers
- SchedulerBackend
 - ✓ backend interface for scheduling systems that allows plugging in different implementations (Mesos, YARN, Standalone, local)
- BlockManager
 - ✓ provides interfaces for putting and retrieving blocks both locally and remotely into various stores (memory, disk)

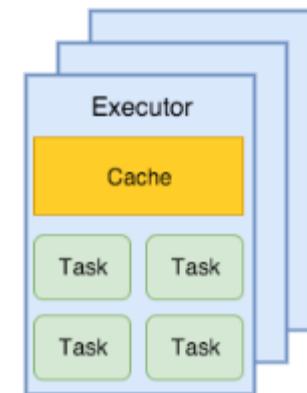


Executors

- Worker processes responsible for running individual tasks in given Spark job
- Launched when application is started and typically run till lifetime of application
- Even if executors fails, Spark application does not fail
- Roles of executors
 - ✓ Run the tasks and return the result to the driver
 - ✓ Provide in-memory storage for RDDs that are cached by user programs



Workers

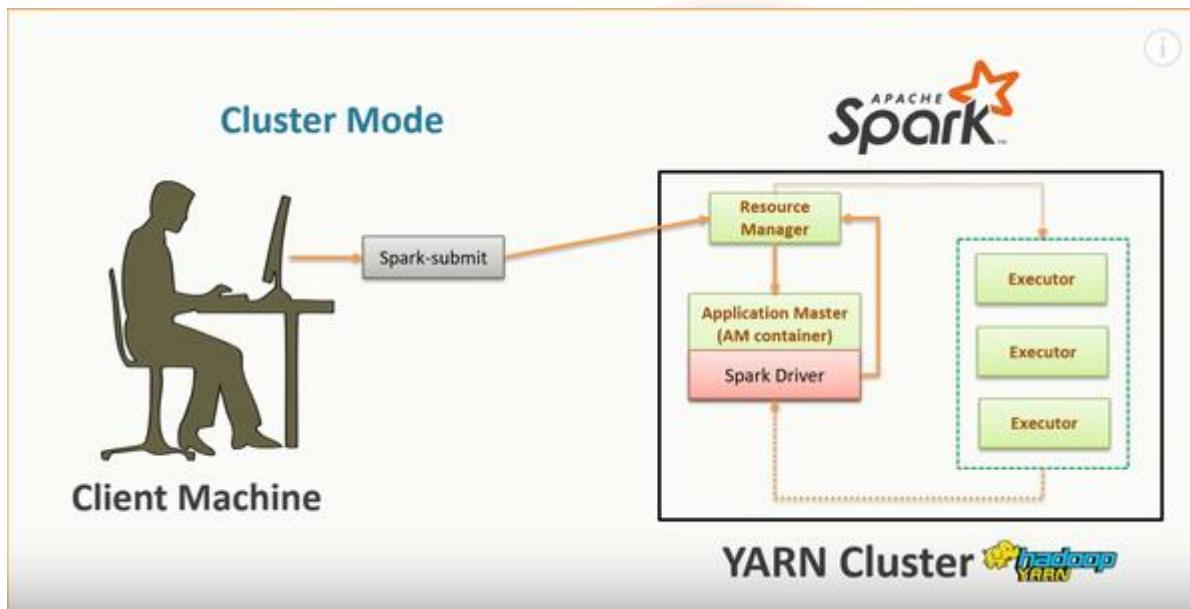


Cluster Manager

- Cluster manager is used to launch the Spark application
- Pluggable component
- Allows sparks to run on top of different external managers like YARN, Mesos as well as standalone cluster manager
- Single script to submit your application to cluster i.e. spark-submit
 - ✓ With different options, it can be used to connect to different clusters
 - ✓ Used to manage the resources allotted to the application



Application execution



Source : [Quora answer](#)

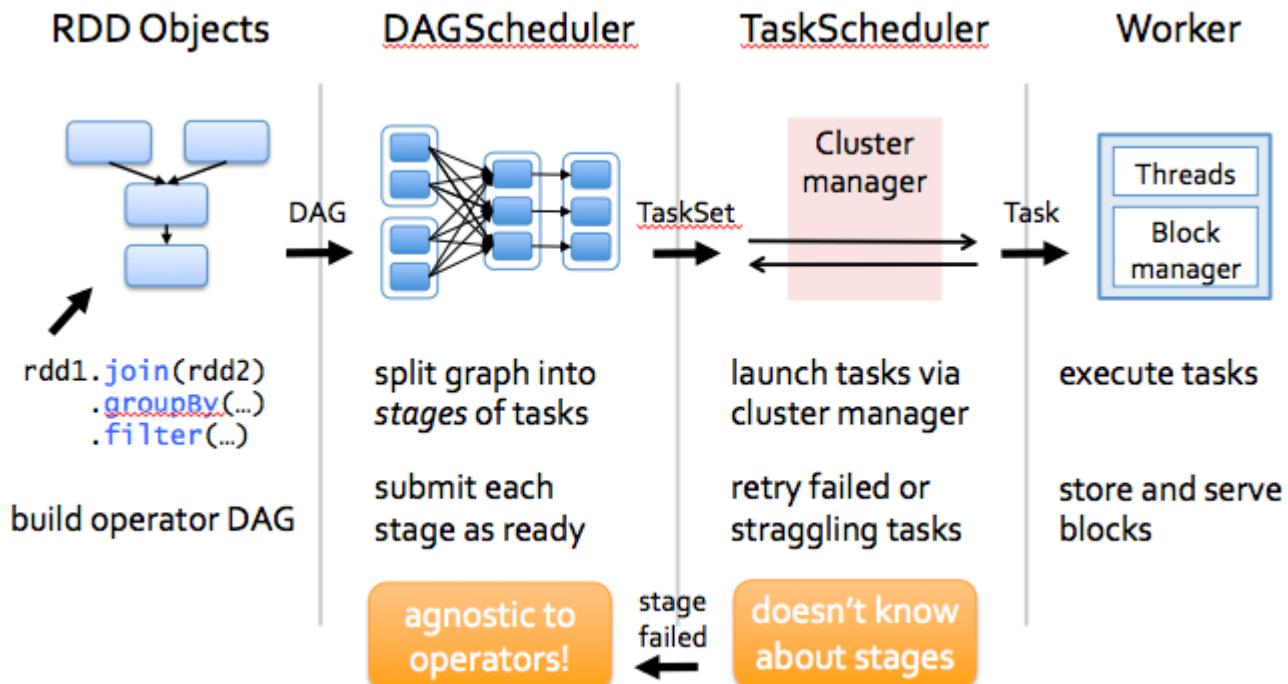
Application execution (2)

Detailed

1. User submits the application using spark-submit
2. Spark-submit launches the driver program and invokes the main() method of program
3. Driver program contacts the cluster manager to ask for resources to launch executors
4. Cluster manager launches the executors on the behalf of driver program
5. Driver process runs through user application
 - Based on RDD transformations and actions in the program, driver sends work to executors in form of tasks
6. Tasks are run on executor processes to compute and save results
7. If the driver's main method exits or calls SparkContext.stop(), it will terminate all executors and release resources from the cluster manager

Application execution (3)

Application flow



Source : [Spark Intro blog](#)



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Application Submission

Pravin Y Pawar

Cluster concepts

Application	User program built on Spark. Consists of a <i>driver program</i> and executors on the cluster.
Application jar	A jar containing the user's Spark application. In some cases users will want to create an "uber jar" containing their application along with its dependencies. The user's jar should never include Hadoop or Spark libraries, however, these will be added at runtime.
Driver program	The process running the <code>main()</code> function of the application and creating the <code>SparkContext</code>
Cluster manager	An external service for acquiring resources on the cluster (e.g. standalone manager, Mesos, YARN)
Deploy mode	Distinguishes where the driver process runs. In "cluster" mode, the framework launches the driver <i>inside</i> of the cluster. In "client" mode, the submitter launches the driver <i>outside</i> of the cluster.
Worker node	Any node that can run application code in the cluster
Executor	A process launched for an application on a worker node, that runs tasks and keeps data in memory or disk storage across them. Each application has its own executors.
Task	A unit of work that will be sent to one executor
Job	A parallel computation consisting of multiple tasks that gets spawned in response to a Spark action (e.g. <code>save</code> , <code>collect</code>); you'll see this term used in the driver's logs.
Stage	Each job gets divided into smaller sets of tasks called stages that depend on each other (similar to the map and reduce stages in MapReduce); you'll see this term used in the driver's logs.

Submitting Applications

Spark-submit

- Used to launch applications on a cluster
- Can use all of Spark's supported cluster managers through a uniform interface
 - ✓ don't have to configure application especially for each one
- Bundling Application's Dependencies
 - If code depends on other projects, need to package them alongside application in order to distribute the code to a Spark cluster
 - ✓ Need to create an assembly jar (or "uber" jar) containing code and its dependencies
 - ✓ Both sbt and Maven have assembly plugins
 - ✓ Spark and Hadoop as provided dependencies – provided at runtime
 - ✓ Call the bin/spark-submit script while passing jar
 - For Python,
 - ✓ Can use the --py-files argument of spark-submit to add .py, .zip files to be distributed with application
 - ✓ If depend on multiple Python files recommend packaging them into a .zip or .egg.

Launching Applications with spark-submit

- Once a user application is bundled, it can be launched using the bin/spark-submit script
- This script takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports

```
./bin/spark-submit \
--class <main-class> \
--master <master-url> \
--deploy-mode <deploy-mode> \
--conf <key>=<value> \
... # other options
<application-jar> \
[application-arguments]
```

Launching Applications with spark-submit (2)

Common options

- **--class:**
 - ✓ The entry point for application (e.g. org.apache.spark.examples.SparkPi)
- **--master:**
 - ✓ The master URL for the cluster (e.g. spark://23.195.26.187:7077)
- **--deploy-mode:**
 - ✓ Whether to deploy your driver on the worker nodes (cluster) or locally as an external client (client) (default: client)
- **--conf:**
 - ✓ Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap "key=value" in quotes
- **application-jar:**
 - ✓ Path to a bundled jar including application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an hdfs:// path or a file:// path that is present on all nodes.
- **application-arguments:**
 - ✓ Arguments passed to the main method of your main class, if any

Launching Applications with spark-submit (3)

On Standalone cluster

```
# Run application locally on 8 cores
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master local[8] \
  /path/to/examples.jar \
  100

# Run on a Spark standalone cluster in client deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000

# Run on a Spark standalone cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

Launching Applications with spark-submit (4)

On YARN cluster

```
# Run on a YARN cluster
export HADOOP_CONF_DIR=XXX
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master yarn \
  --deploy-mode cluster \ # can be client for client mode
  --executor-memory 20G \
  --num-executors 50 \
  /path/to/examples.jar \
  1000
```

Launching Applications with spark-submit (4)

On Mesos and Kubernetes cluster

```
# Run on a Mesos cluster in cluster deploy mode with supervise
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master mesos://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  http://path/to/examples.jar \
  1000

# Run on a Kubernetes cluster in cluster deploy mode
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master k8s://xx.yy.zz.ww:443 \
  --deploy-mode cluster \
  --executor-memory 20G \
  --num-executors 50 \
  http://path/to/examples.jar \
  1000
```

Launching Applications with spark-submit (5)

Submitting Python Code

```
# Run a Python application on a Spark standalone cluster
./bin/spark-submit \
--master spark://207.184.161.138:7077 \
examples/src/main/python/pi.py \
1000
```



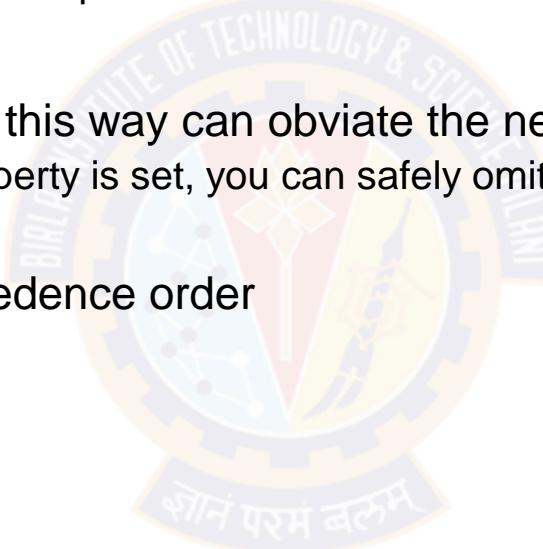
Master URLs

Master URL	Meaning
local	Run Spark locally with one worker thread (i.e. no parallelism at all).
local[K]	Run Spark locally with K worker threads (ideally, set this to the number of cores on your machine).
local[*]	Run Spark locally with as many worker threads as logical cores on your machine.
spark://HOST:PORT	Connect to the given Spark standalone cluster master. The port must be whichever one your master is configured to use, which is 7077 by default.
spark://HOST1:PORT1,HOST2:PORT2	Connect to the given Spark standalone cluster with standby masters with Zookeeper . The list must have all the master hosts in the high availability cluster set up with Zookeeper. The port must be whichever each master is configured to use, which is 7077 by default.
mesos://HOST:PORT	Connect to the given Mesos cluster. The port must be whichever one your is configured to use, which is 5050 by default. Or, for a Mesos cluster using ZooKeeper, use <code>mesos://zk://....</code> To submit with <code>--deploy-mode cluster</code> , the HOST:PORT should be configured to connect to the MesosClusterDispatcher .
yarn	Connect to a YARN cluster in <code>client</code> or <code>cluster</code> mode depending on the value of <code>--deploy-mode</code> . The cluster location will be found based on the <code>HADOOP_CONF_DIR</code> or <code>YARN_CONF_DIR</code> variable.

Loading Configuration from a File

Configuration file

- The spark-submit script can load default Spark configuration values from a properties file and pass them on to application
 - ✓ By default, it will read options from conf/spark-defaults.conf in the Spark directory
- Loading default Spark configurations this way can obviate the need for certain flags to spark-submit.
 - ✓ For instance, if the spark.master property is set, you can safely omit the --master flag from spark-submit.
- In general, configuration values precedence order
 - ✓ Directly set values on a SparkConf
 - ✓ then flags passed to spark-submit
 - ✓ then values in the defaults file.
- If you are ever unclear where configuration options are coming from, you can print out fine-grained debugging information by running spark-submit with the --verbose option.





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Cluster Managers

Pravin Y Pawar

Scheduling Spark Applications

- Many clusters are shared between multiple users/programs
 - ✓ Challenge is how to schedule the jobs?
 - ✓ What happens if two users launch applications that each want to use entire clusters resources?
- For scheduling in multitenant clusters, Spark relies on cluster managers to share resources between spark applications
- When a application asks for executors from the cluster manager, it may receive more or fewer resources depending upon the availability and contention in clusters
- Variety of cluster managers
 - ✓ Standalone
 - ✓ Hadoop YARN
 - ✓ Apache Mesos

Standalone Cluster Manager

Submitting applications

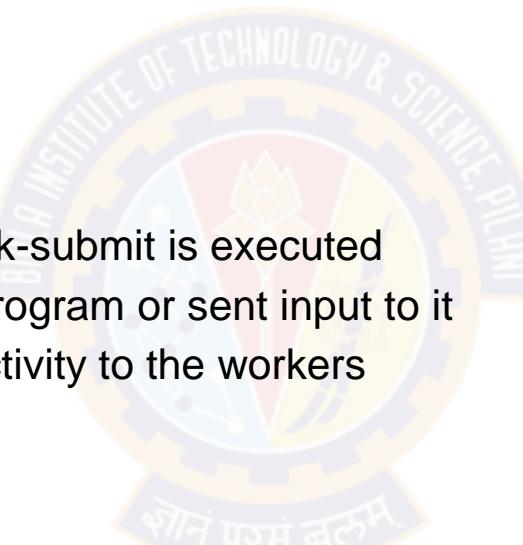
- Simple way to run applications on cluster
- Consists of master and multiple workers , each with configured amount of memory and CPU cores
- When app is submitted, user specifies
 - ✓ how much memory its executors will use
 - ✓ Total number of cores across all executors
- Submitting applications
 - ✓ Need to set master argument as sparkL//masternode:7077
 - ✓ Web UI can be accessed at same cluster URL



Standalone Cluster Manager (2)

Deployment modes

- Two deploy modes – where driver program of application runs
 - ✓ Client mode (default)
 - ✓ Cluster mode
- Client mode
 - ✓ Driver runs on machine where spark-submit is executed
 - ✓ Directly can see the output of the program or sent input to it
 - ✓ Machine needs to have fast connectivity to the workers
- Cluster mode
 - ✓ Driver is launched within the cluster, as another process on one of worker nodes
 - ✓ Spark-submit is “fire and forget” – close the laptop still applications runs!
 - ✓ Will be able to access logs through cluster managers Web UI



Standalone Cluster Manager (3)

Resource configurations

- Needs to decide how to allocate resources between the executors
- Managed through two settings
- Executor memory
 - ✓ Configured using –executor-memory argument to spark-submit (default : 1GB)
 - ✓ Each application will have at most one executor on each worker node
 - ✓ This setting controls how much of that workers memory the application will claim
- Maximum number of total cores
 - ✓ Total number of cores used across all executors for an application (default : unlimited)
 - ✓ Application will launch executors on every available node of cluster
 - ✓ Set through –total-executor-cores argument to spark-submit

Standalone Cluster Manager (4)

High Availability

- Cluster needs to be available even if the worker nodes fails
- Gracefully support the failure of worker nodes
- If the master also needs to be highly available, then zookeeper can be used to keep multiple masters on standby and switch to a new one when any of them fails



Hadoop YARN

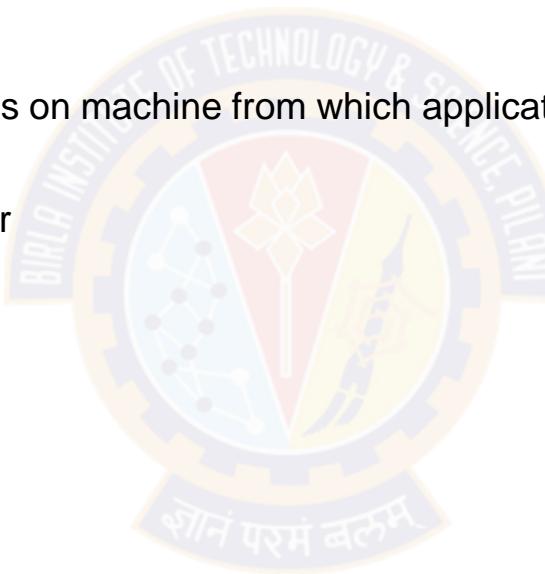
Submitting applications

- Introduced in Hadoop 2.0
- Typically installed on the same nodes as the HDFS
- Lets Spark access HDFS data quickly, on the same nodes where the data is stored
- Using it straightforward:
- Set environment variable pointing to Hadoop configuration directory
 - ✓ HADDDOP_CONF_DIR
 - ✓ Directory that contains yarn-site.xml and other config files
 - ✓ Usually HADOOP_HOME/conf
- Submit jobs to a special master URL with spark-submit
 - spark-submit –master yarn my_app

Hadoop YARN (2)

Deployment modes

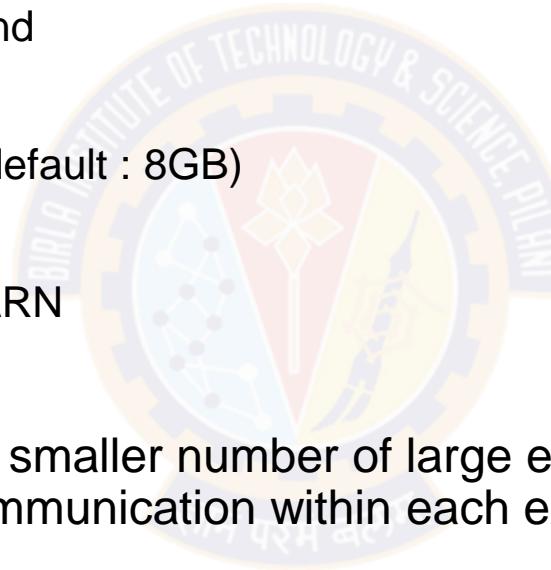
- Set the mode through –deploy-mode argument to spark-submit
- Two deployment modes
 - ❖ Client mode
 - ✓ Driver program for application runs on machine from which application is submitted
 - ❖ Cluster mode
 - ✓ Driver runs inside YARN container



Hadoop YARN (3)

Resource configurations

- --num-executors
 - ✓ Spark applications use a fixed number of executors (default:2)
 - ✓ Set through spark-submit command
- --executor-memory
 - ✓ Memory used by each executor (default : 8GB)
- --executor-cores
 - ✓ Number of cores it claims from YARN
- Usually Spark will run better with a smaller number of large executors (with more memory and cores) since it can optimize the communication within each executor



Apache Mesos

Submitting applications

- General purpose cluster manager – can run analytics workloads and long running services
- Need to pass mesos URI with spark-submit
 - ✓ Spark-submit –master mesos://masternode:5050 my_app
- Zookeeper can be used to elect a master when running on multimaster node
- Connecting Spark to Mesos
 - ✓ To use Mesos from Spark, you need a Spark binary package available in a place accessible by Mesos, and a Spark driver program configured to connect to Mesos.
 - ✓ Alternatively, you can also install Spark in the same location in all the Mesos slaves, and configure spark.mesos.executor.home (defaults to SPARK_HOME) to point to that location.

Apache Mesos (2)

Scheduling modes

- Spark can run over Mesos in two modes: “coarse-grained” mode
- In “coarse-grained” mode
 - ✓ Each Spark executor runs as a single Mesos task.
 - ✓ Spark executors are sized according to the following configuration variables:
 - Executor memory: `spark.executor.memory`
 - Executor cores: `spark.executor.cores`
 - Number of executors: `spark.cores.max/spark.executor.cores`
- Executors are brought up eagerly when the application starts, until `spark.cores.max` is reached
 - ✓ If you don't set `spark.cores.max`, the Spark application will consume all resources offered to it by Mesos
- The benefit of coarse-grained mode is much lower startup overhead, but at the cost of reserving Mesos resources for the complete duration of the application.

Apache Mesos (3)

Scheduling modes

- Spark can run over Mesos in two modes: “fine-grained” mode
- In “fine-grained” mode,
 - ✓ Each Spark task inside the Spark executor runs as a separate Mesos task
 - ✓ This allows multiple instances of Spark (and other frameworks) to share cores at a very fine granularity, where each application gets more or fewer cores as it ramps up and down, but it comes with an additional overhead in launching each task
 - ✓ This mode may be inappropriate for low-latency requirements like interactive queries or serving web requests
 - ✓ It is deprecated now!

Apache Mesos (4)

Deployment modes

- Set the mode through –deploy-mode argument to spark-submit
- Two deployment modes
 - ❖ Client mode
 - ✓ Driver program for application runs on machine from which application is submitted
 - ❖ Cluster mode
 - ✓ Driver runs inside Mesos Cluster



Which one to use?

Guidelines to choose a cluster manager

- If this is new deployment
 - ✓ Start with standalone cluster as its easy to setup
 - ✓ Provides almost all the features as the other cluster managers
- If needs to run Spark alongside with other applications or to use richer resource scheduling capabilities
 - ✓ Both YARN and Mesos provide these features
 - ✓ YARN will be preinstalled in Hadoop distributions
 - ✓ Mesos offers more fine grained resource management
- In all cases,
 - ✓ Best to run Spark on the same nodes as HDFS for fast access to storage
 - ✓ Hadoop distribution installs YARN and HDFS together
 - ✓ Mesos and Standalone cluster managers can be installed manually on the same nodes



Thank You!

In our next session: Apache Flink



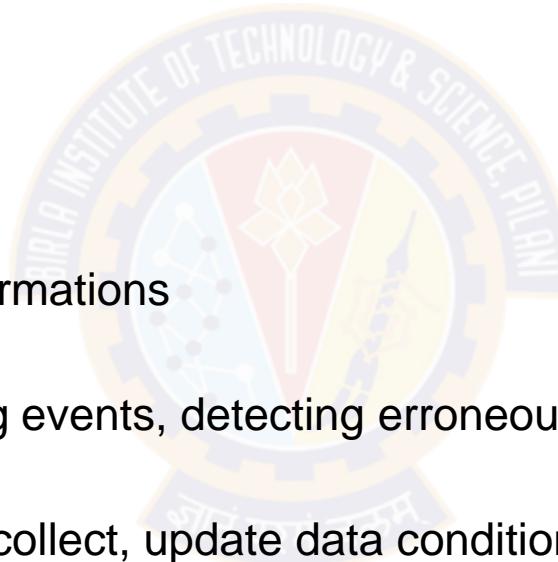
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Necessity of Streaming SQL

Pravin Y Pawar

SQL Usecases

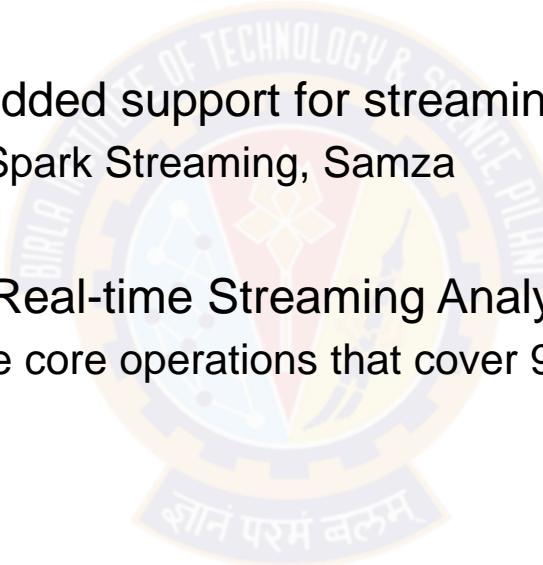
- Traditionally SQL is used a lot for data analysis
 - ✓ Main reason easy to understand , easy to code
- Streaming Use cases
 - ✓ Simple counting
 - ✓ Counting with Windows
 - ✓ Pre-processing: filtering, transformations
 - ✓ Alerts , thresholds
 - ✓ Data Correlation, Detect missing events, detecting erroneous data
 - ✓ Joining event stream
 - ✓ Merge with data in a database, collect, update data conditionally
 - ✓ Detecting Event Sequence Patterns
 - ✓ Tracking - follow some related entity's state in space, time etc.
 - ✓ Detect trends – Rise, turn, fall, Outliers, Complex trends like triple bottom etc.



Reference : <https://www.kdnuggets.com/2015/03/sql-query-language-realtime-streaming-analytics.html>

Why can't use SQL with streaming data?

- Why we can not think of using SQL with streaming data?
 - ✓ The main difference is element of time
- Almost all stream processors has added support for streaming SQL around 2015
 - ✓ like Storm, Flink, Kafka Streams, Spark Streaming, Samza
- Need SQL like query language for Real-time Streaming Analytics
 - ✓ to be expressive, short, fast, define core operations that cover 90% of problems
 - ✓ to be easy to follow and learn



Why use SQL with streaming data?

- List of reasons for SQL like query language
 - ✓ Real-time analytics are hard. Every developer do not want to hand implement sliding windows and temporal event patterns, etc.
 - ✓ Easy to follow and learn for people who knows SQL, which is pretty much everybody
 - ✓ SQL like languages are Expressive, short, sweet and fast!!
 - ✓ SQL like languages define core operations that covers 90% of problems
 - ✓ Real-time analytics runtime can better optimize the executions with SQL like model.

Streaming SQL Operators

- Two kinds of operators
- First one is same as the SQL operators used in databases or in batch processing
 - ✓ Projection, which lets us control what data goes into the output
 - ✓ Filters, which let us only select some of the data
 - ✓ Group by, which lets us calculate aggregates for groups
 - ✓ Having, which lets us filter aggregated data
- The second class includes special operators that
 - ✓ extends the power of SQL to handle streaming-specific use cases
 - ✓ supports the concept of time
 - ✓ like window, join and patterns



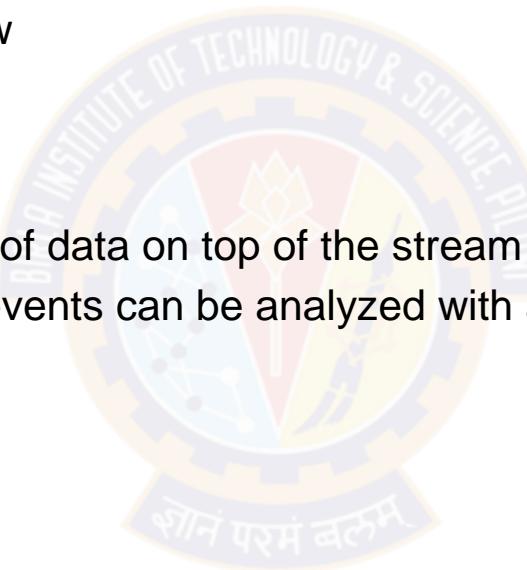
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Streaming SQL : Windows

Pravin Y Pawar

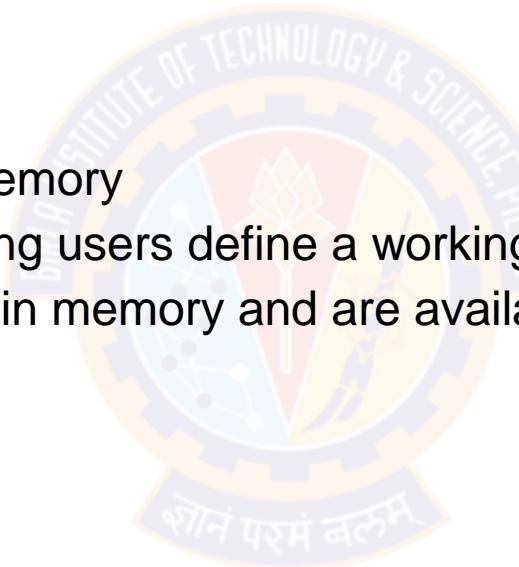
Window

- Assume that we want to calculate the moving average of the temperature reading from a stream
 - ✓ Operation used to understand the data over time
 - ✓ Not about calculating the value now
- Use Window operator
 - ✓ that captures the idea of a window of data on top of the stream
 - ✓ Once grouped under the window, events can be analyzed with aggregate functions, such as average, count, etc. in classic SQL.



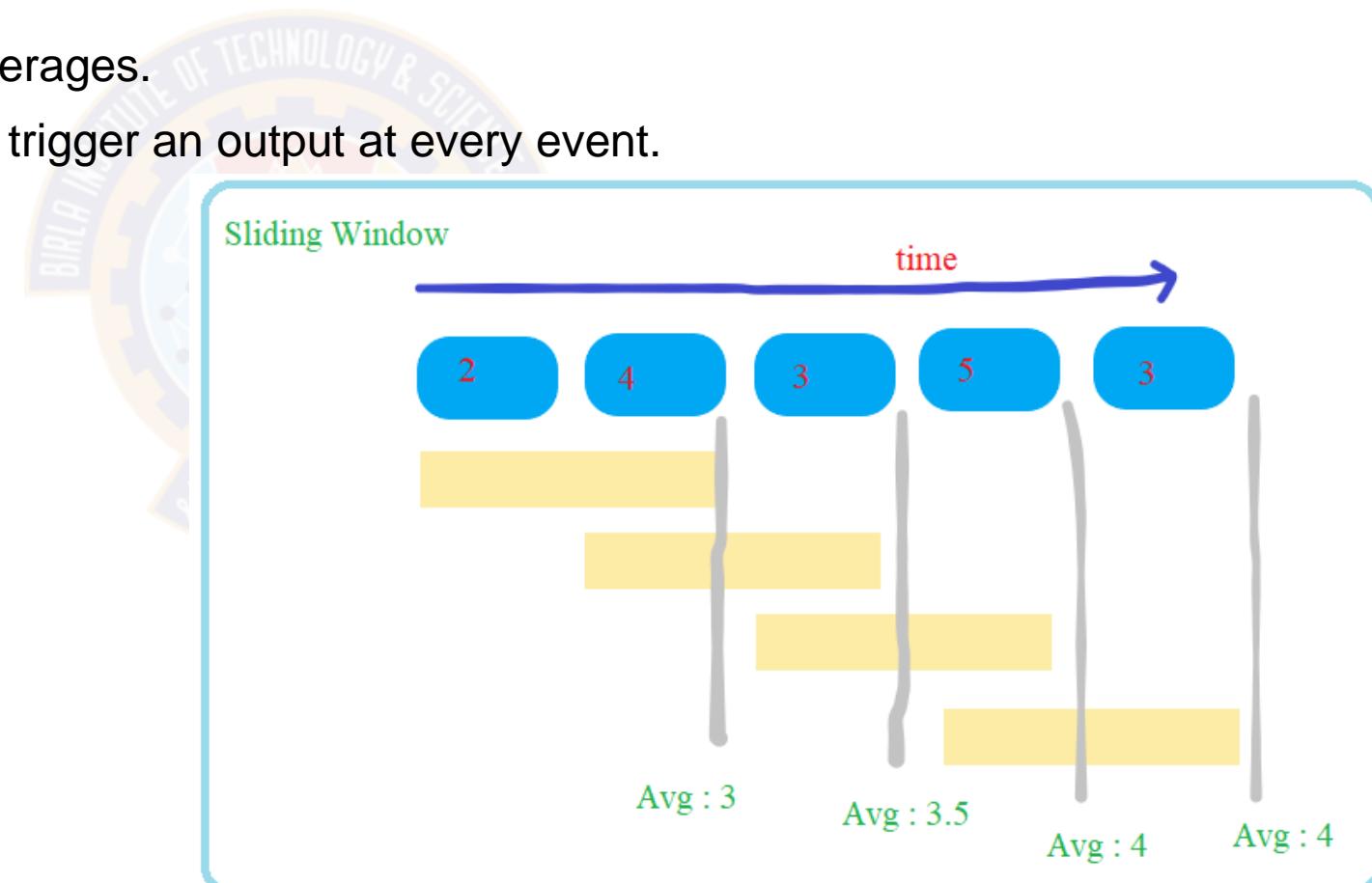
Window(2)

- Usual cases considers filters and projection
 - ✓ looks only at a single event at a time
- Most real-world use cases
 - ✓ Need some form of working memory
 - ✓ Streaming handles this by letting users define a working memory as a window.
 - ✓ Events in the window are kept in memory and are available for further processing with operators



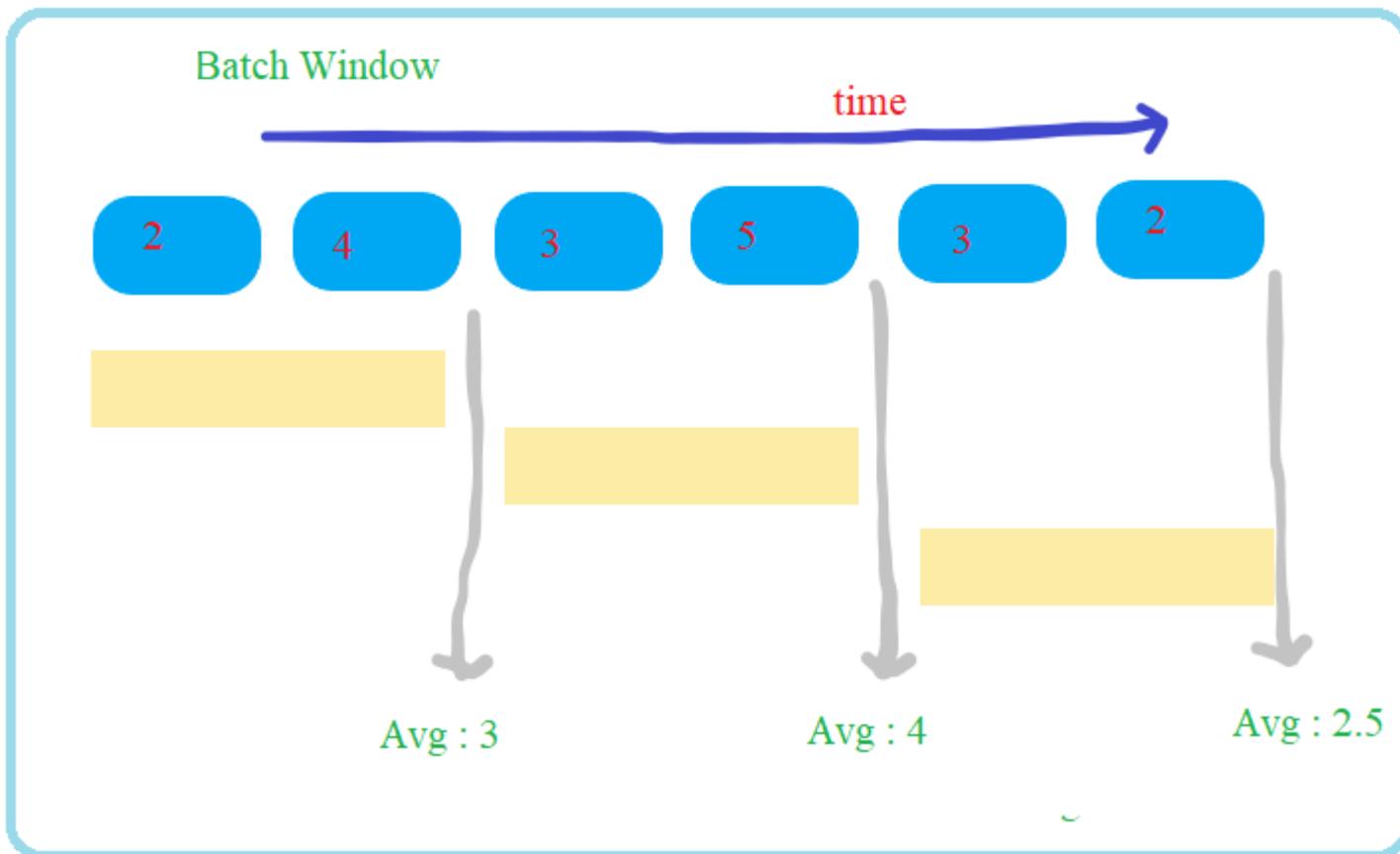
Sliding length window

- The most simple available window
- Keeps last N events in the window
- Often used to calculate moving averages.
- sliding means that the window will trigger an output at every event.



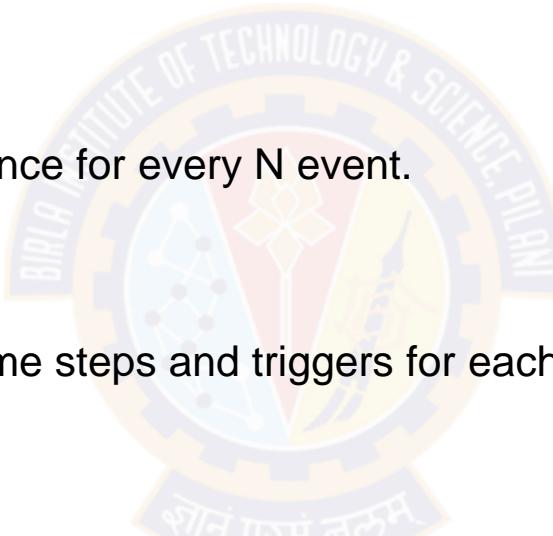
Batch Window

- where the window will only trigger an output at the end of the batch



Types of Windows

- Sliding length window
 - ✓ keeps last N events and triggers for each new event.
- Batch length window
 - ✓ keeps last N events and triggers once for every N event.
- Sliding time window
 - ✓ keeps events triggered at last N time steps and triggers for each new event.
- Batch time window
 - ✓ keeps events triggered at last N time steps and triggers once for the time period at the end.



Example

- KafkaSQL

```
Select boiler_id, avg(T) as T
```

```
From BoilerStream
```

```
WINDOW HOPPING (SIZE 10, ADVANCE BY 1)
```





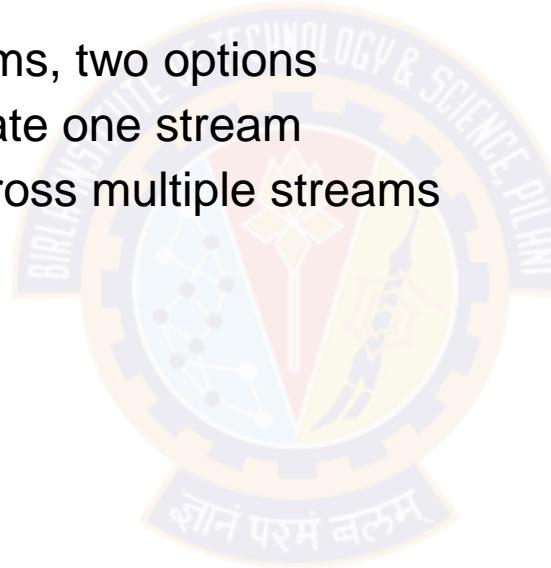
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Streaming SQL : Joins

Pravin Y Pawar

Streaming SQL :Joins

- To handle data from multiple tables, we use the JOIN operator in SQL
- To handle data from multiple streams, two options
 - ✓ First is to join the two and create one stream
 - ✓ Second is to write patterns across multiple streams

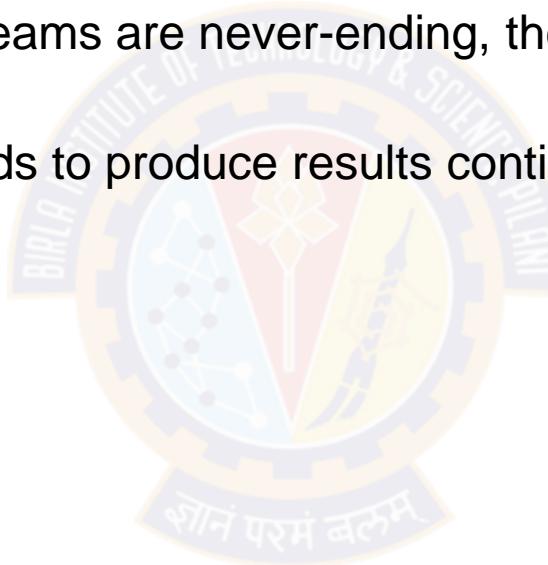


Reference : <https://wso2.com/library/articles/2018/02/stream-processing-101-from-sql-to-streaming-sql-in-ten-minutes/>

Streaming SQL :Joins(2)

Challenges

- First challenge of joining data from multiple streams is that they need to be aligned as they come because they will have different time stamps
- Second challenge is that, since streams are never-ending, the joins must be limited; otherwise the join will never end
- Third challenge is that the join needs to produce results continuously as there is no end to the data



Streaming SQL :Joins(4)

Example

- Stream S1 has attributes “x” and “id”
- Stream S2 has attributes “y” and “id”
- Lets join the two streams based on ID attributes

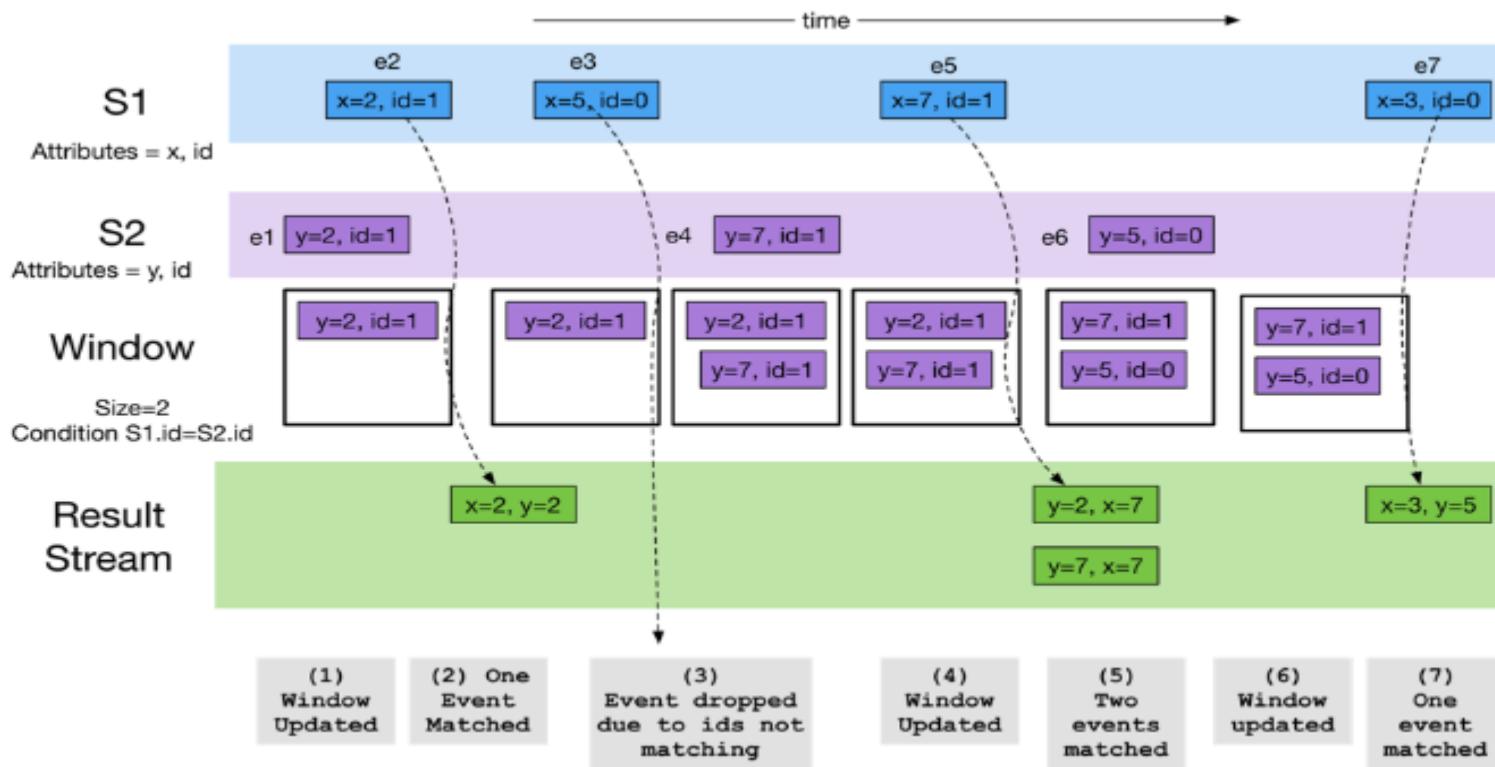
Select x, y

```
From S1 as s1 join S2 as s2  
on s1.id=s2.id  
insert into JoinedStream
```



Streaming SQL :Joins(5)

Example (continued...)



Streaming SQL :Joins(6)

Example (continued...)

1. When e1 happens, it is collected in the window as it occurs in stream S2.
2. When e2 happens, the event is joined with e1 and produces a new event.
3. Event e3, which occurs in the active stream, is dropped as it does not match the ID with any events in the window.
4. Event e4 is retained in the window because it is in stream S2.
5. Event e5 matches with events in the window and generates two events as output.
6. Event e6 is retained in the window, and the oldest event e1 is removed as the window is of size 2.
7. Event e7 matches with events in the window and produces two events as output.

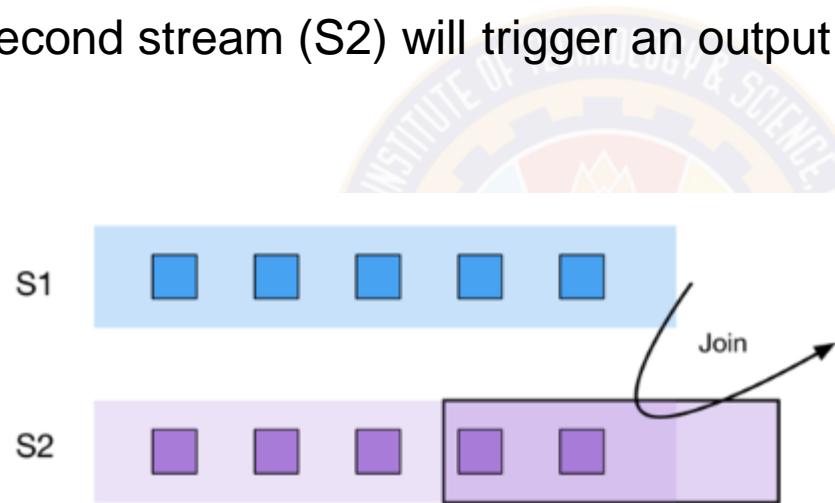
Types of Join

- At least one stream has to be bounded or in other words must have a window attached
- Two cases :
 - One window join
 - Two window join



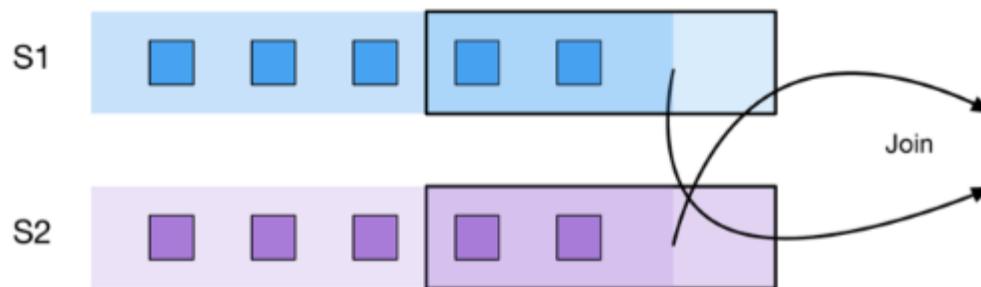
One Window Join

- One stream (S2) is bounded by a window, and events retained in the window are matched against events coming in the second stream (S1)
- Only events in the second stream (S2) will trigger an output



Two Window Join

- A two-window query will retain events coming from both the streams in the window
- A new event coming in will either trigger a match and an output





BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Approaches to Stream Processing

Pravin Y Pawar

Approaches to Stream Processing

Two approaches

- Important techniques to stream processing
- Bulk-synchronous processing
- One-at-at-time processing



Microbatching

Application of Bulk-Synchronous Processing

- Gist of BSP includes:
- A split distribution of asynchronous work
 - Is an idea that each of the successive steps of work to be done in streaming is separated in a number of parallel chunks
 - Roughly proportional to number of executors available to perform the task
 - Each executor receives the chunk(s) of work and works separately until the second element comes in
 - A particular resource is tasked with keeping track of progress of computation
 - Between these scheduled steps, all executors on the cluster are doing the same thing
- A synchronous barrier, coming in at fixed intervals
 - Frequency at which further rounds of processing are scheduled id dictated by time period
 - Implemented at small, fixed intervals that better approximate the real time motion of data processing
- Function-passing style – asynchronously pass safe functions to data
 - Functions are passed around the scheduling process that describe processing to be done on data
 - Data is already on various executors, delivered directly to resources
- Spark Streaming follows this style

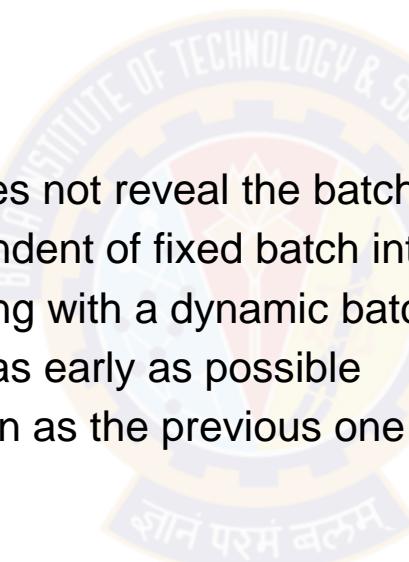
One-Record-at-a-Time Processing

Pipelining the steps

- Works based on pipelining
 - Analyses the whole computation as described by user-specified functions and deploys it as a pipeline using the resources of the cluster
 - Flow the data through various resources, following prescribed pipeline
 - Each step of computation is materialized at some place in cluster at any given point
- Apache Flink, Storm and IBM Streams follows this style
- Latency
 - For microbatching system is batch interval + processing time
 - For one-record-at-a-time system is only processing time as it reacts as it meets the event of interest

Bringing Microbatch and One-record-at-time Together

- Marriage is implemented in systems like Flink and Naiad
 - Still subject of research!
- Spark Structured Streaming
 - Backed up by microbatching but does not reveal the batch interval at API level
 - Allows for processing that is independent of fixed batch interval
 - Execution model mixes microbatching with a dynamic batch interval
 - Trigger the execution of next batch as early as possible
 - New batch should be started as soon as the previous one has been processed



The Trade-Offs

Microbatching vs One-at-a-time

- Despite high latency, microbatching offers significant advantages :
- Able to adapt at the synchronization barrier boundaries
 - Might represent the task of recovering from failure
 - Give opportunity to add or remove executor nodes, possibility to grow or shrink resources depending upon cluster load
- Have easier time providing strong consistency
 - Batch determinations – beginning and end of batch of data – are deterministic and recorded
 - Any kind of computation can be redone and produce same results the second time
- Perform efficient optimizations
 - Data available as a set can provide ideas on the way to compute on data
 - Allows an efficient way of specifying programming both batch processing and streaming data
 - Even for mere instances, looks like data at rest



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

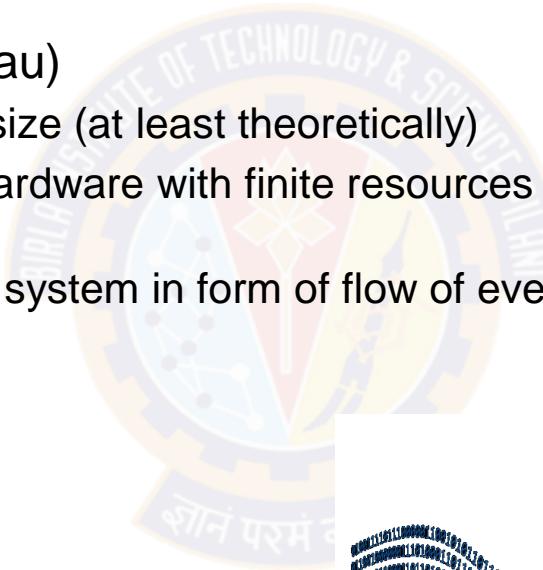
Distributed Stream Processing with Apache Spark

Pravin Y Pawar

Stream Processing

Unbounded data

- The discipline and related set of techniques used to extract information from unbounded data
 - Unbounded data (as per Tyler Akidau)
 - A type of dataset that is infinite in size (at least theoretically)
 - Information systems are built on hardware with finite resources like memory and storage, they cannot hold unbounded datasets
 - Data is received at the processing system in form of flow of events over time - as streams of data



dzone

Stream Processing (2)

Factors

- Processing
 - Concerned with processing of data as it arrives to the system
 - Stream processors runs constantly for as long as the stream is delivering the new data
 - Theoretically , forever
- “Time”
 - Stream processing program assumes input is potentially infinite in size and observed over time
 - Data at rest is data from past – initially was stream of data collected over time into storage
 - Data in motion needs to take time difference into consideration
 - Event time and processing time
 - Becomes more important when need to correlate, order or aggregate event wrt another
- Uncertainty
 - No assumptions are made on the throughput at which events are received
 - Needs to match up with sudden arrival of input with computing resources necessary to process
 - If not planned, might face delays, resource constriction, of failure
 - Dealing with this uncertainty in future in critical aspect of streaming processing

Distributed Stream Processing

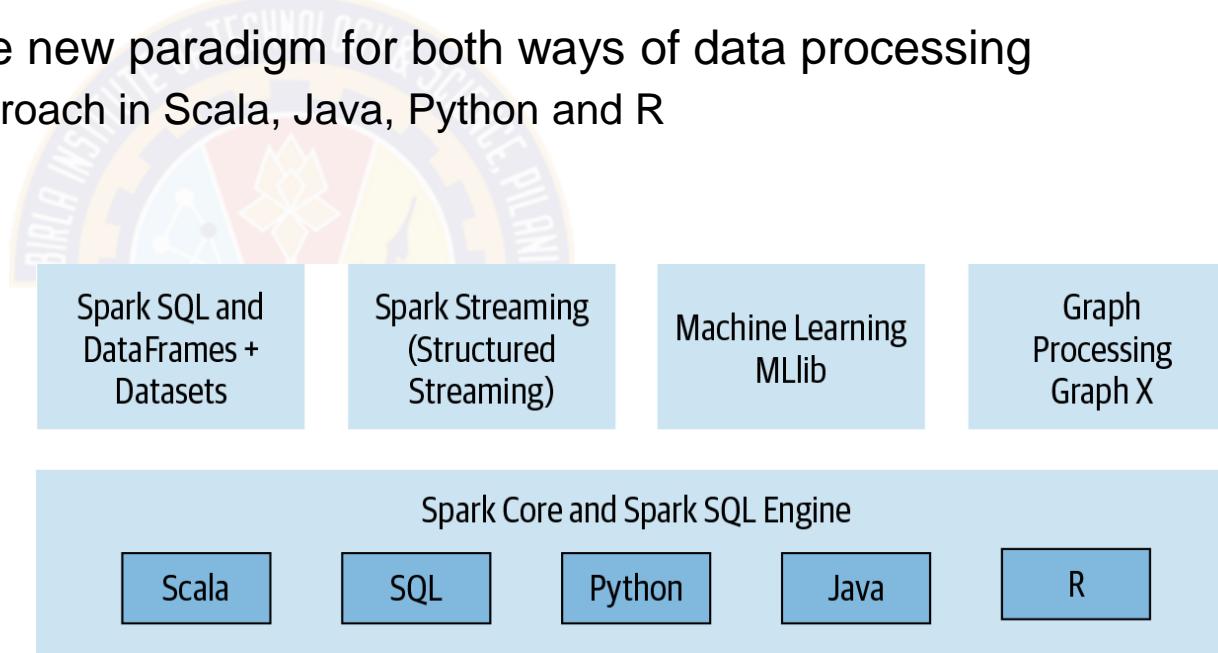
Necessity

- Distributed processing
 - Batch processing has access to complete dataset , with streams only small portion of dataset is available at any time
 - Becomes more aggravated in distributed system, input stream is further divided into partitions and resides on different nodes in system
 - **Needs to provide an abstraction that hides complexity from user!**
- Stateful Stream Processing
 - Poses additional burdens on the distributed stream processing system
 - Need to ensure that
 - state is preserved over the time
 - data consistency guarantees, even in case of partial system failures

Apache Spark as a Solution

A unified engine for data processing

- Fast, reliable and fault-tolerant distributed computing framework for large scale data processing
- A unified analytics engine offering both batch and streaming capabilities
- Developers needs to learn only one new paradigm for both ways of data processing
 - With compatible polyglot APIs approach in Scala, Java, Python and R



oreilly

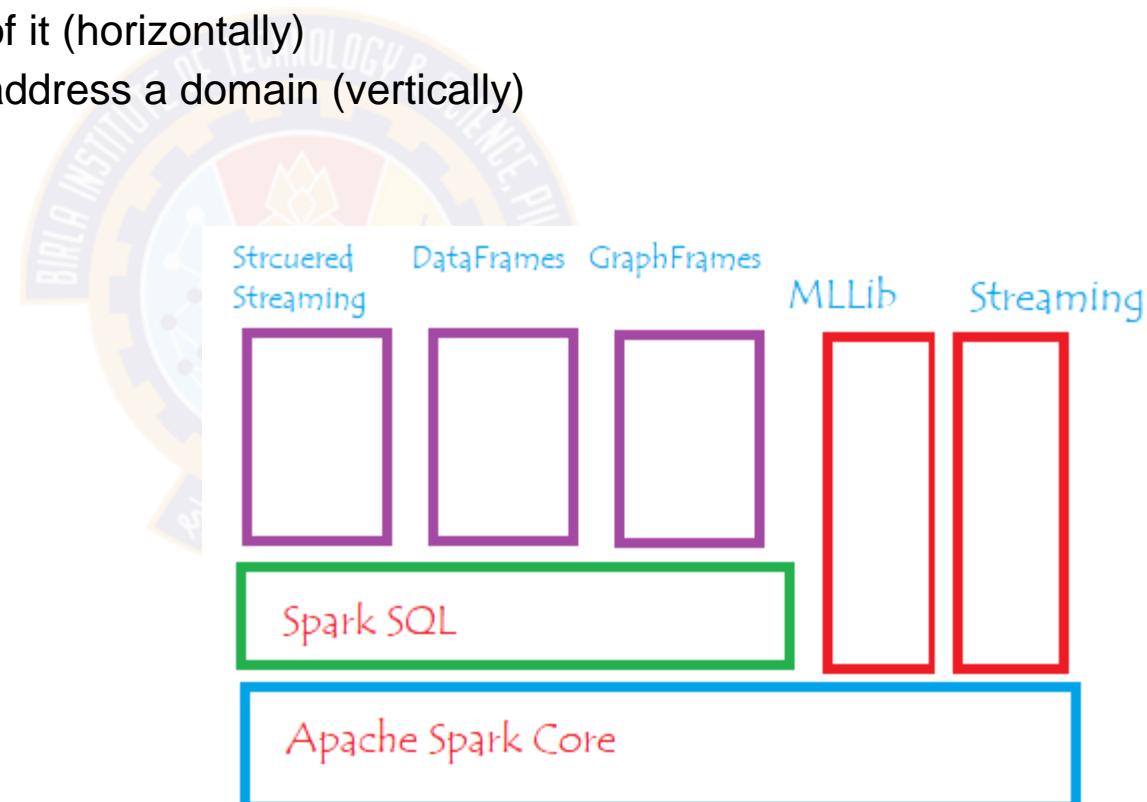
Evolution of Spark

Two waves

- First Wave : Functional APIs
 - Early days, known for novel use of memory and functional APIs
 - Use of in-memory model for data processing resulted into faster data processing
 - Core abstraction Resilient Distributed Dataset (RDD) brought a rich functional programming model abstracting the complexities of distributed data processing
 - Transformations and actions
- Second Wave : SQL
 - Second game changer was introduction of Spark SQL , DataFrames (and Datasets) APIs
 - Adds SQL support to any dataset that has schema
 - Possible to query CSV, Parquet or JSON dataset similar to a SQL database
 - Lowered barriers for adaption – data scientists, business analysts can use it easily

Spark Components

- Consists of
 - core engine
 - a set of abstractions built on top of it (horizontally)
 - libraries that use abstractions to address a domain (vertically)



Tale of Two APIs

Spark Streaming and Structured Streaming

- Spark Streaming
 - First streaming engine based on distributed capabilities of Spark
 - Spark 0.7.0 release in Feb 2013
 - Based on simple but powerful premise
 - Apply Sparks distributed computing capabilities to stream processing by transforming continuous streams of data into discrete data collections
 - **Micro batching!**
 - Uses same functional programming paradigm as Spark core but introduces a new abstraction
 - Discretized streams – exposes a programming model to operate on data in stream
- Structured Streaming
 - Stream processor built on top of Spark SQL abstractions
 - Extends Dataset and DataFrame APIs with streaming capabilities
 - Adopts schema oriented transformation model – structured part in name
 - Inherits optimizations implemented in Spark SQL
 - Introduced in 2017 with Spark 2.2 release
 - **Still evolving fast with each new version of Spark!**



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Streaming Application

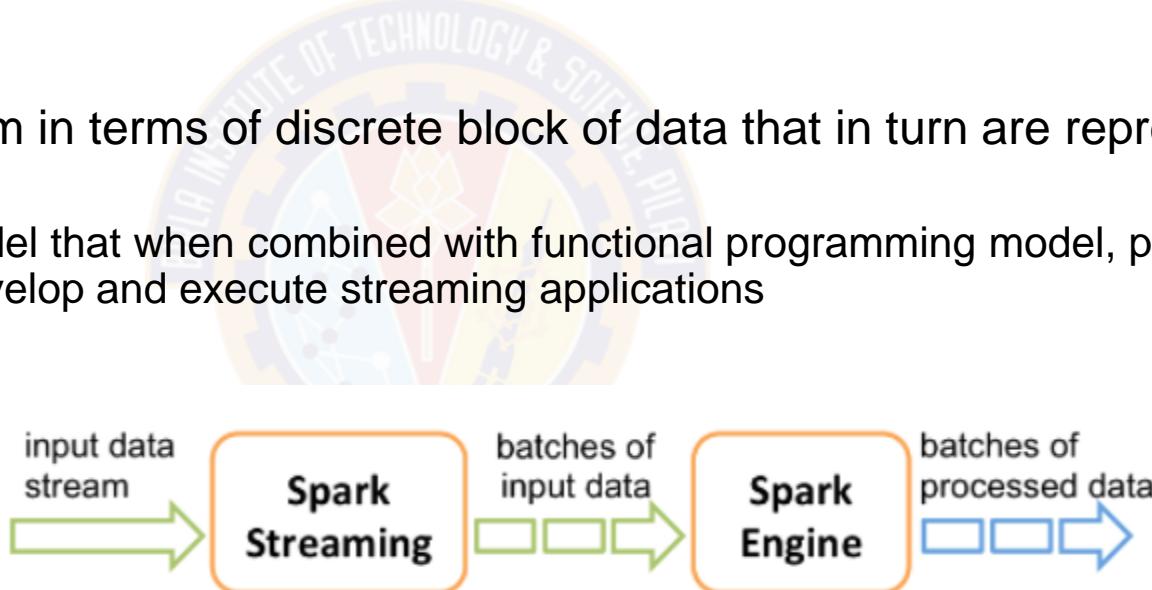
Pravin Y Pawar

Introducing Spark Streaming

- First stream processing framework built on top of Spark's distributed processing capabilities
 - Widely adapted in the industry for large scale data streams
- Spark RDD abstraction
 - Permits creation of programs that treat distributed data as a collection
 - Allows applying data processing logic in form of transformation of distributed dataset
- Spark Streaming created upon simple yet powerful premise
 - apply Spark's distributed computing capabilities to stream processing by transforming a continuous stream of data into discrete data collection on which Spark could operate
- Main task is to take data from stream, package it down into small batches and provide them to Spark for further processing

The DStream Abstraction

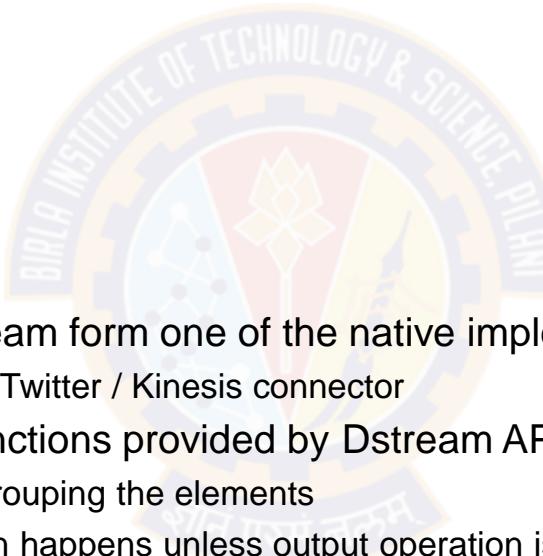
- Spark streaming relies on the much more fundamental Spark abstraction of RDD
- Introduces a new concept : The Discretized Stream or DStream
- DStream represents a stream in terms of discrete block of data that in turn are represented as RDDs over time
 - Primarily an execution model that when combined with functional programming model, provides a complete framework to develop and execute streaming applications



DStreams as a Programming model

Code representation

- Provides functional programming APIs consistent with RDD APIs and augmented with stream specific functions to deal with
 - Aggregations
 - Time based operations
 - Stateful computations
- In Spark Streaming, we
 - Consume a stream by creating Dstream form one of the native implementations or the connectors available
 - From SocketInputStream or Kafka / Twitter / Kinesis connector
 - Implement application logic using functions provided by Dstream API
 - Such as counting the elements or grouping the elements
 - Uses transformations – no execution happens unless output operation is not called
 - Execute output operations to yield out the results
- DStream programming model consists of functional composition of transformations over the stream payload, materialized by one or more output operations and recurrently executed by Spark engine



DStreams as an Execution model

Batch interval

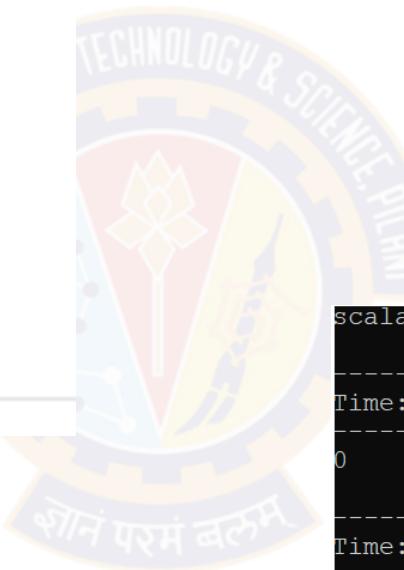
- In programming model, it's described how data is transformed from original form to intended result as a series of lazy evaluations
- Spark streaming engine is responsible for taking that chain of transformations and turning it into an actual execution plan
 - Happens by receiving data from input stream(s), collecting that into batches and feeding to spark in timely manner
- The measure of time to wait for data in **batch interval**
 - Central unit of time
 - Short amount of time, ranging from 200ms to 1 min, depending app's latency requirement
- At each batch interval, the data corresponding to the previous interval is sent to Spark for processing while new data is received
 - Process is repeated as long as Streaming job is active and healthy
- **DStream model dictates that a continuous stream of data is discretized into micro batches using a regular time interval**

The structure of Spark Streaming Application

- Any Spark streaming application needs to do following four things:
 - Create a Spark Streaming Context
 - Define one or more DStreams from data sources or other Dstreams
 - Define one or more output operations to materialize the results of these DStream operations
 - Start the Spark Streaming Context to get the Stream Processing going on
- Points to note
 - Behaviour of job is defined in the operations between the moment that the instance of Streaming context is defined and the moment it is started
 - Manipulation of the context defines the scaffolding for the streaming application – behaviour and execution for the duration of application
 - During the definition phase, all the Dstreams and transformations will be defined and behaviour of Spark streaming application will be wired
 - Once the Spark Streaming context is started, no new Dstreams can be added or it nor can any existing DStream can be structurally modified

Example

```
1 import org.apache.spark.streaming._  
2 val ssc = new StreamingContext(sc, Seconds(2))  
3 val dstream = ssc.socketTextStream("localhost", 8088)  
4 val countStream = dstream.count()  
5 countStream.print()  
6 ssc.start()
```



```
scala> ssc.start()  
Time: 1629967088000 ms  
0  
Time: 1629967090000 ms  
0  
Time: 1629967092000 ms  
0  
Time: 1629967094000 ms  
0
```

Reference

Stream Processing with Apache Spark

Mastering Structured Streaming and Spark Streaming

By Gerard Maas and Francois Garillot



Thank You!

In our next session:

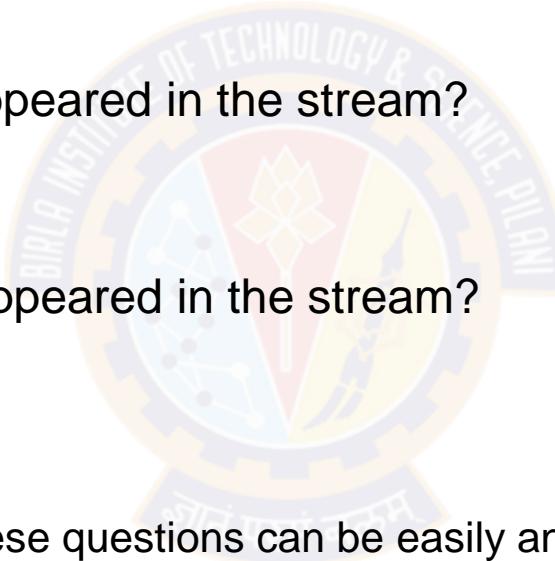


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Sketching Algorithm Fundamentals

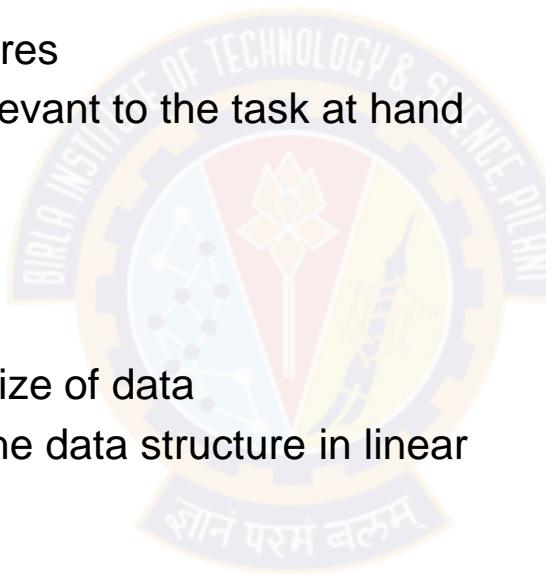
Pravin Y Pawar

Usual questions

- Has this element appeared in stream earlier?
 - ✓ Set membership
 - How many unique elements has appeared in the stream?
 - ✓ Cardinality estimation
 - How many time this element has appeared in the stream?
 - ✓ Frequency
- 
- ✓ If the cardinality of data is less, these questions can be easily answered
 - ✓ But when it starts increasing, need to find out alternative approaches

Sketch algorithms

- Developed to approximate the answers to the usual questions
- Informally
 - ✓ Algorithms + compact data structures
 - ✓ maintains the summary of data relevant to the task at hand
- Features
 - ✓ Constant time updates for data
 - ✓ Storage space is independent of size of data
 - ✓ Worst time required for querying the data structure in linear
- Downside
 - ✓ Errors are introduced in result



Sketch algorithms(2)

- Determining
- Set membership
 - ✓ Bloom Filter
- Cardinality estimation
 - ✓ HyperLogLog
- Frequency
 - ✓ Count-min sketch





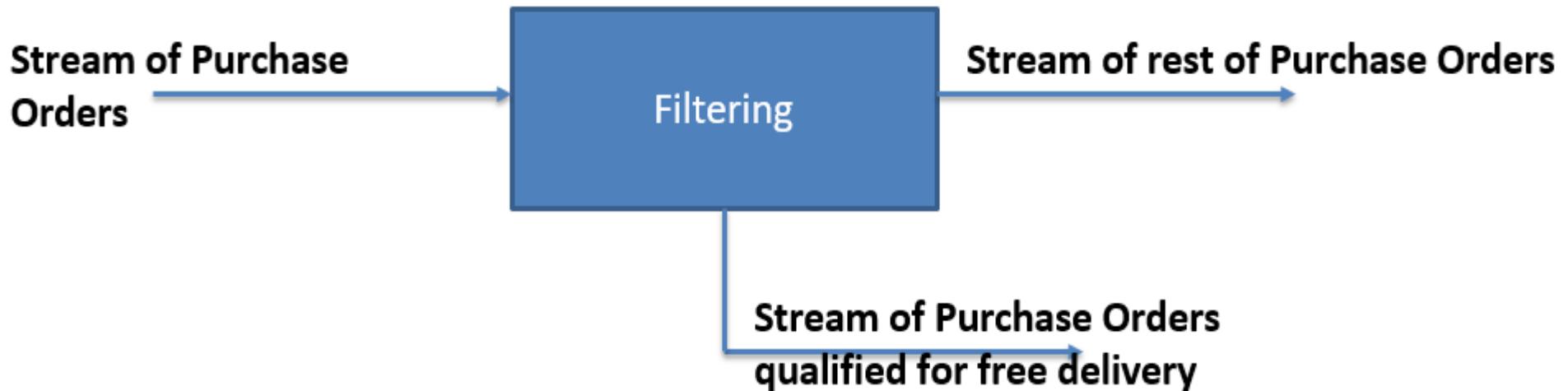
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

The Bloom Filter

Pravin Y Pawar

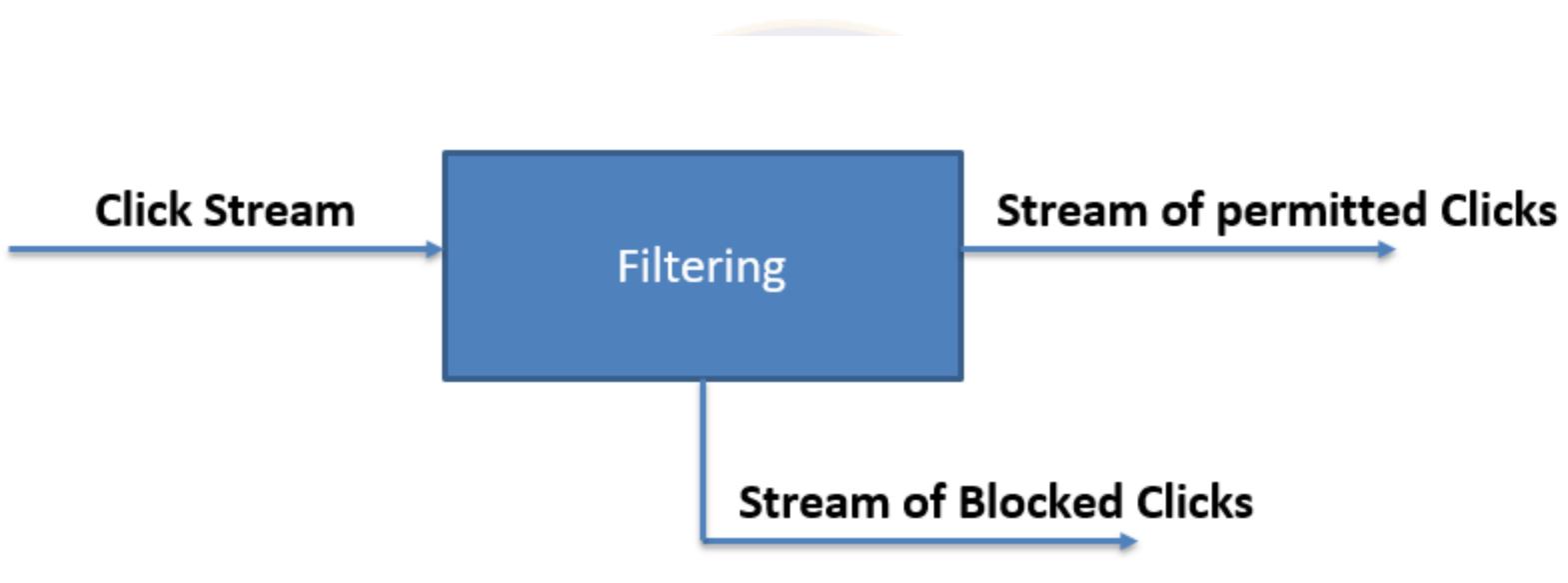
Filtering in a Stream

- Refers to selection of elements in a stream



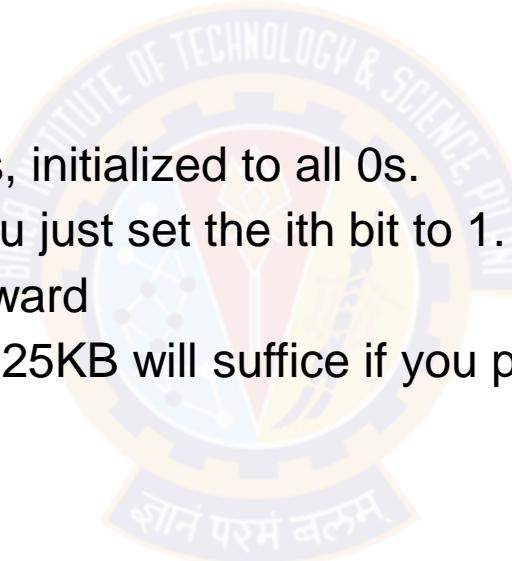
Filtering in a Stream

- Refers to selection of elements in a stream



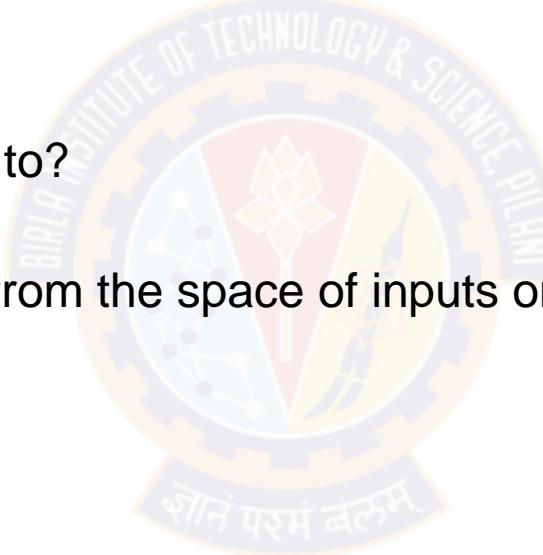
Exact Solution to Membership Problem

- Assume that we want to keep track of which of a million possible items seen
 - Items labelled with ID number
- Solution-1:
 - ✓ Keep an array of one million bits, initialized to all 0s.
 - ✓ Every time you see an item i , you just set the i th bit to 1.
 - ✓ Querying for item j is straightforward
 - ✓ The structure is very compact: 125KB will suffice if you pack the bits into memory.



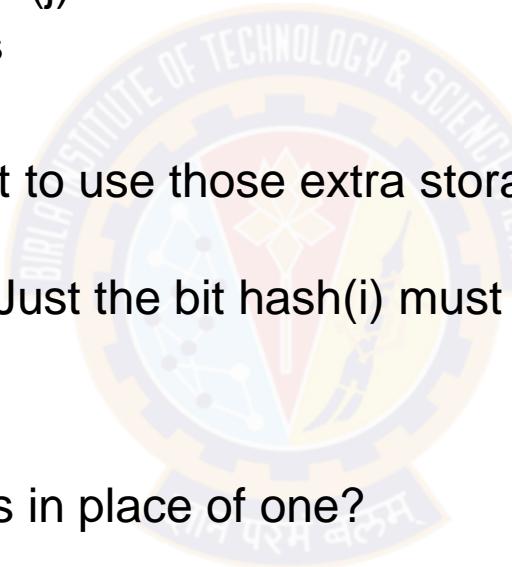
Exact Solution to Membership Problem(2)

- What if the size of the set is much larger / or not enumerable in practice?
 - ✓ names of customers, where the number of possible name strings is huge
 - ✓ All pair of IP's communicating
- Where does the solution takes you to?
 - ✓ Hash tables?
 - ✓ Use a hash function h to map from the space of inputs onto the range of indices for your table



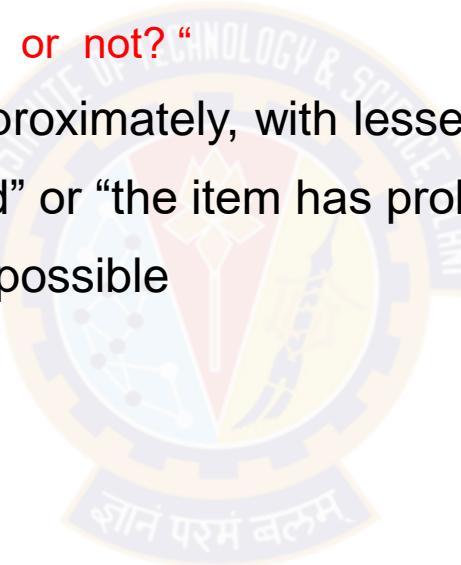
Exact Solution to Membership Problem(3)

- For an item i , set $\text{hash}(i)$ bit to 1
- What if for distinct i, j , $\text{hash}(i) = \text{hash}(j)$?
 - ✓ Use collision resolution techniques
- Let us assume that you do not want to use those extra storage, or complicate the mechanism to resolve collisions
 - ✓ That is, no additional storage. Just the bit $\text{hash}(i)$ must be relevant bit
 - ✓ Hard to handle collision
- What if we use more hash functions in place of one?



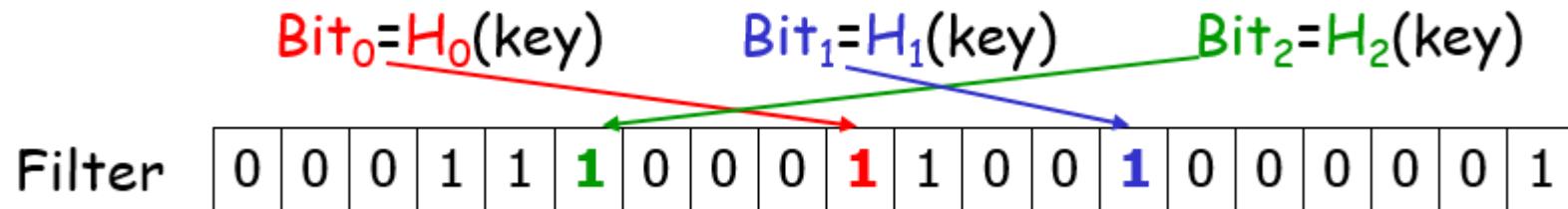
Exact Solution to Membership Problem(4)

- A compact data structure that summarizes a set of items
- Dictionary answering membership questions
 - ✓ “Is an item x stored in the structure or not?”
- Answers the membership question approximately, with lesser storage
- “the item has definitely not been stored” or “the item has probably been stored.”
- Listing of items in the dictionary is not possible



Blooms Filters

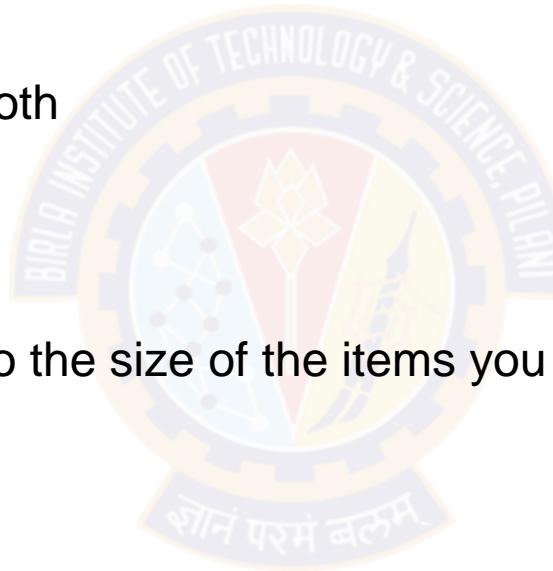
- Bloom filter: A bit vector that represents a set of keys
- A key is hashed d (e.g. d=3) times and represented by d bits



- Construct: for every key in the set, set its 3 bits in vector
- Membership Test: given a key, check if all its 3 bits are 1
 - ✓ Definitely not in the set if some bits are 0
 - ✓ May have false positives

Bloom filters (2)

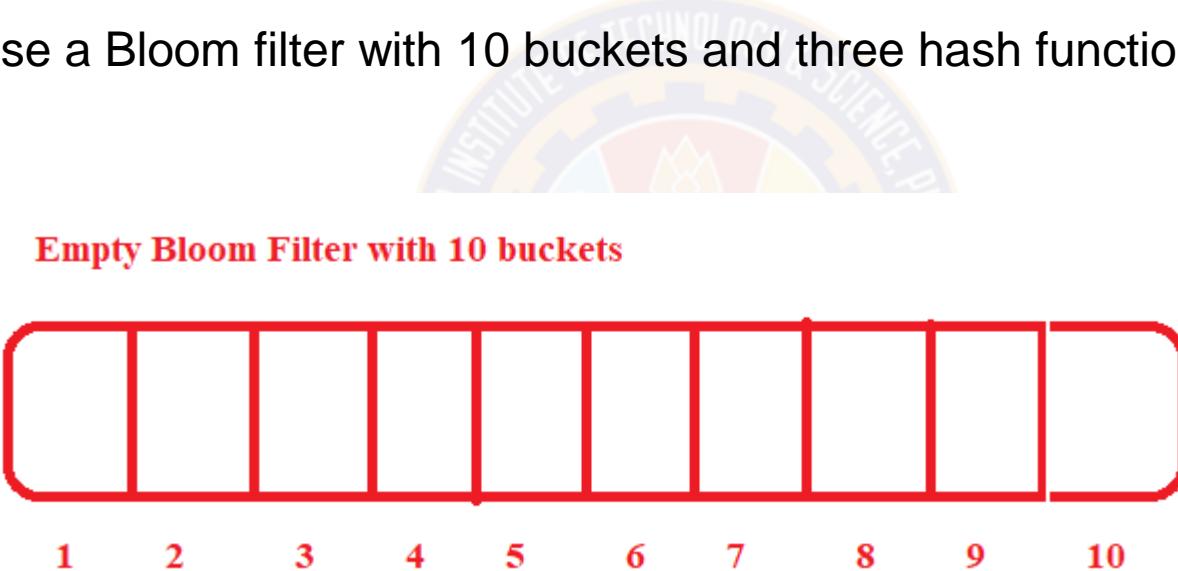
- Can very quickly answer variations on the Yes/No question
 - ✓ “is this item in the set?”, like “have I seen this item before?”
- Has constant time complexity for both
 - ✓ adding items
 - ✓ asking whether they are present
- Requires very little space relative to the size of the items you need to store and check



Bloom Filter with example

Step 0

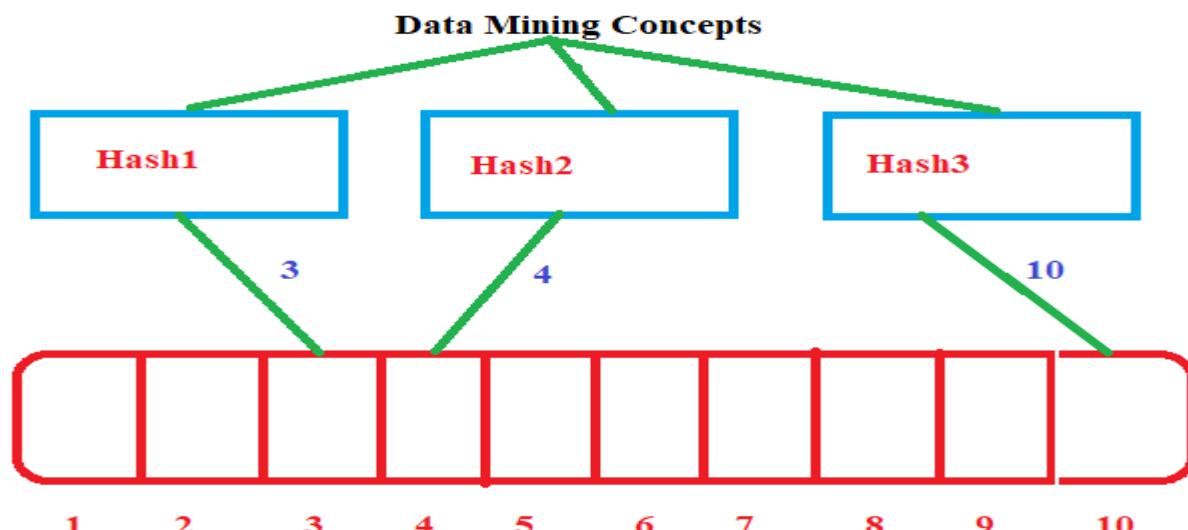
- Think of a Bloom filter as a large set of numbered buckets, each one starting off empty.
- Imagine we want to keep track of the books that I own.
- To do this, lets use a Bloom filter with 10 buckets and three hash functions.



Bloom Filter with example (2)

Step 1 - Initialization

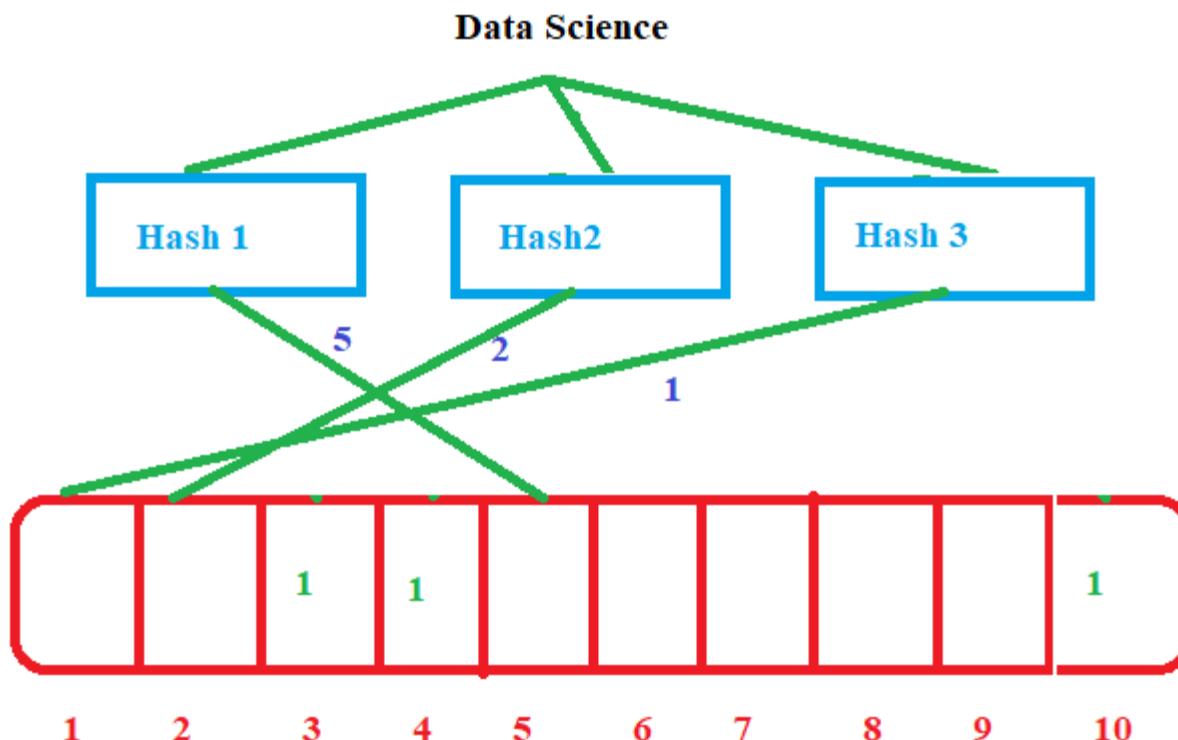
- Lets start by putting "Data Mining Concepts" into our Bloom filter
- Internally, the filter takes book name and passes it to the three hash functions, which return three identifiers.
- In this case, "Data Mining Concepts" hashes to identifiers 3, 4, and 10.
- The filter goes and fills up every bucket whose number matches one of the identifiers — 3, 4, and 10.
- It's able to do this very quickly because it can jump straight to each bucket to fill it up, just like you jump straight to page numbers from an index.



Bloom Filter with example (3)

Step 2 - Addition

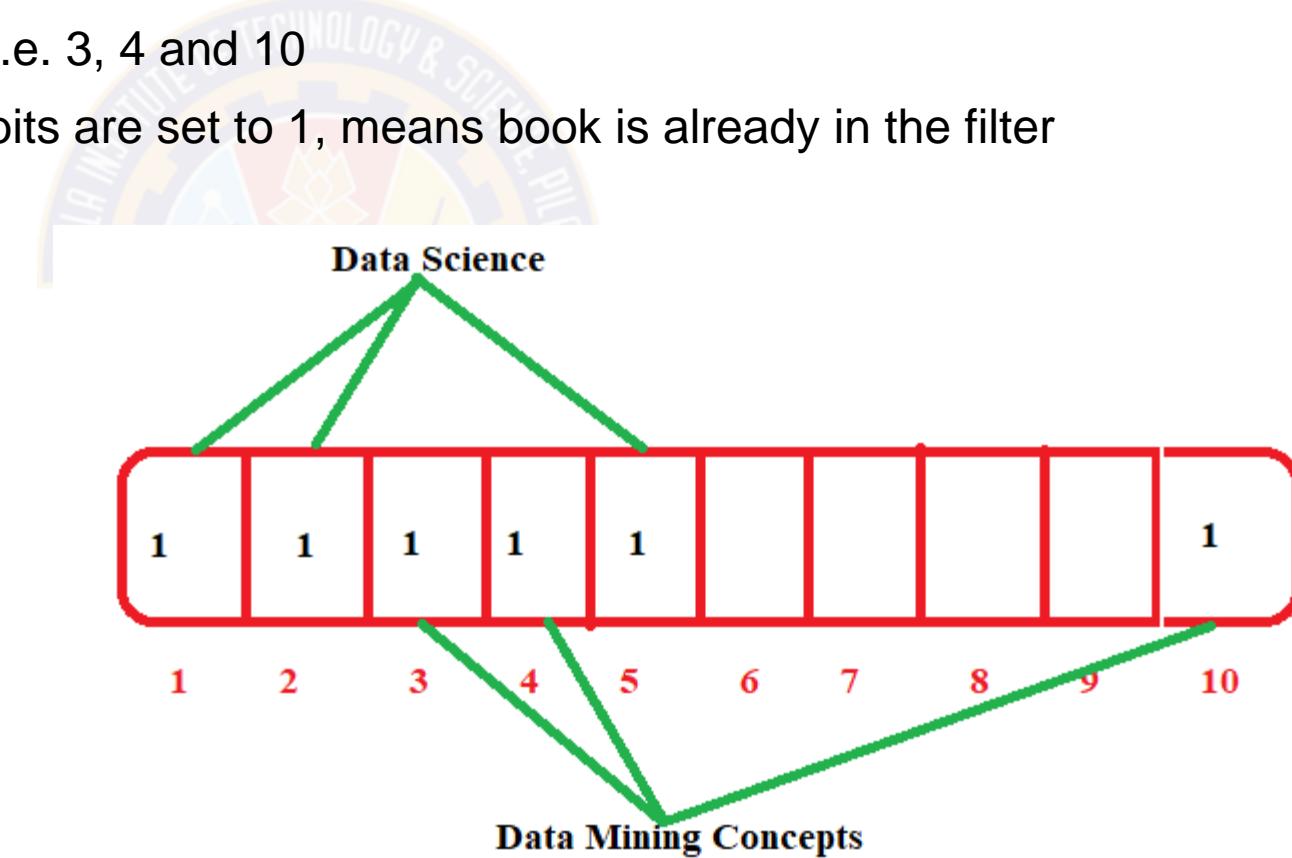
- Next we put "Data Science" into our filter.
- Using the same hash functions, "Data Science" hashes to 5, 2, and 1, so those buckets get filled up too.
- After we've done both of these operations, buckets 1, 2, 3, 4, 5, and 10 are full.



Bloom Filter with example (3)

Step 3 – Querying

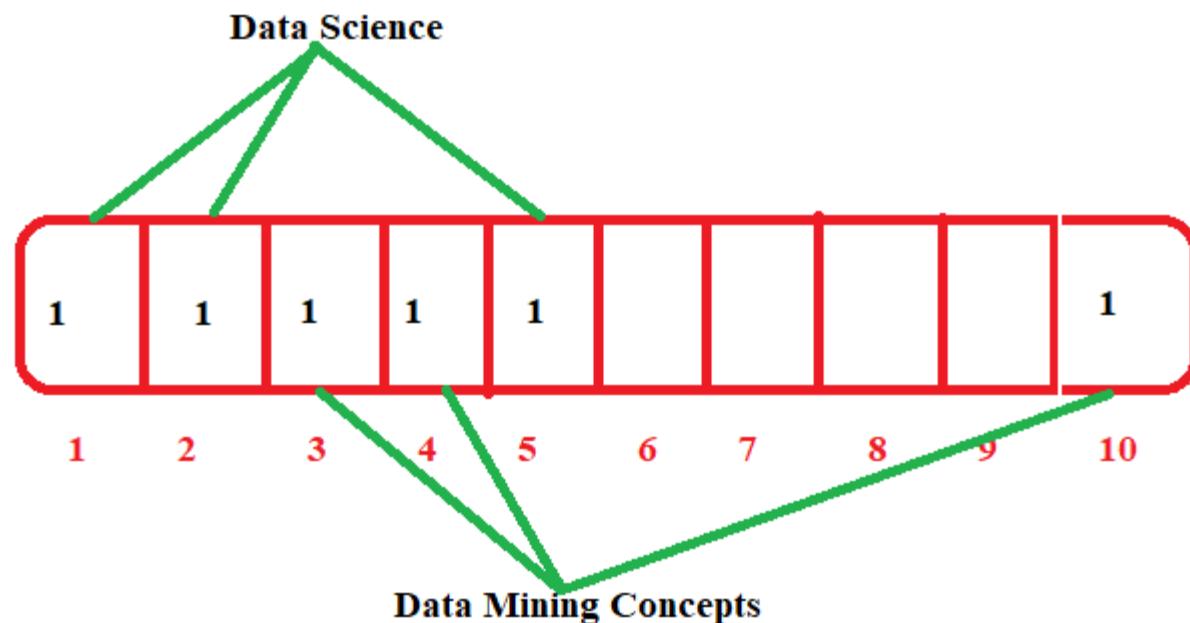
- Lets ask filter whether “Data Mining Concepts” is present or not?
- Book name will go through same three hash function
- Hash will result into three values i.e. 3, 4 and 10
- Go to positions 3, 4 and 10, if all bits are set to 1, means book is already in the filter



Bloom Filter with example (3)

Step 4 - Querying

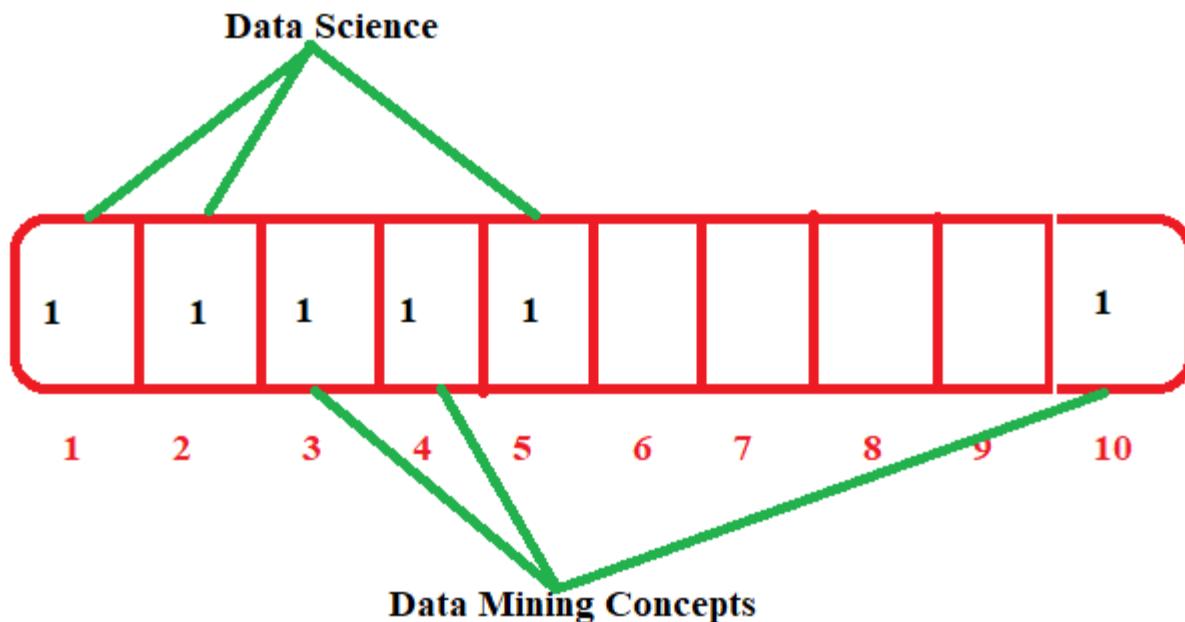
- Lets ask filter whether “Streaming Data” is present or not?
- Book name will go through same three hash function
- Lets say hash will result into three values i.e. 3, 4 and 7
- Go to positions 3, 4 and 7 , not all bits are set to 1, hence book is not seen earlier.



Bloom Filter with example (3)

Step 5 - False Positives

- Lets check book “Data Mining” is present or not
- Book name will go through same three hash function
- Lets say hash will result into three values i.e. 3, 4 and 10
- Now check positions related to 3, 4, and 10, all bits are set to 1 , means books is known one
- But actually filter never has seen that book





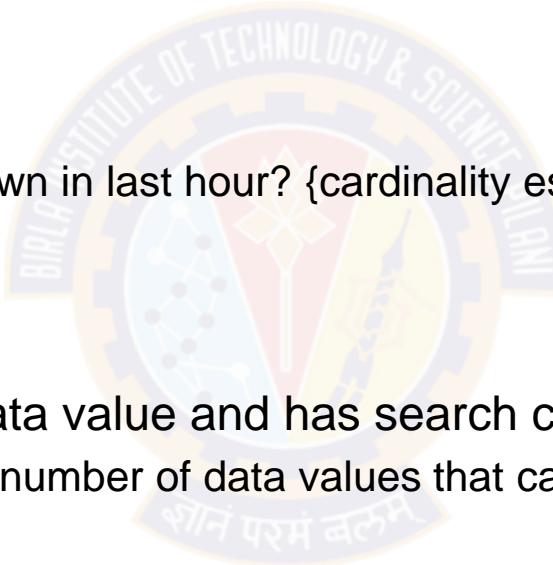
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Distinct Value Sketches

Pravin Y Pawar

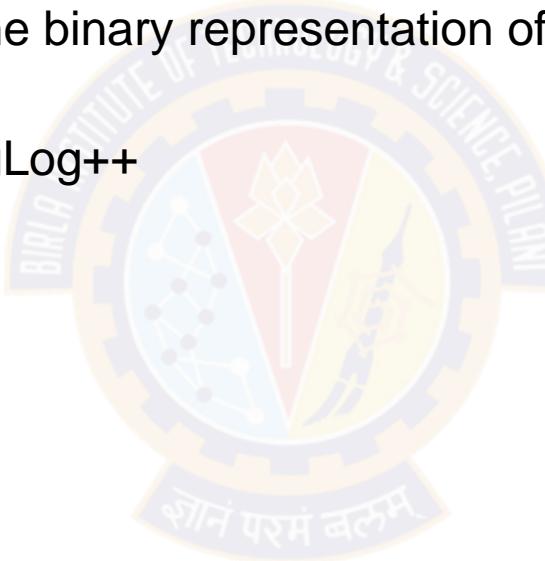
Counting Distinct Elements

- Easy task in non streaming world but not that easy in streaming world
 - ✓ Memory constraints are there so can not keep all stream data in memory
- Question to be answered :
 - ✓ How many distinct value were shown in last hour? {cardinality estimation}
- Approach1 :
- Use hash table that can hold the data value and has search capabilities
 - ✓ But still there will be limitations on number of data values that can be stored in hash table
- Use bit pattern based data structure for the same



Bit pattern based algorithm

- Based on the observation patterns of bits that occur at the beginning of binary value of each element of stream
- Mostly using the leading zeros in the binary representation of hash of stream element – cardinality is determined
- Example, HyperLogLog , HyperLogLog++



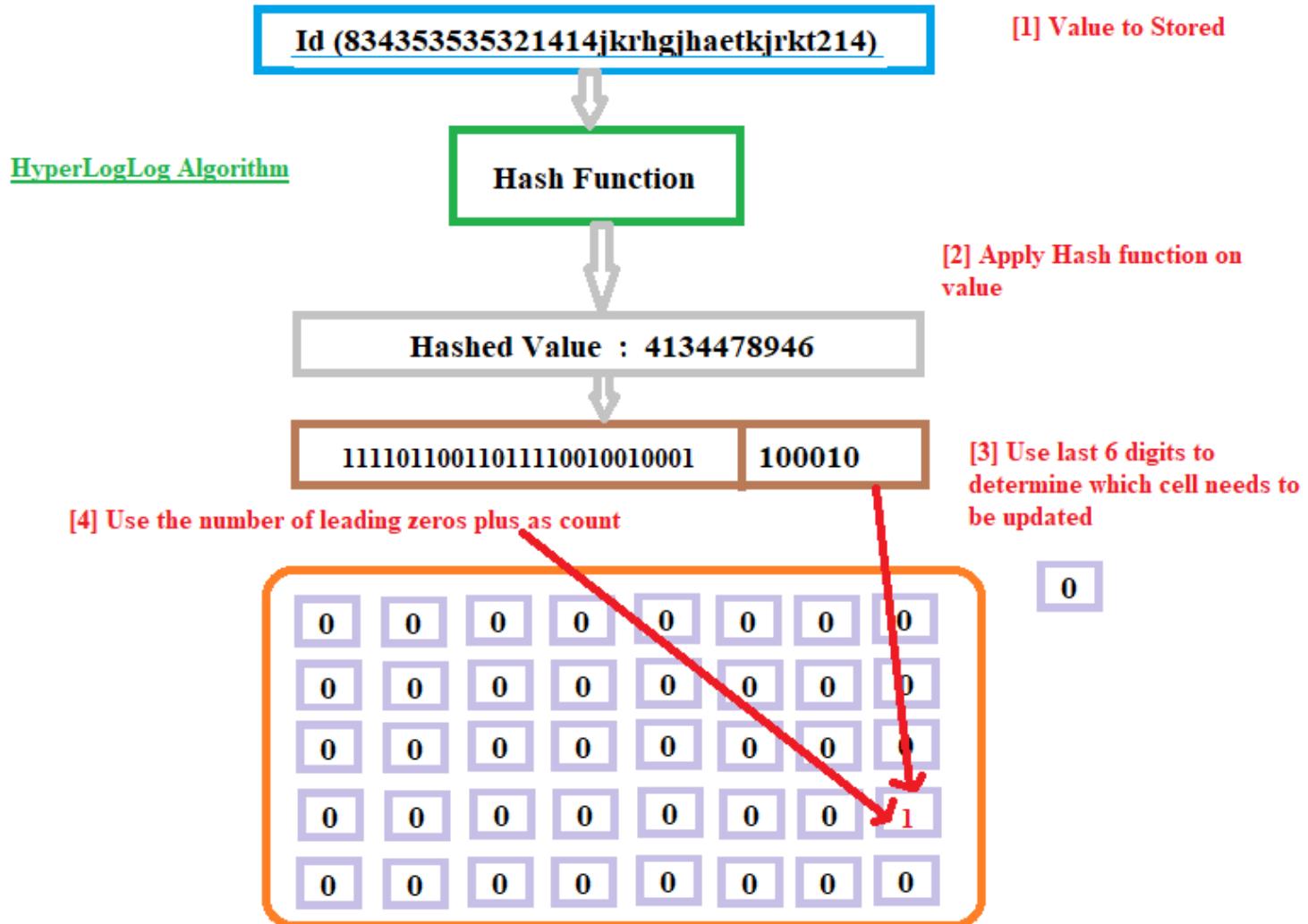
HyperLogLog

Working

- Step1 – Identify a unique ID for the data value
- Step2 – Pass the ID through a hash function , it will result into a hashed value
- Step3 – Hashed value is converted into a binary representation
- Step4 – Need to determine the place which needs to be updated and with what value.
 - ✓ Take the least 6 significant digits of binary number and convert it to a decimal value. That gives you the position where value needs to be updated
 - ✓ Count the number of leading zeros in the binary number , add one to it and use that number as a value to be stored in the position identified earlier
- Step5- Determine the distinct counts by taking the harmonic mean of all register values.

Reference : [Harmonic Mean](#)

HyperLogLog (2)





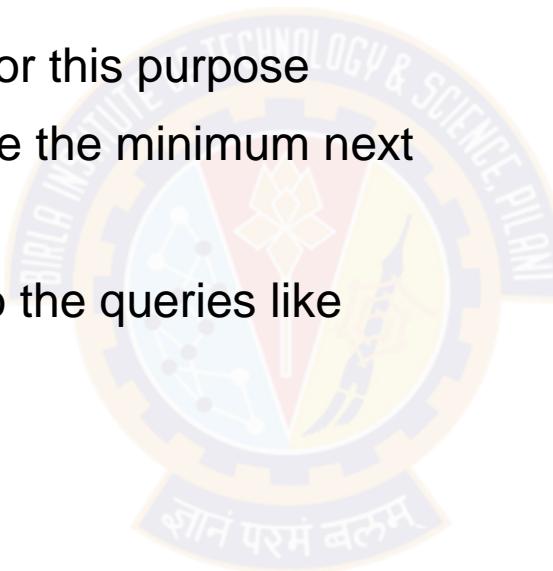
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

The Count-Min Sketch

Pravin Y Pawar

Frequency

- Question : How many times has stream element X occurred?
- Count-Min is the known algorithm for this purpose
- Designed to count first and compute the minimum next
- Can also be used to get answers to the queries like
 - ✓ Getting frequencies in a range



Count-Min algorithm

Working

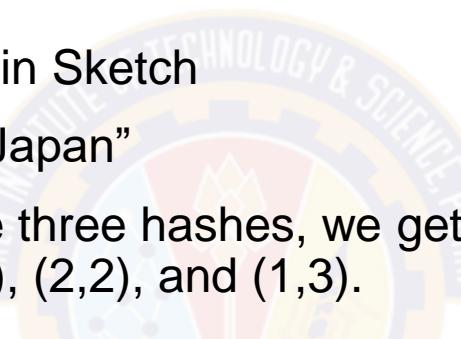
- The internal structure of a Count-Min Sketch is a table, similar to a Hash Table
- Count-Min Sketches use multiple hash functions, one for each column
- Initially, every cell in the Count-Min-Sketch is initialized to 0
- When an event occurs, the event's id is hashed over every column
- Each hash function outputs a row value, and the counter at each resulting row-column combination is incremented
- To query an event's count, take the minimum of that event's counts over all of the hash functions

Hash 1	Hash 2	Hash 3
0	0	0
0	0	0
0	0	0
0	0	0

Count-Min algorithm(2)

Example

- Lets say we want count how many times an article is read
- Lets have a 3-column, 4-row Count-Min Sketch
- First user decides to read “History of Japan”
- When that articles' id is hashed by the three hashes, we get 1, 2, and 1, indicating that we should increment the counters at (1,1), (2,2), and (1,3).



Hash 1	Hash 2	Hash 3
0	0	0
0	0	0
0	0	0
0	0	0

Hash 1	Hash 2	Hash 3
1	0	1
0	1	0
0	0	0
0	0	0

put(Japan)

hash(Japan) = (hash1(Japan), hash2(Japan), hash3(Japan)) = (1,2,1)

Count-Min algorithm(3)

Example (continued)

- Now suppose that another user reads “European History” article, which hashes to 1, 1, and 4
- We should increment the counters at (1,1), (1,2), and (4,3).

Hash 1	Hash 2	Hash 3
1	0	1
0	1	0
0	0	0
0	0	0

→

Hash 1	Hash 2	Hash 3
2	1	1
0	1	0
0	0	0
0	0	1

Count-Min algorithm(4)

Example (continued)

- Following that, yet another user reads “History of Japan”
- When that articles' id is hashed by the three hashes, we get 1, 2, and 1, indicating that we should increment the counters at (1,1), (2,2), and (1,3).



Hash 1	Hash 2	Hash 3		Hash 1	Hash 2	Hash 3
2	1	1		3	1	2
0	1	0	→	0	2	0
0	0	0		0	0	0
0	0	1		0	0	1

put(Japan)

hash(Japan) = (hash1(Japan), hash2(Japan), hash3(Japan)) = (1,2,1)

Count-Min algorithm(5)

Example (continued)

- Lets examine how many times users read "European History".
- "European History" hashes to 1,1, and 4, so should examine the counts at the cells (1,1), (1,2), and (4,3)
- These counts are 3, 1, and 1 respectively.
- Minimum of these counts i.e. 3, 1, 1 is 1, so article "European History" is read once.

	Hash 1	Hash 2	Hash 3
3	3	1	1
0	0	2	0
0	0	0	0
0	0	0	1

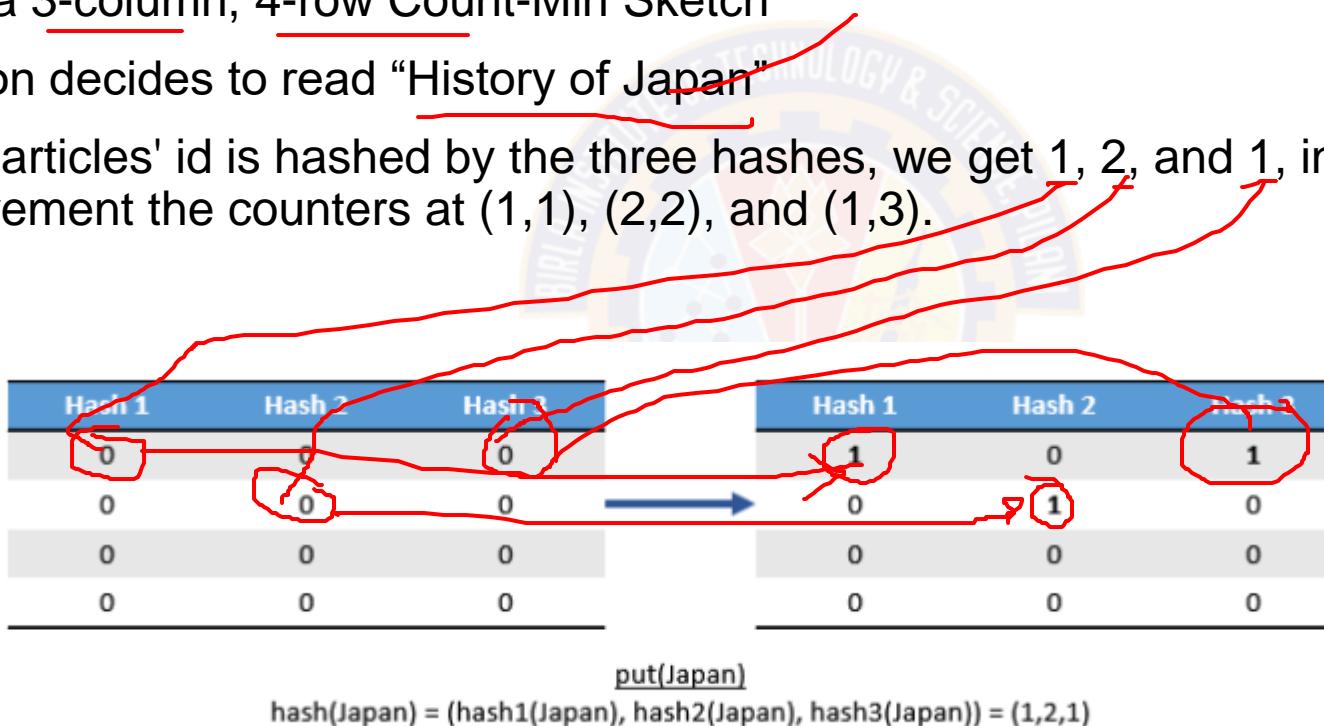
get (EHistroy)

$$\begin{aligned}\text{hash (EHistroy)} &= (\text{hash1 (EHistroy)}, \text{hash2 (EHistroy)}, \text{hash3 (EHistroy)}) = (1, 1, 4) \\ &= \min<3, 1, 1> = 1\end{aligned}$$

Count-Min algorithm(6)

Example (Incorrect Count)

- Lets consider a new sequence ,
- Lets have a 3-column, 4-row Count-Min Sketch
- First user on decides to read “History of Japan”
- When that articles' id is hashed by the three hashes, we get 1, 2, and 1, indicating that we should increment the counters at (1,1), (2,2), and (1,3).



Count-Min algorithm(7)

Example (Incorrect Count)

- Next user on decides to read “Blockchain”
- When that articles' id is hashed by the three hashes, we get 1, 1, and 1, indicating that we should increment the counters at (1,1), (1,2), and (1,3).

Hash 1	Hash 2	Hash 3
1	0	1
0	1	0
0	0	0
0	0	0

put(Blockchain)

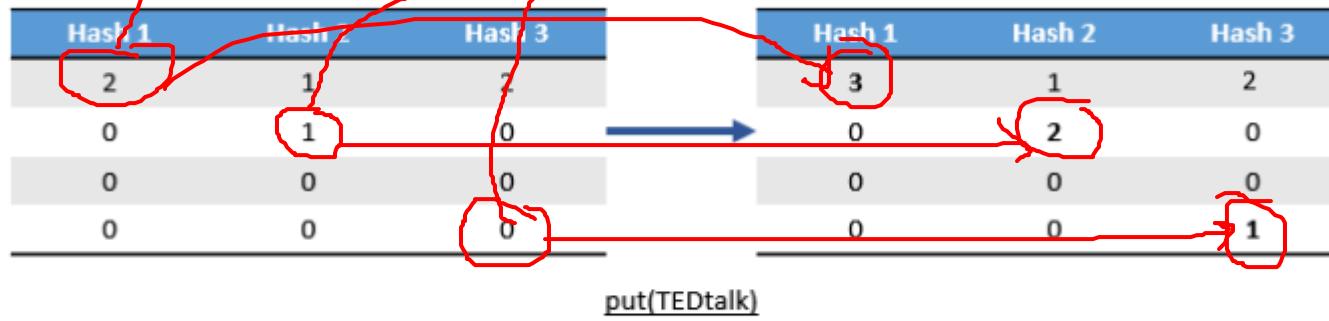
Hash 1	Hash 2	Hash 3
2	1	2
0	1	0
0	0	0
0	0	0

hash(Blockchain) = (hash1(Blockchain), hash2(Blockchain), hash3(Blockchain)) = (1,1,1)

Count-Min algorithm(8)

Example (Incorrect Count)

- Next user on decides to read "TedTalk"
- When that articles' id is hashed by the three hashes, we get 1, 2, and 4, indicating that we should increment the counters at (1,1), (2,2), and (4,3)



Count-Min algorithm(9)

Example (Incorrect Count)

- Lets examine how many times users read "Hello".
- "Hello" hashes to 1,1, and 4, so should examine the counts at the cells (1,1), (1,2), and (4,3).
- These counts are 3, 1, and 1 respectively.
- Minimum of these counts i.e. 3, 1, 1 is 1, so article "Hello" is read once.

Hash 1	Hash 2	Hash 3
3	1	1
0	2	0
0	0	0
0	0	1

get(Hello)
hash(Hello) = (hash1(Hello), hash2(Hello), hash3(Hello)) = (1,1,4)
count(Hello) = min<3,1,1> = 1

- That's incorrect! 'Hello' is never read by any user.

Count-Min algorithm(10)

Example (Incorrect Count)

- Why?
 - ✓ Even though “Hello” was never watched in this sequence, each of the articles collided with “Hello” in at least one hash function.
 - ✓ As a result, when we query the view count for “Hello”, we are incorrectly told that it has been watched once.
- Should use it when goal is to understand general trends rather than to make precise measurements
- Rectification
 - ✓ Can reduce both the probability and magnitude of errors by using a larger table



Thank You!

In our next session: The Bloom Filter



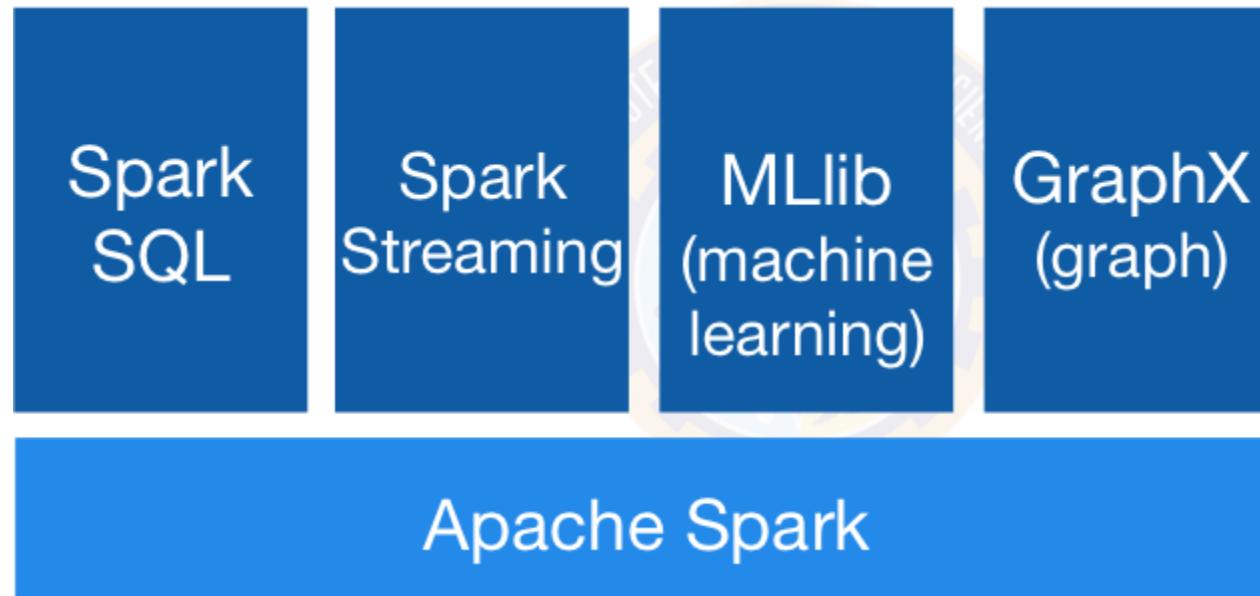
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark SQL -Introduction

Pravin Y Pawar

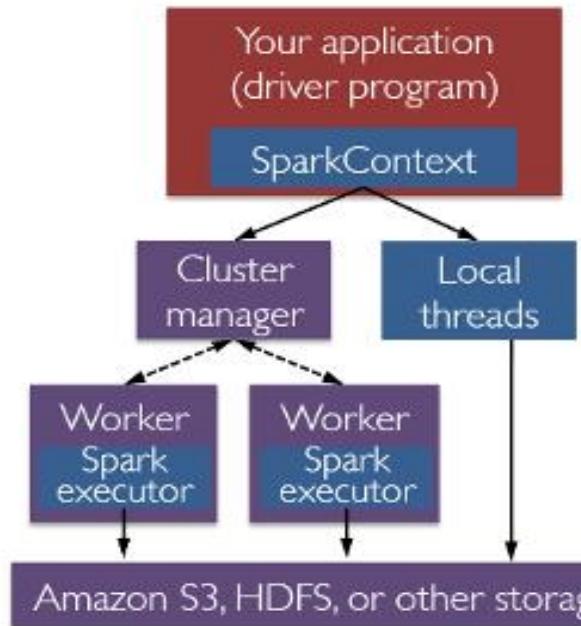
Spark Components

Spark Architectural Component



Spark Application Execution

Cluster Components

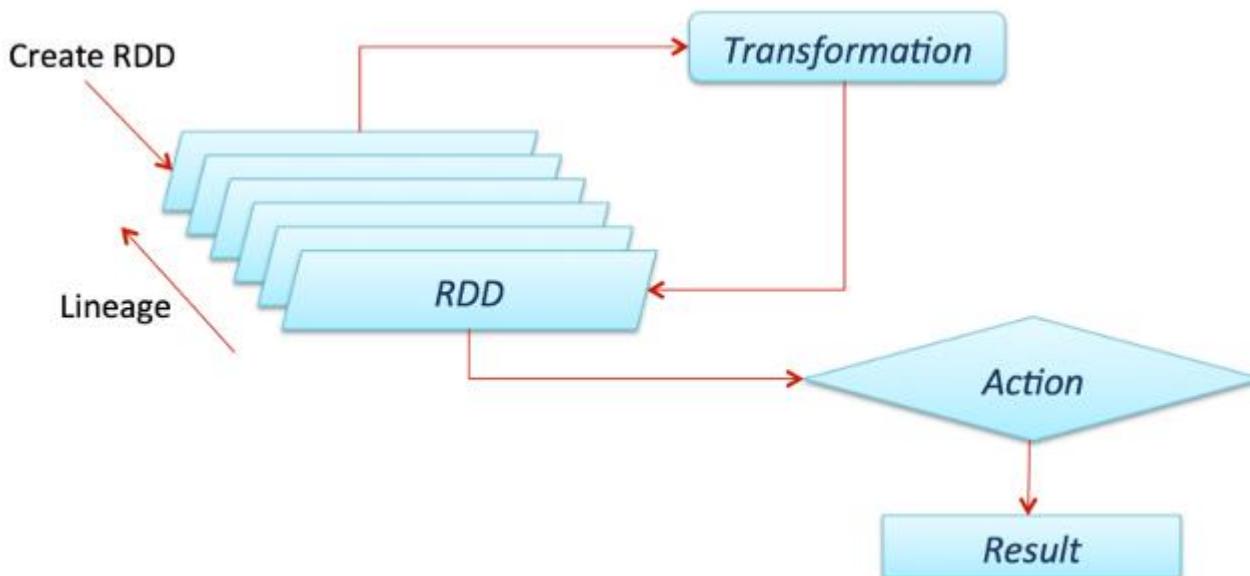


- A Spark program is two programs:
 - » A **driver program** and a **workers program**
- Worker programs run on cluster nodes or in local threads
- RDDs are distributed across workers

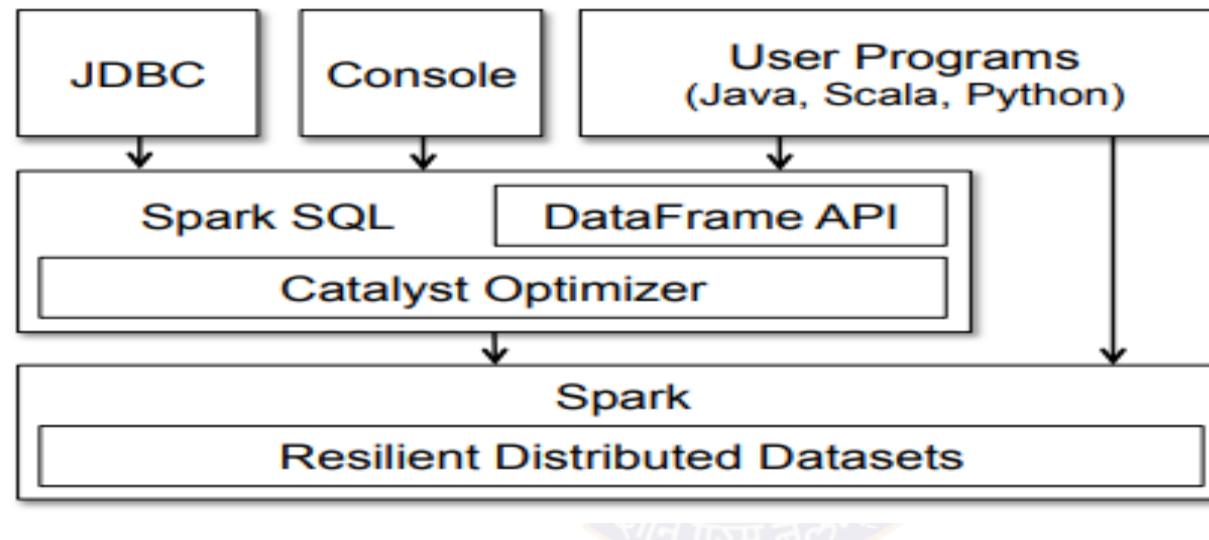
Spark RDD

Main Abstraction of Spark

- The biggest contributor behind all of Spark's success stories
- A data structure, or rather a distributed memory abstraction
 - ✓ that allows programmers to perform in-memory computations on large distributed clusters while retaining aspects like fault tolerance
 - ✓ That parallelize a lot of computations and transformations and track the whole lineage of transformations, which can help in efficiently recomputing lost data

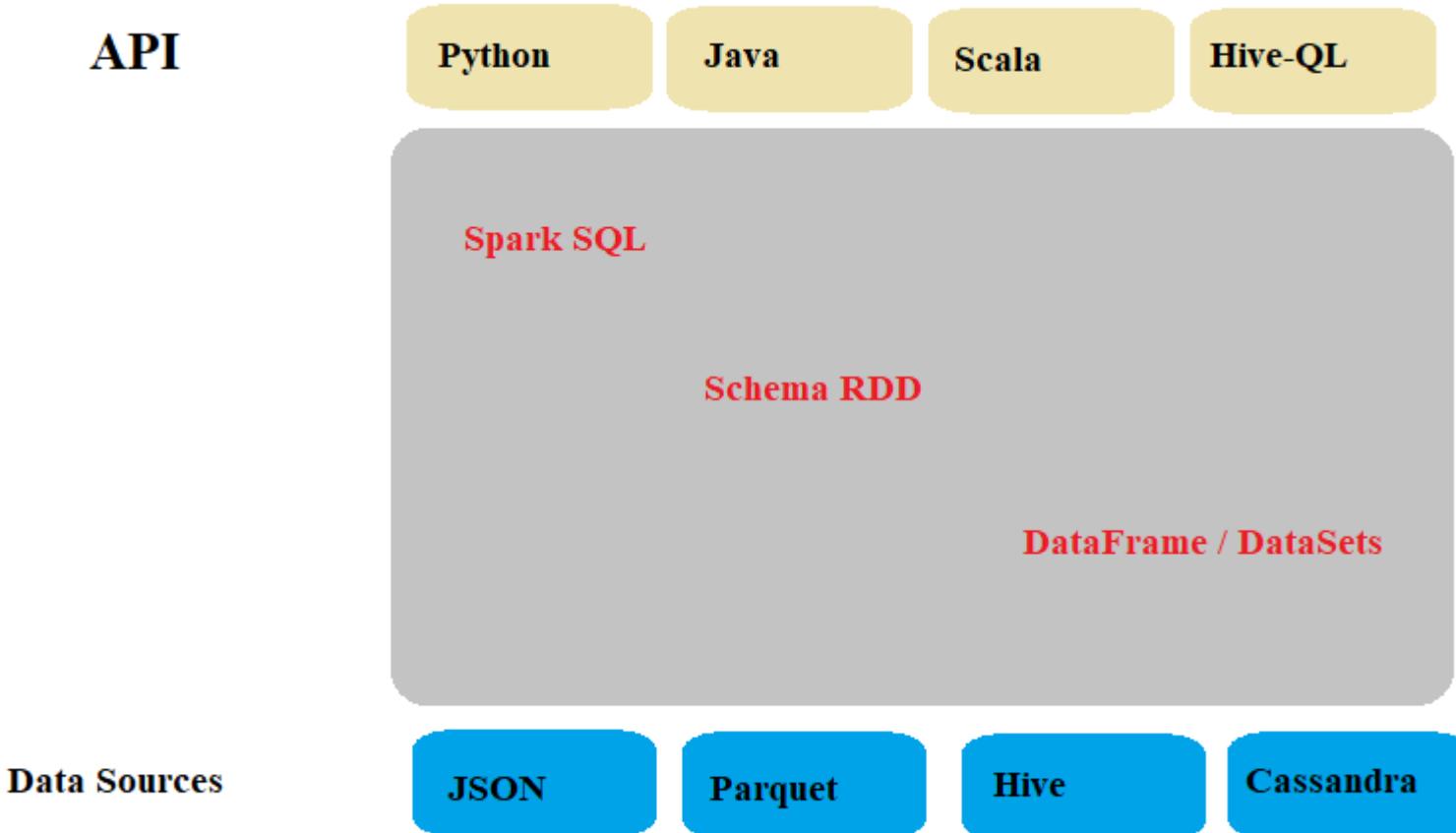


Spark SQL Architecture



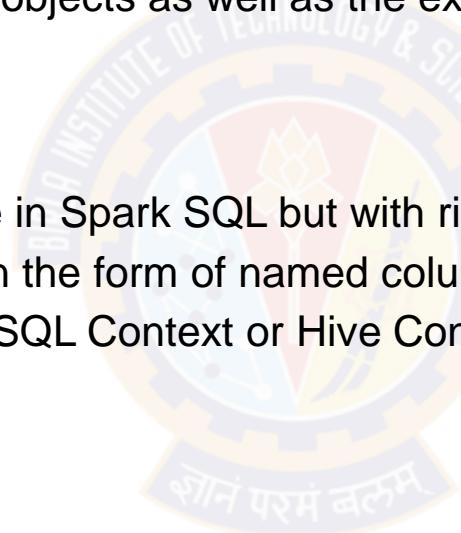
Spark SQL Architecture (2)

Another View



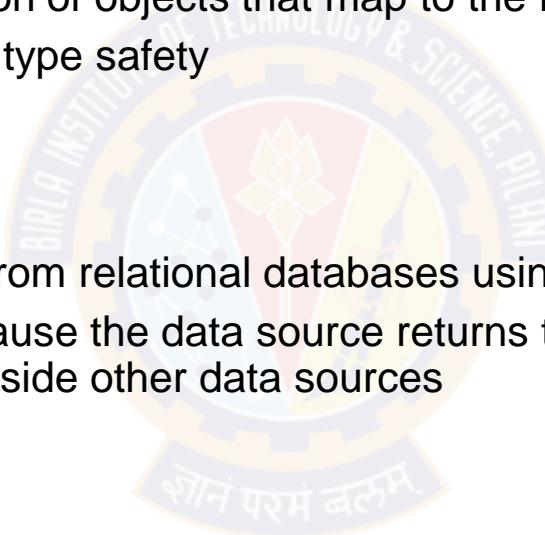
Spark SQL Interfaces

- **SQLContext**
 - ✓ entry point for working with structured data (rows and columns) in Apache Spark
 - ✓ Allows the creation of DataFrame objects as well as the execution of SQL queries
- **DataFrame**
 - ✓ is equivalent to the relational table in Spark SQL but with richer optimization
 - ✓ is a distributed collection of data in the form of named column and row
 - ✓ For accessing data frames either SQL Context or Hive Context is needed
- **Hive Context**
 - ✓ Used while working with Hive tables
 - ✓ is more battle-tested and provides a richer functionality than SQLContext



Spark SQL Interfaces(2)

- DataSets
 - ✓ Provides the benefits of RDDs
 - ✓ Strongly-typed, immutable collection of objects that map to the relational schema already present
 - ✓ extend the benefit of compile-time type safety
- JDBC Datasource
 - ✓ JDBC data source can read data from relational databases using JDBC API
 - ✓ has preference over the RDD because the data source returns the results as a DataFrame, can be handled in Spark SQL or joined beside other data sources



Features of Spark SQL

- Integrated
 - ✓ Seamlessly mix SQL queries with Spark programs
 - ✓ Can query structured data as a distributed dataset (RDD) in Spark, with integrated APIs in Python, Scala and Java
- Unified Data Access
 - ✓ Load and query data from a variety of sources including Apache Hive tables, parquet files and JSON files
- Hive Compatibility
 - ✓ Run unmodified Hive queries on existing warehouses
 - ✓ Need to simply install Spark alongside Hive
- Standard Connectivity
 - ✓ Connect through JDBC or ODBC
- Scalability
 - ✓ Same engine for both interactive and long queries
 - ✓ Takes advantage of the RDD model to support mid-query fault tolerance, letting it scale to large jobs too



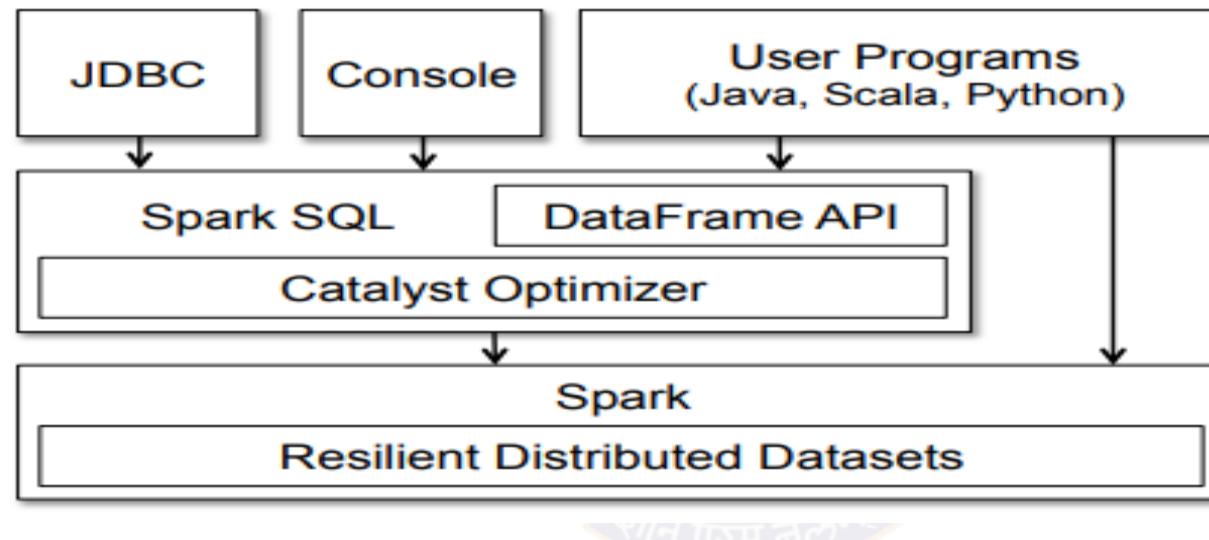


BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark SQL - DataFrame

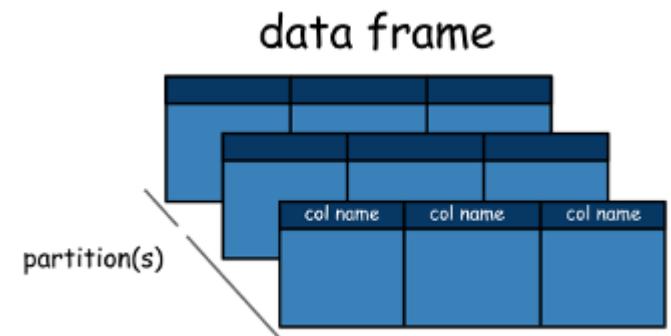
Pravin Y Pawar

Spark SQL Architecture



DataFrames

- Is very powerful and allows users to finally intermix procedural and relational code!
- Advanced functions like UDFs (user-defined functions) can also be exposed in SQL
- Is basically a distributed collection of rows (row types) with the same schema
 - ✓ Basically a Spark Dataset organized into named columns
 - ✓ Datasets are an extension of the DataFrame API that provides a type-safe, object-oriented programming interface
 - ✓ available only in Java and Scala
- A DataFrame is equivalent to a table in a relational database but with more optimizations under the hood
 - ✓ Can also be manipulated in similar ways to the "native" distributed collections in Spark (RDDs).



DataFrames Properties

- Keeps track of schema and support various relational operations
 - ✓ that lead to a more optimized execution
- Can be constructed from tables
 - ✓ like existing Hive tables or even from existing RDDs
- Can be manipulated with direct SQL queries and also using the DataFrame DSL (domain-specific language),
 - ✓ various relational operators and transformers such as where and groupBy can be used
- Can also be viewed as an RDD of row objects, allowing users to call procedural Spark APIs such as map
- Are lazy, in that each DataFrame object represents a logical plan to compute a dataset
 - ✓ execution occurs until the user calls a special "output operation" such as save



Starting Point: SparkSession

- The entry point into all functionality in Spark is the SparkSession class
- To create a basic SparkSession, just use SparkSession.builder

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Creating DataFrames

- With a SparkSession, applications can create DataFrames from
 - ✓ an existing RDD
 - ✓ from a Hive table
 - ✓ from Spark data sources
- Following creates a DataFrame based on the content of a JSON file



```
# spark is an existing SparkSession
df = spark.read.json("examples/src/main/resources/people.json")
# Displays the content of the DataFrame to stdout
df.show()
# +---+---+
# | age| name|
# +---+---+
# | null|Michael|
# |  30| Andy|
# |  19| Justin|
# +---+---+
```

DataFrame Operations

Untyped Dataset Operations

- DataFrames provide a domain-specific language for structured data manipulation
 - ✓ Scala, Java, Python and R
- DataFrames are just Dataset of Rows in Scala and Java API
 - These operations are also referred as “untyped transformations” in contrast to “typed transformations” come with strongly typed Scala/Java Datasets

Some basic examples of structured data processing:

```
# spark, df are from the previous example
# Print the schema in a tree format
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)

# Select only the "name" column
df.select("name").show()
# +---+
# |  name|
# +---+
# |Michael|
# |  Andy|
# | Justin|
# +---+
```

DataFrame Operations (2)

More Examples

```
# Select only the "name" column
df.select("name").show()
# +---+
# | name|
# +---+
# |Michael|
# | Andy|
# | Justin|
# +---+  
  
# Select everybody, but increment the age by 1
df.select(df['name'], df['age'] + 1).show()
# +---+-----+
# | name| (age + 1)|
# +---+-----+
# |Michael|    null|
# | Andy|        31|
# | Justin|       20|
# +---+-----+
```

DataFrame Operations (3)

More Examples

```
# Select people older than 21
df.filter(df["age"] > 21).show()
# +---+
# |age|name|
# +---+---+
# | 30|Andy|
# +---+---+

# Count people by age
df.groupBy("age").count().show()
# +---+---+
# | age|count|
# +---+---+
# | 19|    1|
# | 20|    1|
# | 30|    1|
# +---+---+
```

Running SQL Queries Programmatically

- The `sql` function on a `SparkSession` enables applications to run SQL queries programmatically and returns the result as a `DataFrame`.

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
# +---+-----+
# | age| name|
# +---+-----+
# | 11| Michael|
# | 30| Andy|
# | 19| Justin|
# +---+-----+
```

Global Temporary View

- Temporary views in Spark SQL are session-scoped
 - ✓ will disappear if the session that creates it terminates
- If you want to have a temporary view that is shared among all sessions and keep alive until the Spark application terminates
 - ✓ can create a global temporary view
- Global temporary view is tied to a system preserved database `global_temp`
 - ✓ must use the qualified name to refer it, e.g. `SELECT * FROM global_temp.view1`

```
# Register the DataFrame as a global temporary view
df.createGlobalTempView("people")

# Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
# +-----+
# | age|  name|
# +----+----+
# | null|Michael|
# |  30|  Andy|
# |  19| Justin|
# +----+----+
```

Global Temporary View (2)

Across sessions

```
# Register the DataFrame as a global temporary view
df.createGlobalTempView("people")

# Global temporary view is tied to a system preserved database `global_temp`
spark.sql("SELECT * FROM global_temp.people").show()
# +---+-----+
# | age| name|
# +---+-----+
# | null|Michael|
# | 30| Andy|
# | 19| Justin|
# +---+-----+

# Global temporary view is cross-session
spark.newSession().sql("SELECT * FROM global_temp.people").show()
# +---+-----+
# | age| name|
# +---+-----+
# | null|Michael|
# | 30| Andy|
# | 19| Justin|
# +---+-----+
```



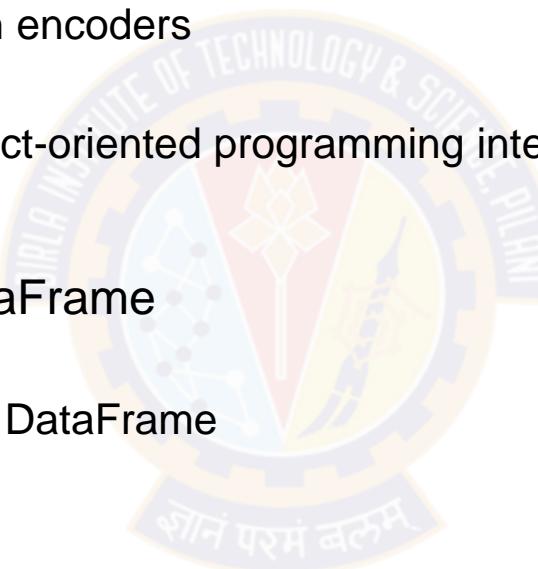
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Spark Dataset

Pravin Y Pawar

Spark SQL - Dataset

- Dataset is a data structure in SparkSQL which is strongly typed and is a map to a relational schema
 - ✓ Represents structured queries with encoders
 - ✓ Is an extension to data frame API
 - ✓ Provides both type safety and object-oriented programming interface
- Clubs the features of RDD and DataFrame
 - ✓ Provides the convenience of RDD
 - ✓ Gives performance optimization of DataFrame
 - ✓ Static type-safety of Scala
- Provides a more functional programming interface to work with structured data



Creating Datasets

- Datasets are similar to RDDs, however, instead of using Java serialization or Kryo they use a specialized Encoder to serialize the objects for processing or transmitting over the network

```
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +---+---+
// |name|age|
// +---+---+
// |Andy| 32|
// +---+---+
```

DataFrame to Dataset

Example

```
case class Person(name: String, age: Long)

// DataFrames can be converted to a Dataset by providing a class. Mapping will be done by name
val path = "examples/src/main/resources/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +---+-----+
// | age|  name|
// +---+-----+
// | null|Michael|
// |  30|  Andy|
// |  19| Justin|
```

Interoperating with RDDs

Inferring the Schema Using Reflection

- Method uses reflection to infer the schema of an RDD that contains specific types of objects
- Reflection-based approach leads to more concise code and works well when you already know the schema while writing your Spark application.

```
from pyspark.sql import Row

sc = spark.sparkContext

# Load a text file and convert each line to a Row.
lines = sc.textFile("examples/src/main/resources/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = spark.createDataFrame(people)
schemaPeople.createOrReplaceTempView("people")

# SQL can be run over DataFrames that have been registered as a table.
teenagers = spark.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

# The results of SQL queries are DataFrame objects.
# rdd returns the content as an :class:`pyspark.RDD` of :class:`Row`.
teenNames = teenagers.rdd.map(lambda p: "Name: " + p.name).collect()
for name in teenNames:
    print(name)
# Name: Justin
```



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Aggregation of Streaming Data

Pravin Y Pawar

Example

Unending Stream of Data

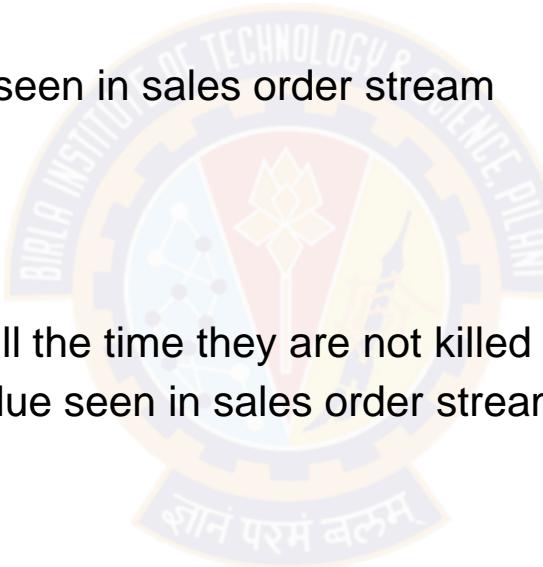
- Critical / time bound processing requirements
 - ✓ Telecommunications companies keep track of the activity on their networks
 1. to identify overall network health
 2. to spot anomalies
 3. To detect changes in behavior - millions of network events per hour, per network element



Queries in Streaming system

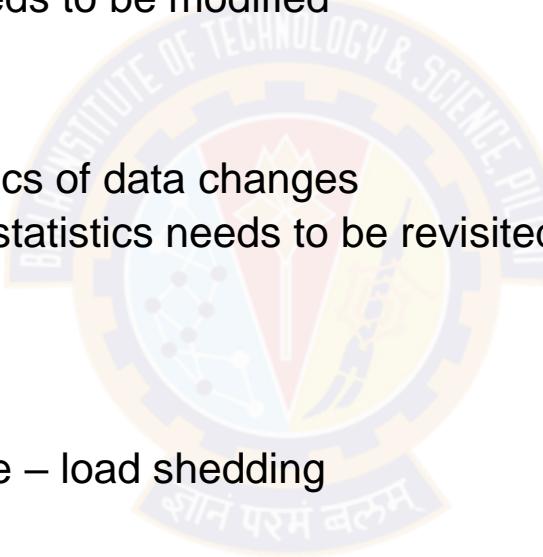
Types

- Ad-hoc queries
 - ✓ One time queries on a stream
 - ✓ Fire and forget
 - ✓ E.g. what is minimum sales value seen in sales order stream
- Continuous queries
 - ✓ Running all the times
 - ✓ Registered only once , continues till the time they are not killed
 - ✓ E.g. what is the minimum sales value seen in sales order stream in last hour



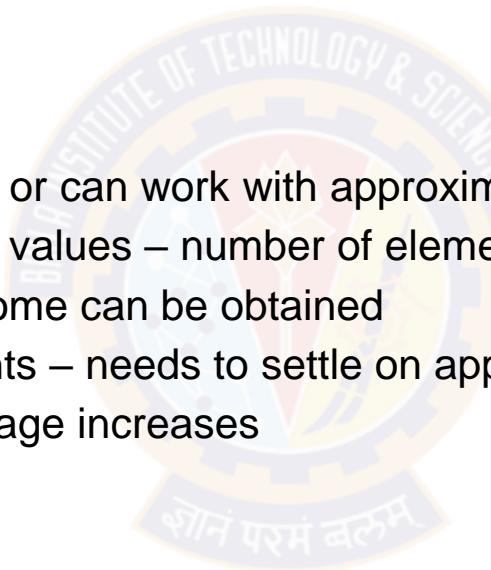
Constraints for Streaming Queries

- One-pass
 - ✓ Data is processed only once
 - ✓ Many of data mining algorithm needs to be modified
- Concept drift
 - ✓ Over the period of time, the statistics of data changes
 - ✓ Predictive models based old data statistics needs to be revisited
- Resource constraints
 - ✓ No control over arrival rate of data
 - ✓ Tuples can be dropped at pick time – load shedding
- Domain constraints
 - ✓ Particular to a business scenario
 - ✓ E.g. Storage required for maintaining tweets generated by users – looks like a resource issue but caused by business



Basic Aggregation

- Counting and summation of various data elements
 - ✓ Frameworks has limited built in support for aggregations and mostly its left for users to implement
- Requirements
 - ✓ Whether exact outcome is needed or can work with approximate aggregate values?
 - ✓ Depends on the cardinality of data values – number of elements in data
 - ❖ If less elements – exact outcome can be obtained
 - ❖ If too large number of elements – needs to settle on approximate answers as computation time and storage increases





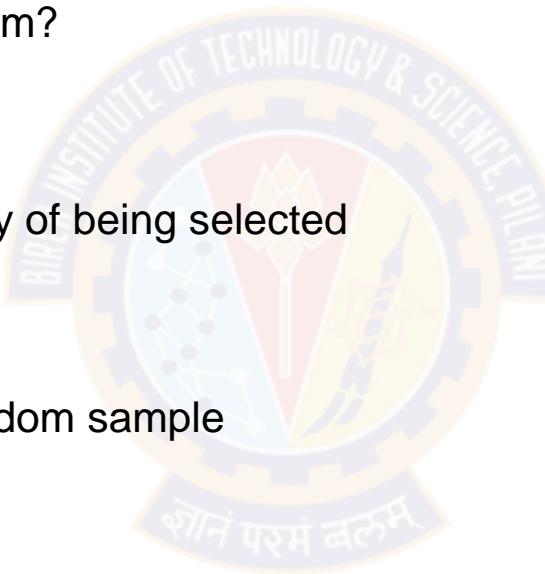
BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Sampling Algorithm

Pravin Y Pawar

Sampling

- Select a subset of data to be analyzed such that it has approximately the same properties of the original data
 - ✓ Do we have the length of the stream?
- Random Sampling
 - ✓ Each element has equal probability of being selected
- Reservoir sampling
 - ✓ A technique to maintain online random sample



Reservoir Sampling

- Used to maintain an online random sample
- Maintains a sample of size k , known as reservoir
- Every new element has a probability k/n of replacing an old element in the reservoir
- **Let us say, $k=1$:** An item in a stream of length n replaces the element in the reservoir with probability $1/n$



Reservoir Sampling Algorithm

input: S : a stream of values

k : size of the reservoir

begin

 /* Creates uniform sample of fixed size k */

 Insert the first k elements into the reservoir;

foreach $v \in S$ **do**

 Let i be the position of v ;

 Generate a random integer M in the range $1, \dots, i$;

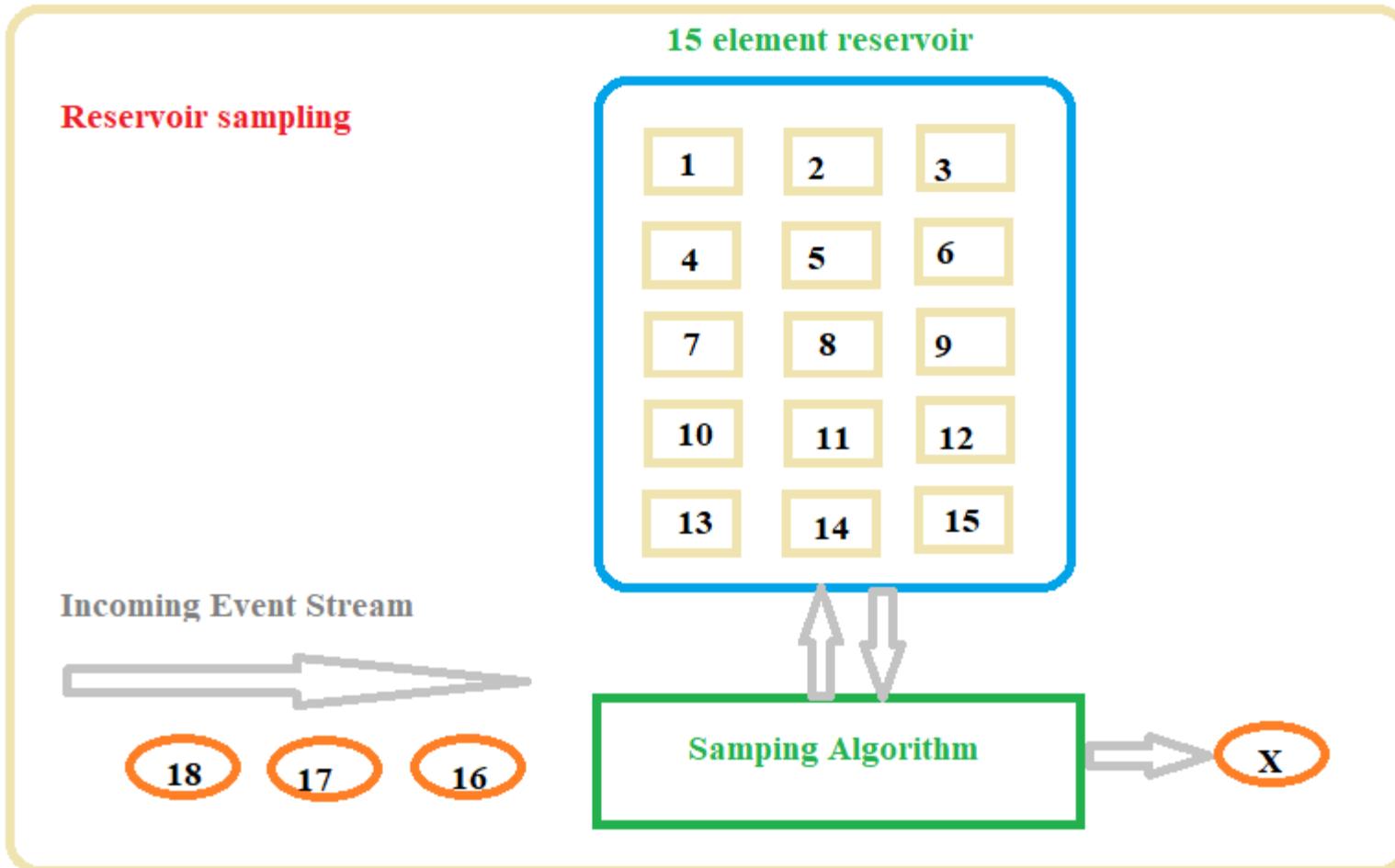
if $M \leq k$ **then**

 Insert v in the reservoir;

 Delete an instance from the reservoir at random.

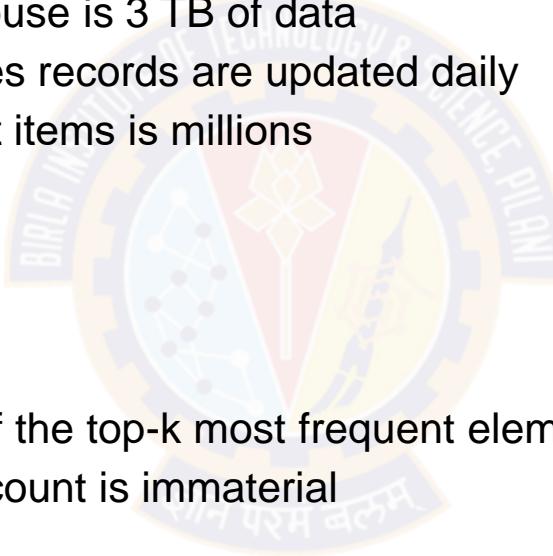
end

Reservoir Sampling Example



Hot List Problem

- Hot List for a Retail Business
 - ✓ Retail data warehouse of big ecommerce platform
 - ✓ The actual size of the data warehouse is 3 TB of data
 - ✓ Hundreds of gigabytes of new sales records are updated daily
 - ✓ Order of magnitude of the different items is millions
 - ✓ Restricted memory
 - ✓ The goal is to
 - ❖ continuously maintain a list of the top-k most frequent elements in a stream
 - ❖ rank the items and absolute count is immaterial



Hot List Problem – Frequent Algorithm

```
input:  $S$ : A Sequence of Examples
begin
    foreach example ( $e$ )  $\in S$  do
        if  $e$  is monitored then
            ↘ Increment  $Count_e$ ;
        else
            if there is a  $Count_j == 0$  then
                ↘ Replace element  $j$  by  $e$  and initialize  $Count_e = 1$ ;
            else
                ↘ Subtract 1 to each  $Count_i$ ;
    end
```



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Decaying Window

Pravin Y Pawar

Most Popular Element

- Stream of elements – Movie Tickets
- “Maintain a summary of most popular movies currently”
 - ✓ Currently is a vague / imprecise term – doesn’t refer to a specific time frame – no fixed size sliding window
 - ✓ Movie that sold $2n$ ticket a month back is not as popular as the movie that sold n movies tickets last few days.
- Approach -1
 - ✓ Choose a window
 - ✓ Estimate number of tickets for each movie and rank movie accordingly.
 - ✓ Within a window, the oldest movie and the newest movie receive same importance of getting reported as popular

Decaying Window

Idea

- Attach weights to elements in sliding window
- Recent elements receive higher weights, with the older elements receive decaying weights
- For a new element
 - ✓ First reduce the weight of all the existing elements by a constant factor
 - ✓ Assign the new element with a specific weight
 - ✓ The aggregate sum of the decaying exponential weights can be calculated using the following formula.
- Let stream is $a_1, a_2, a_3 \dots$ and taking sum of stream as follows

$$\begin{aligned} & - \sum_{i=0}^{t-1} a_{t-i} (1 - c)^i \\ & - c \text{ is a small constant such as } 10^{-6} \text{ or } 10^{-9} \end{aligned}$$

Decaying Window(2)

Working

- The decaying window algorithm
 - ✓ tracks the most recurring elements in an incoming data stream
 - ✓ also discounts any random spikes or spam requests that might have boosted an element's frequency
- Working
 - ✓ Assign a score or weight to every element of the incoming data stream.
 - ✓ Calculate the aggregate sum for each distinct element by adding all the weights assigned to that element
 - ✓ The element with the highest total score is listed as trending or the most popular.

Decaying Window Example

Example

- For example, consider a sequence of twitter tags below:

fifa, ipl, fifa, ipl, ipl, ipl, fifa

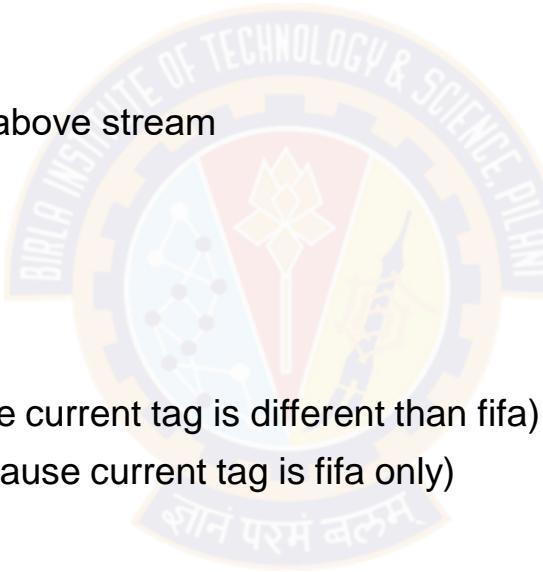
- Assume that each element in sequence has weight of 1.
- Let's c be 0.1



Decaying Window Example(2)

The aggregate sum calculation for “fifa”

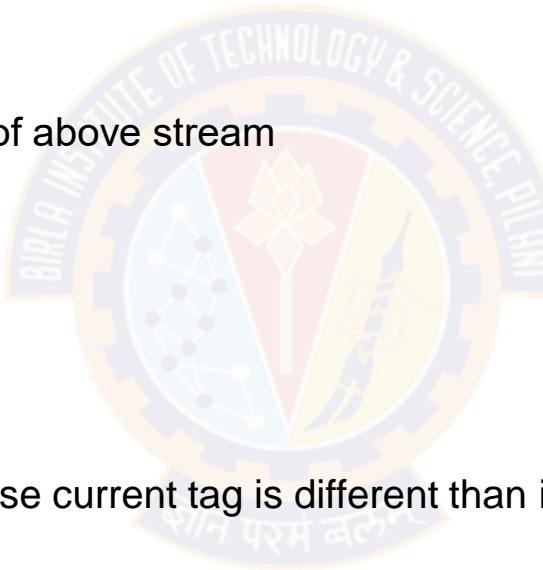
- Stream ---→ fifa, ipl, fifa, ipl, ipl, ipl, fifa
- Weight → 1
- C → 0.1
- The aggregate sum of tag “fifa” in the end of above stream
- **Fifa**
- $fifa - 1 * (1-0.1) = 0.9$
- $ipl - 0.9 * (1-0.1) + 0 = 0.81$ (adding 0 because current tag is different than fifa)
- $fifa - 0.81 * (1-0.1) + 1 = 1.729$ (adding 1 because current tag is fifa only)
- $ipl - 1.729 * (1-0.1) + 0 = 1.5561$
- $ipl - 1.5561 * (1-0.1) + 0 = 1.4005$
- $ipl - 1.4005 * (1-0.1) + 0 = 1.2605$
- $fifa - 1.2605 * (1-0.1) + 1 = \mathbf{2.135}$



Decaying Window Example(3)

The aggregate sum calculation for “ipl”

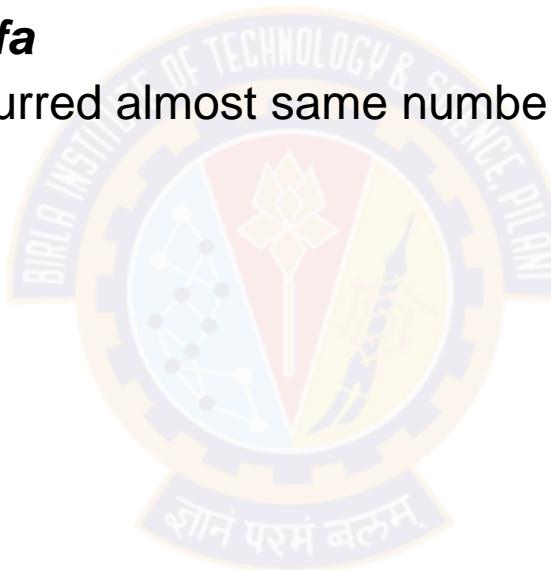
- Stream ---> fifa, ipl, fifa, ipl, ipl, ipl, fifa
- Weight -> 1
- C -> 0.1
- The aggregate sum of tag “ipl” in the end of above stream
- **ipl**
- $\text{fifa} - 0 * (1-0.1) = 0$
- $\text{ipl} - 0 * (1-0.1) + 1 = 1$
- $\text{fifa} - 1 * (1-0.1) + 0 = 0.9$ (adding 0 because current tag is different than ipl)
- $\text{ipl} - 0.9 * (1-0.1) + 1 = 1.81$
- $\text{ipl} - 1.81 * (1-0.1) + 1 = 2.7919$
- $\text{ipl} - 2.7919 * (1-0.1) + 1 = 3.764$
- $\text{fifa} - 3.764 * (1-0.1) + 0 = \mathbf{3.7264}$



Decaying Window Example(4)

Result

- In the end of the sequence
 - ✓ The score of **fifa** is 2.135 but **ipl** is 3.7264
 - ✓ **So ipl** is more trending than **fifa**
 - ✓ Even though both of them occurred almost same number of times in input there score is still different.

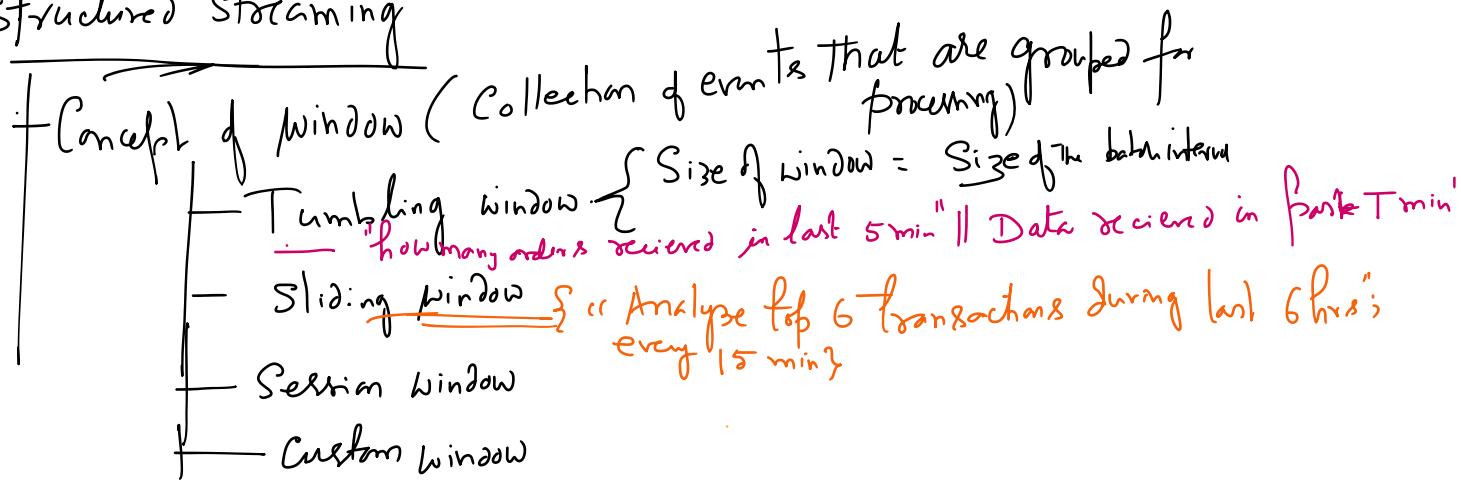




Thank You!

In our next session: Sampling Algorithm

Structured Streaming



Sliding window

- how much data you want to analyze {Size of batch} [Batch interval]
- how frequently you analyse {Size of window} Sliding interval

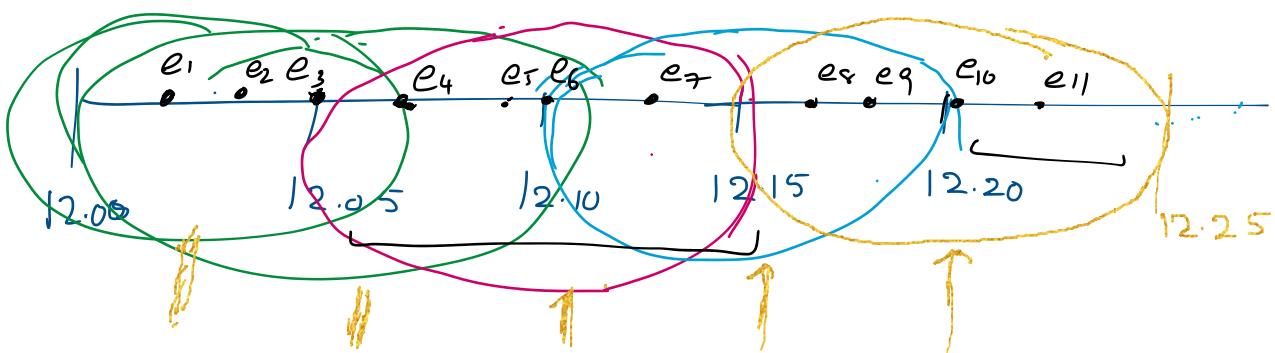
Sliding interval → to trigger processing

Batch interval → how much past data you require.

Tumbling window → window sets are disjoint

$$\boxed{e_1 \dots e_{i-1}}_{w_1} \quad \boxed{e_2 \dots e_i}_{w_2} \quad S_{w_1} \cap S_{w_2} = \emptyset$$

Sliding window can have common elements/events
Sliding interval: 5 min; Batch interval 10M



$$W_{[12.05]} = \{e_1, e_2, e_3\}$$

$$W_{[12.15]} = \{e_3, e_4, e_5, e_6, e_7\}$$

$$W_{[12.10]} = \{e_4, e_5, e_6, e_7, e_8, e_9\}$$

$$W_{[12.15]} = \{e_7, e_8, e_9, e_{10}, e_{11}\}$$

$$W_{12.10} = \{e_1, e_2, e_3, e_4, e_5, e_6\} \quad | \quad W_{12.20} = \{e_6, e_7, e_8, e_9, e_{10}\}$$

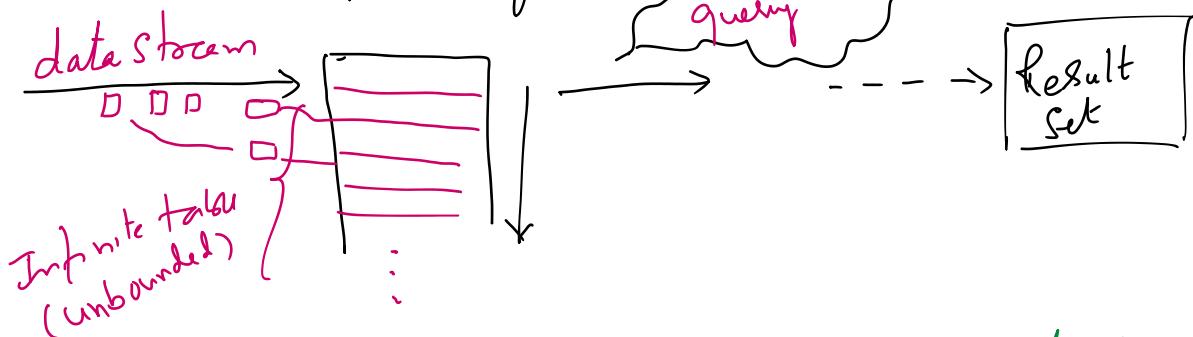
$$W_{12.10} \cap W_{12.15} \neq \emptyset = \{e_3, e_5, e_6, e_7\}$$

Trigger at 12.05 || * moving average calculation
 → sliding window - stampfire

Water Marking

① Data processing in-memory

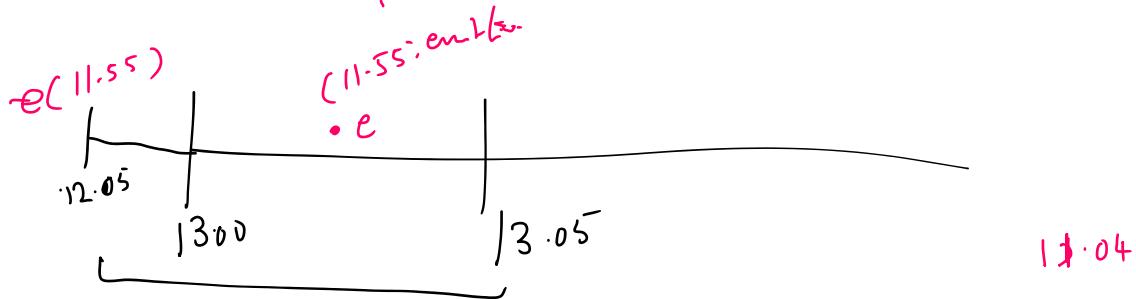
how window processing happens?



State information in Spark is based on aggregation being performed
 — especially append mode()

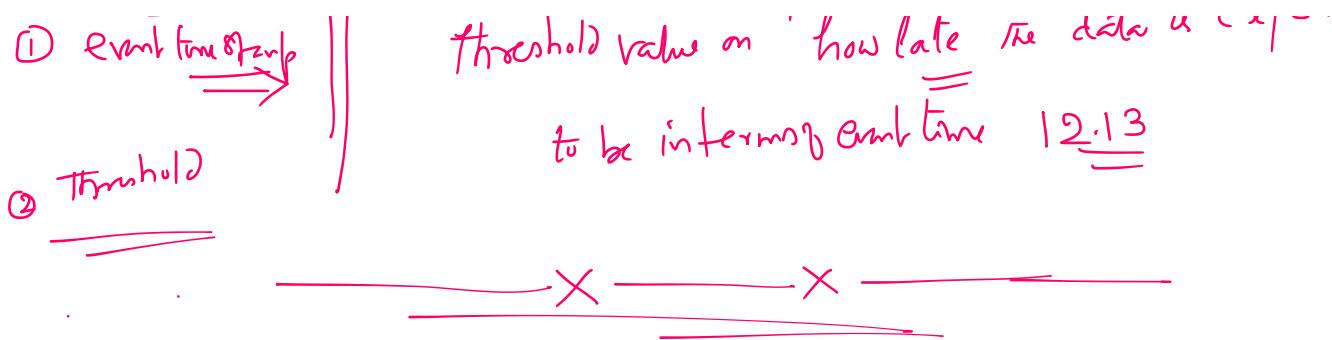
(water) marker strategy: "Data older than marker threshold is being ignored
 — Not being considered."

Let water marking of 1 hr duration



A watermark of a query is specified using "event time"

① Event time stamp || threshold value on + how late the data is expected

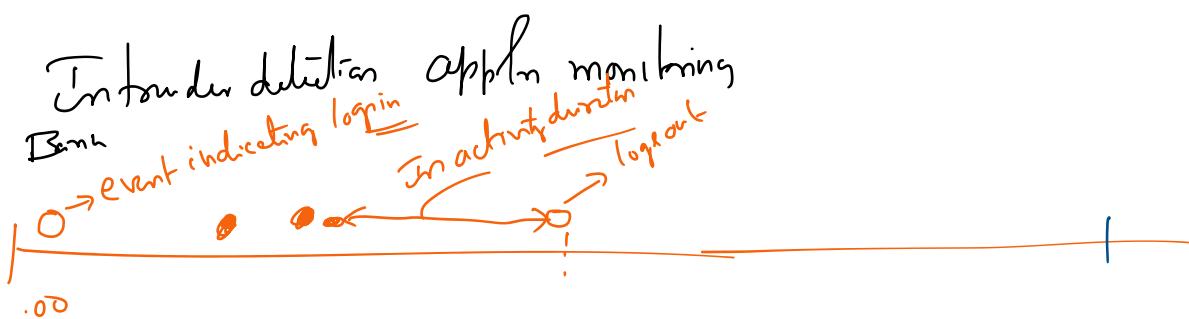


Session window

Specify - < inactivity interval >

There is no fixed batch interval

\Rightarrow A Session is marked based on some activity ;
 Gap (duration) within the activity is called
 " inactivity duration "



Keep accumulating events until the idle time on the threshold
 (4min)

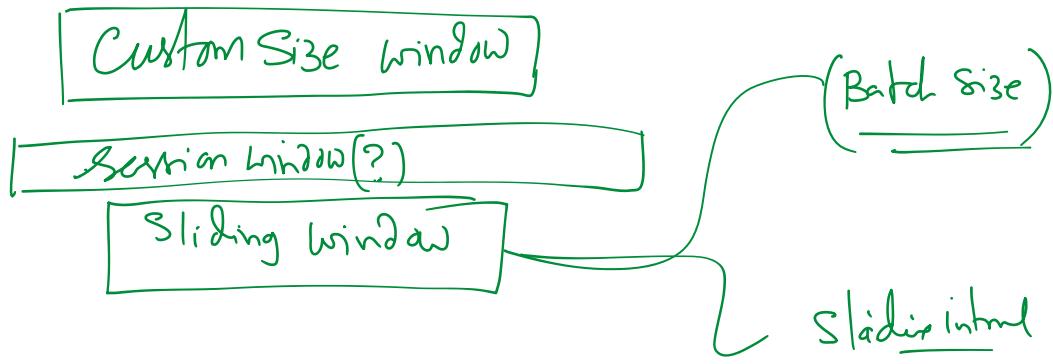
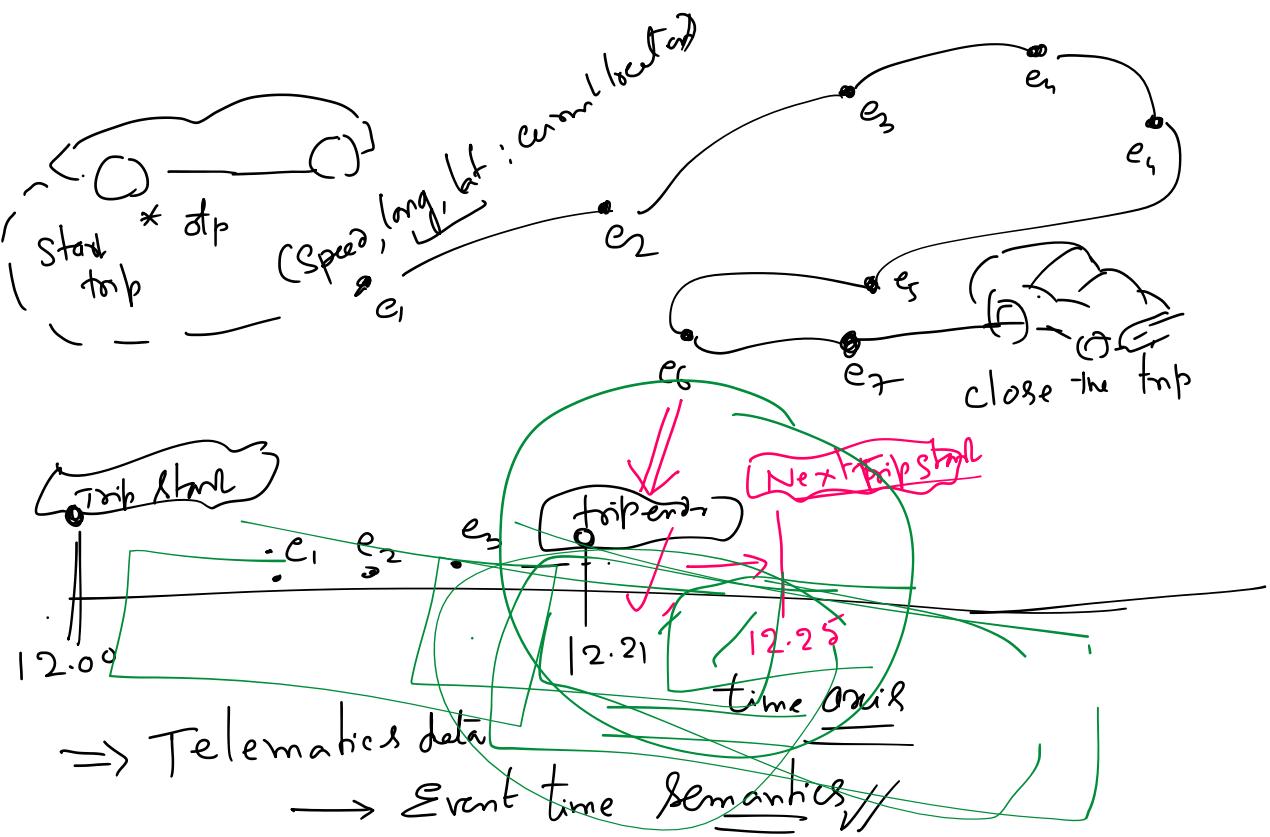
Sessions are unpredictable periods of events that occur such that
 the time gap between the successive events \leq threshold

Custom size window

Customize size of the window

- ① Based on occurrence of certain event, window will start
- ② or gets closed due to occurrence of other event

$$\{e_1, e_2, e_3, \dots, e_7\}$$



Processing of data & formats

Agenda

- Memberships
- Cardinality
- Popularity (most frequent) {Compute freq. of events}

① scanning algorithms are non-deterministic
 Linear search on sorted list is deterministic

```
for (i = 0; i < N; i++) {
    if (lst[i] == key) {
        return i;
    }
}
return -1;
```

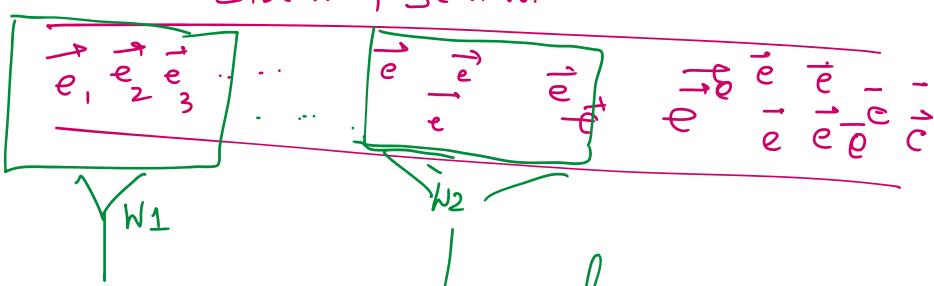
Captions all elements from the data

There is no sampling heuristic

```
if (key in lst)
    ↓
    true
```

```
for i in range(len(lst)):
    if (lst[i] == key):
        return i
return -1
```

Scanning Scenario:



Non-deterministic because next event D. is possibly 1 unit

triggered by policy
Eviction policy
n ... event

Non deterministic because each event e_i is equally likely

Predictive policy

Capturing of an event

To be part of window

window date ~ β_{ad}

Window can't guarantee

each event $\in W$

each event $\notin W$

○ you either enough sampling

\equiv

Strategy (Wavre)

list: [31, 35, 40, 45, 50]

Size of list = $\text{len}(\text{list}) \times \text{size}(\text{int})$

$5 \times 4 \text{ bytes} = 20 \text{ bytes}$

$\approx 10^6$ [$\because 1 \text{ MB} = \frac{1024 \text{ KB}}{1024 \text{ KB}} = 1024 \text{ KB} \approx 10^3 \cdot 10^3 \cdot \text{bytes} = 10^6$]

Model the storage in terms of a bit vector instead of a list of integers

8 bytes

32 bits

$b_1 \ b_2 \ b_3 \ b_4 \ b_5 \ b_6 \ b_7 \ b_8 \dots b_{32}$

$= 2^5 \text{ bits}$

{5, 7, 9, 32}

$4 \times \text{size}(\text{int}) = 4 \times 4 = 16 \text{ bytes}$

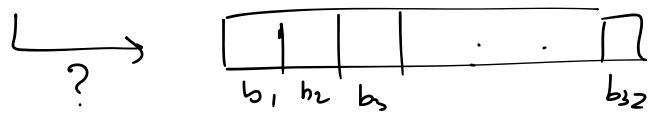
$= 16 \times 8 = 128 \text{ bits}$
 $= (2^7)^2$

Using bit scheme storage space can be reduced exponentially

{46, 57, 92, 105}

$1 \quad | \quad | \quad | \quad n$

46, 57, 92, 105 ↗



We are hashing

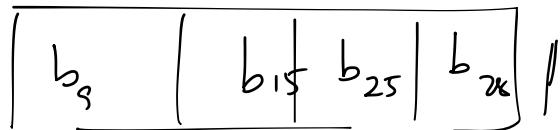
$$k \% 32 (+1)$$

$$57 \% 32 = b_{25}$$

$$92 \% 32 = b_{28}$$

$$105 \% 32 = b_9$$

$$\frac{14+1}{15} = b_{15}$$



Increase number of hash fun

$$h_1(k_i) = i \quad \text{if } \boxed{0 \mid 1 \mid 0 \mid 1 \mid 0} \quad \boxed{1 \mid 0}$$

$$h_2(k_j) = j \quad \text{if } \boxed{0 \mid 1 \mid 0 \mid 1 \mid 0} \quad \boxed{1 \mid 0}$$

$$h_3(k_l) = l$$

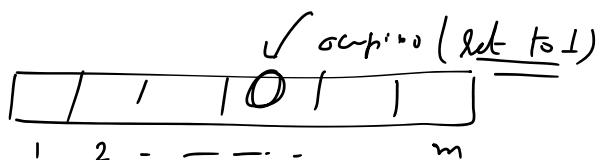
$$(P \text{ \& } B) = 1$$

Bloom filter - window design

Let The filter size = m

Let the number of hash functions = k

Let no. of elements in the filter = n (at a given point in time)



The probability that a bit is zero after 1 insertion using hash function h_1 ,

$$= \left(1 - \frac{1}{m}\right)$$

The probability that a bit is still 0 after 1 insertion using hash fn h_2

$$= \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{1}{m}\right)$$

$$= \left(1 - \frac{1}{m}\right) \cdot \left(1 - \frac{1}{m}\right)$$

↓
Due to h_1

↓
Due to h_2

$$= \left(1 - \frac{1}{m}\right)^2$$

\therefore The prob. That a bit is still 0 after 1 inlation using all hash fun.

$$= \left(1 - \frac{1}{m}\right)^k \quad \Rightarrow m=1$$

The prob. that a bit is still 0 after 2 inlations using all k hash funs:

$$\begin{aligned} & \left(1 - \frac{1}{m}\right)^k \cdot \left(1 - \frac{1}{m}\right)^k \\ & \quad \downarrow \qquad \qquad \downarrow \\ & \quad n=1 \qquad \qquad n=2 \\ & = \left\{ \left(1 - \frac{1}{m}\right)^k \right\}^n \quad \text{with } n=2 \\ & = \left(1 - \frac{1}{m}\right)^{2k} \end{aligned}$$

Prob. that a cell is still 0 after n inlations
using k -hash funs

$$= \left(1 - \frac{1}{m}\right)^{nk}$$

False positive rate: filter returns incorrectly

The prob. of event, incorrectly

$$\boxed{\text{FPR} = 1 - \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)}$$

$$1 - \left(1 - \frac{1}{m}\right)^{nk} = 1 - \sum_{r=1}^{nk} (-1)^r \cdot \binom{nk}{r} \cdot \left(\frac{1}{m}\right)^r \quad \left\{ \because (1-x)^M \sum_{r=0}^M (-1)^r M_r x^r \right\}$$

$$1 - \left(1 - \frac{1}{m}\right)^{nk} = 1 - \sum_{r=0}^{nk} (-1)^r \cdot \binom{nk}{r} \cdot \left(\frac{1}{m}\right)^r \quad \left\{ \because (1-x)^M \sum_{r=0}^M (-1)^r \binom{M}{r} x^r \right\}$$

= 1 -

$$\stackrel{\text{Nole}}{=} (1-x)^N = 1 + \binom{N}{1} (1)^{N-1} \cdot (-x)^1 + \binom{N}{2} (1)^{N-2} \cdot (-x)^2 + \dots + \binom{N}{r} (1)^r (-x)^r + \dots$$

$$= \sum_{r=0}^N \binom{N}{r} (-x)^r = \sum_{r=0}^N \binom{N}{r} (-1)^r \cdot x^r = \underbrace{\sum_{r=0}^N}_{r=i} \binom{N}{r} x^r$$

$$\therefore FPR = 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \binom{nk}{r} \cdot \left(\frac{1}{m}\right)^r \right\} \quad \binom{N}{r} = \frac{\underbrace{(N)}_{(N-r)}!}{\underbrace{(N-r)!}_r}$$

$$= 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \frac{\binom{(nk)}}{\binom{(nk-r)}} \cdot \left(\frac{1}{m}\right)^r \right\}$$

$$= 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \frac{(nk)(nk-1)(nk-2) \dots (nk-r+1)}{\cancel{(nk-r)} \cdot \cancel{r}} \cdot \frac{1}{m^r} \right\}$$

$$= 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \frac{(nk)(nk-1)(nk-2) \dots (nk-(r-1))}{\cancel{r} \cdot \cancel{(nk-r)}} \cdot \frac{1}{m^r} \right\}$$

$$n \sim 10^6; \quad nk \approx nk-1 \\ r \sim \frac{n}{10^6}$$

$$= 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \frac{(nk)(nk)(nk) \dots (nk)}{\cancel{r}} \cdot \frac{1}{m^r} \right\}$$

$$= 1 - \left\{ \sum_{r=0}^{nk} (-1)^r \cdot \frac{(nk)^r}{\cancel{r}} \cdot \frac{1}{m^r} \right\}$$

$$\approx 1 - \left\{ \sum_{r=0}^{\infty} (-1)^r \left(\frac{n_k}{m} \right)^r \cdot \frac{1}{r!} \right\}$$
$$e^{-x} = \sum_{r=0}^{\infty} \frac{(-1)^r}{r!} x^r$$

FPR due to all k hash functions

$$FPR = \left(1 - e^{-\frac{n_k}{m}} \right)^k$$

Given a filter capacity $\rightarrow 3 \text{ MB}$,

filter ended up storing 10^7 elements (events)

wrong 2 hash functions.

Estimate FPR.

$$FPR = \left(1 - e^{-\frac{n_k}{m}} \right)^k$$

$n = 10^7$

$m = 3 \text{ MB}$
 $= 3 \times 10^6 \text{ B}$
 $= 3 \times 10^6 (8 \text{ bits})$

$$m = \underline{24 \times 10^6} \text{ bits}$$

$$R = 2$$

$$\frac{n_k}{m} = \frac{10^7 \cdot 2}{24 \times 10^6} = \frac{20}{24} = \frac{5}{6}$$

$$FPR = \left(1 - e^{-\frac{5}{6}} \right)^2 = \underline{0.31} \approx 31\%$$

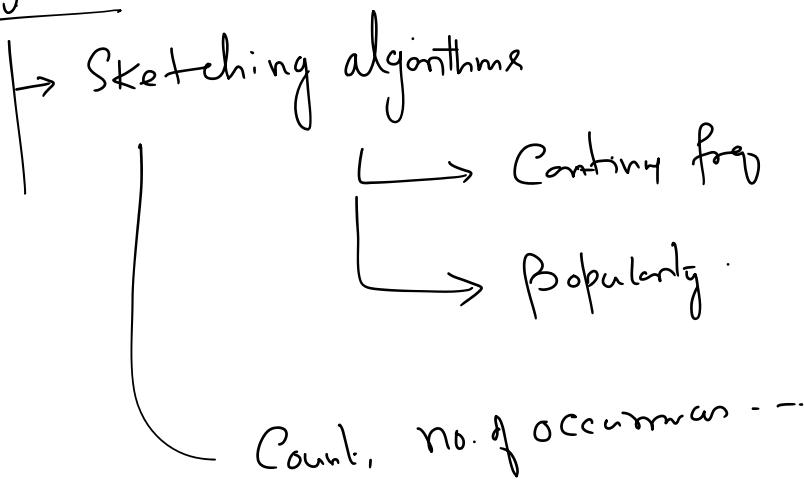
optimal no. of hash fun

$$k_{opt} \Rightarrow -e^{\frac{nk_{opt}}{m}} = \frac{1}{2}$$

$$-\frac{nk_{opt}}{m} \log_2 e = \log_2 \frac{1}{2} \\ = -1$$

✓ $k_{opt} = \boxed{\frac{m}{n} \log_e 2}$ ~~= 0~~

Agenda



Hyperloglog

1. $e \rightarrow$ assign ID
 2. Compute $v = h(\text{ID})$
 3. bit_{log} = Convert2Binary(v) # bit_{log}
 4. last_n_bits = extractnbits(bit_{log})
 5. $d = \text{Convert2Decimal}(\text{last_n_bits})$
 6. Hash_Table [d] ++
- || folding
-
- Count min Sketch (CMS) algorithm

CMS table

No. of rows = m

No. of columns = k (No. of distinct hash functions)
with modulo \underline{m}

$$h(x) = f(x) \% m.$$

$$\underline{h(x)} = f(x) \% m.$$

$$h(x) \in \{0, 1, \dots, m-1\}$$

Let Row index start from 1. The no. of rows

is governed by total number of hashed values

R₀	$h_1(x)$	$h_2(x)$	\dots	$h_k(x)$
R_1				
R_2				
\vdots				
R_{m-1}				

$$a_{ij} \in \text{CMS} = \frac{1}{s.t.} h_j(\text{str}) = i$$

$$h_1(x) = x \% 5, h_2 = (2x+1) \% 5 \rightarrow$$

$$x = \underline{\underline{3}} \quad h_3 = (2^x+1) \% 5$$

$$h_1(3) = 3 \% 5 = \boxed{3} \rightarrow a_{31} = 1$$

Column index of matrix element \swarrow Row index of matrix element

$$h_2(3) = (2 \cdot 3 + 1) \% 5 = 7 \% 5 = 2 \rightarrow a_{22} = 1$$

$$h_3(3) = (3^2 + 1) \% 5 = 10 \% 5 = \cancel{0} \quad \begin{matrix} R_1 \\ \vdots \\ R_{m-1} \end{matrix}$$

Once table is updated with count of events from window

Sketch(key-str, CMS):
entries = []

$n = \text{CMS. columns}$

for ℓ in range($1, n+1$):

$O(n \cdot \ell(h(x)))$

$$r = h[\ell](\text{key}_k) \rightarrow$$

entris.append(CMS[r][\ell])

return $\min\{\text{entris}\}$

$$\begin{aligned} s_1 &= "eat" & s_2 &= \underline{\text{ate}} & s_3 &= "tea" \\ \downarrow & & \downarrow & & & \\ \text{let } \ell &\rightarrow 69, 60, 95 \rightarrow \text{ascii values} & & & & \end{aligned}$$

Suppose:

$$h(x) = \sum_i \text{Ascii}(s_i) \% m$$
$$(95 + 69 + 60) \% m$$
$$60 + 95 + 69 \% m$$

$$\sum_i \text{Ascii}(s_i) \% m$$
$$\begin{array}{c} e \ a \ t \\ 2 \ 1 \ 0 \end{array} \rightarrow (69)^0 + (60)^1 + (95)^2$$
$$\begin{array}{c} t \ e \ a \\ \hline \end{array} \quad (95)^2 + (69)^1 + (60)^0$$

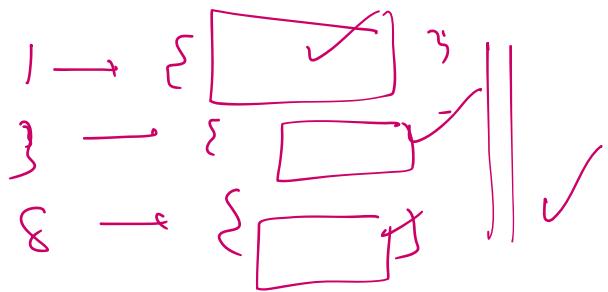
Example

$$h_{1,2}(x) = (2x \pm 1) \bmod 3$$

$$h_{3,4}(x) = (3x \pm 1) \bmod 6$$

$$\text{Seq} = \{1, 3, 3, 8, 1, 3\}$$

(i) Construct CMS table



	$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$
R_1	0	0	0	0
R_2	0	0	0	0
R_3	0	0	0	0

match(1, CMS)

$$\begin{aligned}
 h_1(1) &= (2 \cdot 1 + 1) \% 3 = 0 & \left. \begin{array}{l} \{(1,1), (1,2), (1,3), (2,4)\} \\ = \{(1,2), (2,4)\} \end{array} \right\} \\
 h_2(1) &= (2 \cdot 1 - 1) \% 3 = 1 \\
 h_3(1) &= (3 \cdot 1 + 1) \% 6 = 4 \\
 h_4(1) &= (3 \cdot 1 - 1) \% 6 = 2
 \end{aligned}$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \checkmark \xrightarrow{\text{adj}} \alpha^{11}$$

$$h_1(3) = (2 \cdot 3 + 1) \% 3 = \boxed{1} \quad \left. \begin{array}{l} \{(1,1), (2,2), (4,3), (2,4)\} \\ = \{(1,1), (2,2), (2,4)\} \end{array} \right\}$$

$$h_2(3) = (2 \cdot 3 - 1) \% 3 = 2 \\
 h_3(3) = (3 \cdot 3 + 1) \% 6 = 4$$

$$h_4(3) = (3 \cdot 3 - 1) \% 6 = 2$$

$$\begin{pmatrix} 0 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & \boxed{1} & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \boxed{1}$$

$$\left(\begin{array}{cccc} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array} \right) \rightarrow \left(\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{array} \right)$$

$$\left(\begin{array}{cccc} 2 & 1 & 0 & 0 \\ 0 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{array} \right) \quad \left\{ 1, 3, 3 \right\}$$

$$h_1(s) = \cancel{\text{fatt}} (2 \cdot 8 + 1) \% 3 = 2 \quad \left\{ (2, 1), (\cancel{0}, \cancel{2}), (1, 3), (\cancel{5}, \cancel{4}) \right\}$$

$$h_2(s) = (2 \cdot 8 - 1) \% 3 = 6 \quad \left\{ (2, 1), (1, 3) \right\}$$

$$h_3(s) = (3 \cdot 8 + 1) \% 3 = 1$$

$$h_4(s) = (3 \cdot 8 - 1) \% 6 = 5 \quad \left\{ (1, 2), (2, 4) \right\}$$

$$\left\{ 1, 3, 3, 8, 1 \right\} \rightarrow \left(\begin{array}{cccc} 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{array} \right) \rightarrow$$

$$\left(\begin{array}{cccc} 2 & 2 & 1 & 0 \\ 1 & \cancel{2} & 0 & 4 \\ \frac{1}{6} & 0 & 0 & 0 \end{array} \right) \quad \left\{ 1, 3, 3, 8, 1, 3 \right\} : \boxed{(1, 1)} \quad \boxed{(2, 2)} \quad \boxed{(2, 4)}$$

$$\left(\begin{array}{cccc} 3 & 2 & 1 & 0 \\ 1 & 3 & 0 & 5 \\ 0 & 0 & 0 & 0 \end{array} \right)$$

~~strikethrough 3~~ entries = { 3, 3, 5 }

$$\text{Count min fml}(3) = \min \{ 3, 3, 5 \} = \cancel{\frac{3}{1}}$$

$$\text{Count}_{\min}(\{3\}) = \min\{3, 3, 5\} = \cancel{3}$$