UNIT 2:

STREAM PROCESSING:

What is stream processing?

Stream processing is a data management technique that involves ingesting a continuous data stream to quickly analyze, filter, transform or enhance the data in real time. Once processed, the data is passed off to an application, data store or another stream processing engine.

Stream processing services and architectures are growing in popularity because they allow enterprises to combine data feed from various sources. Sources can include transactions, stock feeds, website analytics, connected devices, operational databases, weather reports and other commercial services.

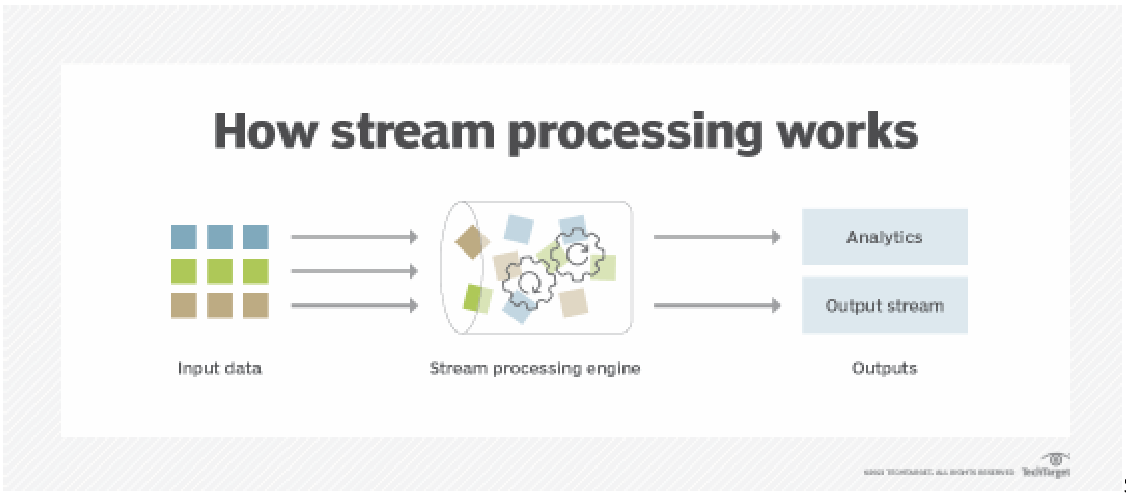
The core ideas behind stream processing have been around for decades but are getting easier to implement with various open source tools and cloud services.

How does stream processing work?

Stream processing architectures help simplify the data management tasks required to consume, process and publish the data securely and reliably. Stream processing starts by ingesting data from a publish-subscribe service, performs an action on it and then publishes the results back to the publish-subscribe service or another data store. These actions can include processes such as analyzing, filtering, transforming, combining or cleaning data.

Stream processing commonly connotes the notion of *real-time analytics*, which is a relative term. *Real time* could mean five minutes for a weather analytics app, millionths of a second for an algorithmic trading app or a billionth of a second for a physics researcher.

However, this notion of real time points to something important about how the stream processing engine packages up bunches of data for different applications. The stream processing engine organizes data events arriving in short batches and presents them to other applications as a continuous feed. This simplifies the logic for application developers combining and recombining data from various sources and from different time scales.


How stream processing works
Analytics
Output stream
Input data
Stream processing engine
Outputs
©2022 TECHTARGET. ALL RIGHTS RESERVED TechTarget
Stream processing allows management of a continuous data stream for use in real time.

Why is stream processing needed?

Stream processing is needed to:

- Develop adaptive and responsive applications
- Help enterprises improve real-time business analytics
- Facilitate faster decisions
- Accelerate decision-making
- Improve decision-making with increased context
- Improve the user experience
- Create new applications that use a wider variety of data sources

How is stream processing used?

Modern stream processing tools are an evolution of various publish-subscribe frameworks that make it easier to process data in transit. Stream processing can reduce data transmission and storage costs by distributing processing across edge computing infrastructure.

Streaming data architectures can also make it easier to integrate data from multiple business applications or operational systems. For example, telecom service providers are using stream processing tools to combine data from numerous operations support systems. Healthcare providers use them to integrate applications that span multiple medical devices, sensors and electronic medical records systems. Stream processing also supports more responsive applications in anomaly detection, trend spotting and root cause analysis.

Common stream processing use cases include:

- Fraud detection
- Detecting anomalous events
- Tuning business application features
- Managing location data
- Personalizing customer experience
- Stock market trading
- Analyzing and responding to IT infrastructure events
- Digital experience monitoring
- Customer journey mapping
- Predictive analytics

What are the stream processing frameworks?

Spark, Flink and Kafka Streams are the most common open source stream processing frameworks. In addition, all the primary cloud services also have native services that simplify stream processing development on their respective platforms, such as Amazon Kinesis, Azure Stream Analytics and Google Cloud Dataflow.

These often go hand in hand with other publish-subscribe frameworks used for connecting applications and data stores. For example, Apache Kafka is a popular open source publish-subscribe framework that simplifies integrating data across multiple applications. Apache Kafka Streams is a stream processing library for creating applications that ingest data from Kafka, process it and then publish the results back to Kafka as a new data source for other applications to consume.

Other stream processing tools with novel capabilities are also growing in popularity. Samza is a distributed stream processing tool that allows users to build stateful applications. Apache Storm supports real-time computation capabilities like online machine learning, reinforcement learning and continuous computation. Delta Lake supports stream processing and batch processing using a common architecture.

What are the differences between stream and batch processing?

Stream processing and batch processing represent two different data management and application development paradigms. Batch processing originated in the days of legacy databases in which data management professionals would schedule batches of updates from a transactional database into a report or business process. Batch processing is suitable for regularly scheduled data processing tasks with well-defined boundaries. It is a good approach for pulling out transactional numbers from the sales database to generate a quarterly report or tallying employee hours to calculate monthly checks.

Stream processing allows developers to think about ingesting data as a continuous data stream. Technically speaking, the data still arrives in batches. Still, the stream processing engine manages the process of filtering out data updates and keeping track of what it has already uploaded into the feed. This frees up more time for data engineering and developer teams to code the analytics and application logic.

History of stream processing

Computer scientists have explored various frameworks for processing and analyzing data across multiple days since the dawn of computers. In the early days, this was called sensor fusion. Then, in the early 1990s, Stanford University professor David Luckham coined the term *complex event processing* (CEP). This helped fuel the development of service-oriented architectures (SOAs) and enterprise service busses (ESBs).

CEP's fundamental principles included abstractions for characterizing the synchronous timing of events, managing hierarchies of events and considering the causal aspects of events. The rise of cloud services and open source software led to more cost-effective approaches for managing event data streams, using publish-subscribe services built on Kafka.

This gave rise to stream processing frameworks that simplified the cost and complexity of correlating data streams into complex events. With the rise of the cloud, the industry is starting to move away from the terms *SOA*, *ESB* and *CEP* toward infrastructure built on microservices, publish-subscribe services and stream processing. Although the terms are different, the core idea inherent in these older technologies lives on.

MINING DATA STREAM:

Introduction to stream concepts :
A data stream is an existing, continuous, ordered (implicitly by entrance time or explicitly by timestamp) chain of items. It is unfeasible to control the order in which units arrive, nor it is feasible to locally capture stream in its entirety.

It is enormous volumes of data, items arrive at a high rate.

Types of Data Streams :

- **Data stream –**

A data stream is a(possibly unchained) sequence of tuples. Each tuple comprised of a set of attributes, similar to a row in a database table.

- **Transactional data stream –**

It is a log interconnection between entities

1. Credit card – purchases by consumers from producer
2. Telecommunications – phone calls by callers to the dialed parties
3. Web – accesses by clients of information at servers

- **Measurement data streams –**

1. Sensor Networks – a physical natural phenomenon, road traffic
2. IP Network – traffic at router interfaces

3. Earth climate – temperature, humidity level at weather stations

Examples of Stream Sources-

1. **Sensor Data –**
   In navigation systems, sensor data is used. Imagine a temperature sensor floating about in the ocean, sending back to the base station a reading of the surface temperature each hour. The data generated by this sensor is a stream of real numbers. We have 3.5 terabytes arriving every day and we for sure need to think about what we can be kept continuing and what can only be archived.

2. **Image Data –**
   Satellites frequently send down-to-earth streams containing many terabytes of images per day. Surveillance cameras generate images with lower resolution than satellites, but there can be numerous of them, each producing a stream of images at a break of 1 second each.

3. **Internet and Web Traffic –**
   A bobbing node in the center of the internet receives streams of IP packets from many inputs and paths them to its outputs. Websites receive streams of heterogeneous types. For example, Google receives a hundred million search queries per day.

Characteristics of Data Streams :

1. Large volumes of continuous data, possibly infinite.
2. Steady changing and requires a fast, real-time response.
3. Data stream captures nicely our data processing needs of today.
4. Random access is expensive and a single scan algorithm
5. Store only the summary of the data seen so far.
6. Maximum stream data are at a pretty low level or multidimensional in creation, needs multilevel and multidimensional treatment.

Applications of Data Streams :

1. Fraud perception
2. Real-time goods dealing
3. Consumer enterprise
4. Observing and describing on inside IT systems

Advantages of Data Streams :

- This data is helpful in upgrading sales
- Help in recognizing the fallacy
- Helps in minimizing costs
- It provides details to react swiftly to risk

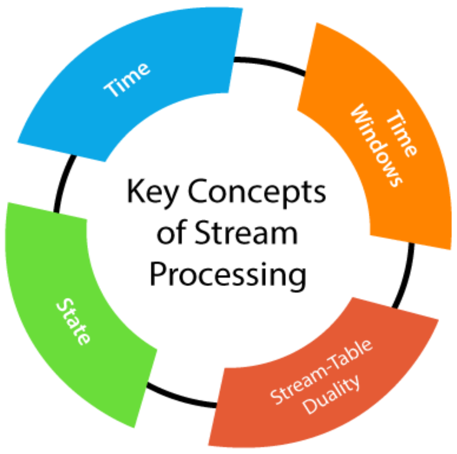
Disadvantages of Data Streams :

- Lack of security of data in the cloud
- Hold cloud donor subordination
- Off-premises warehouse of details introduces the probable for disconnection

INTRODUCTION TO STREAM CONCEPTS:

Key concepts of Stream Processing

There are the following concepts that a user should know about stream processing:



Time

It is essential as well as the most confusing concept. In stream processing, most operations rely on time. Therefore, a common notion of time is a typical task for such stream applications.

Kafka Stream processing refers to following notions of time:

1. **Event Time:** The time when an event had occurred, and the record was originally created. Thus, event time matters during the processing of stream data.
2. **Log append time:** It is that point of time when the event arrived for the broker to get stored.
3. **Processing Time:** The time when a stream-processing application received the event to apply some operations. The time can be in milliseconds, days, or hours. Here, different timestamps are assigned to the same event, depending on exactly when each stream processing application happened to read the event. Also, the timestamp can differ for two threads in the same application. Thus, the processing time is highly unreliable, as well as best avoided.

State

There are different states maintained in the stream processing applications.

The states are:

1. **Internal or local state:** The state which can be accessed only by a specific stream-processing application?s instance. The internal state is managed and maintained with an embedded, in-memory database within the application. Although local states are extremely fast, the memory size is limited.
2. **External state:** It is the state which is maintained in an external data store such as a NoSQL database. Unlike the internal state, it provides virtually unlimited memory size. Also, it can be accessed either from different applications or from their instances. But, it carries extra latency and complexity, which makes it avoidable by some applications.

Stream-Table Duality

A Table is a collection of records which is uniquely identified by a primary key. Queries are fired to check the state of data at a specific point of time. Tables do not contain history, specifically unless we design it. On the other hand, streams contain a history of changes. Streams are the strings of events where each event causes a change. Thus, tables and streams are two sides of the same coin. So, to convert a table into streams, the user needs to capture the commands which modify the table. The commands such as insert, update, and delete are captured and stored into streams. Also, if the user wants to convert streams into a table, it is required to convert all changes which a stream contains. This process of conversion is also called **materializing the stream**. So, we can have the dual process of changing streams into tables as well as tables to streams.

Time Windows

The term time windows means windowing the total time into parts. Therefore, there are some operations on streams which depend on the time window. Such operations are called **Windowed operations**. For example, join operation performed on two streams are windowed. Although people rarely care about the type of window they need for their operations.

STREAM DATA MODEL AND ARCHITECTURE:

What is Streaming Data Architecture?

A streaming data architecture is capable of ingesting and processing massive amounts of streaming data from a variety of different sources. While traditional data solutions focused on batch writing and reading, a streaming data architecture consumes data as it is produced, persists it to storage, and may perform real-time processing, data manipulation, and data analysis.

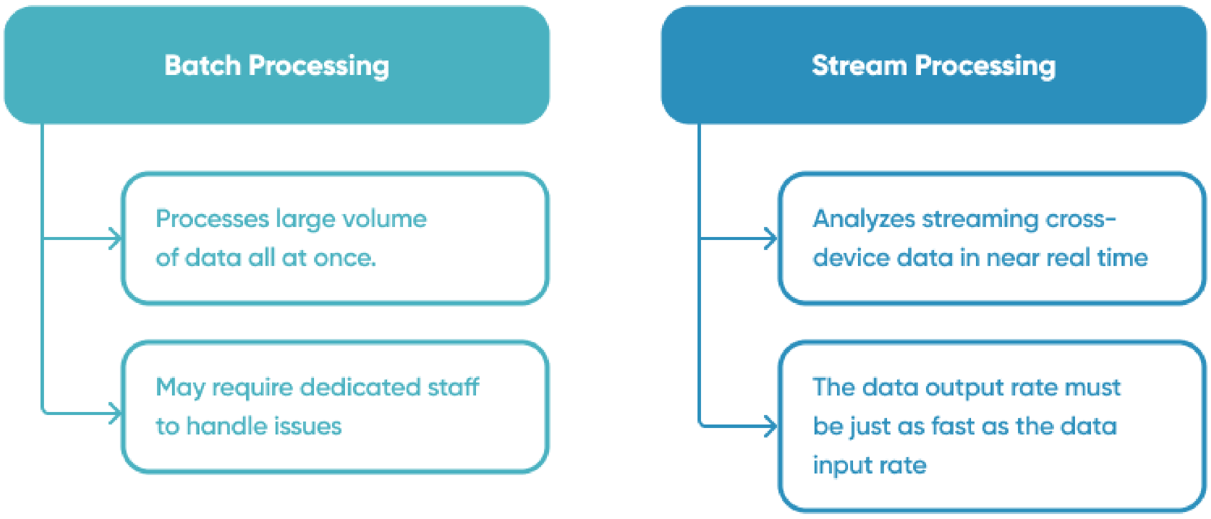
Initially, stream processing was considered a niche technology. Today it's difficult to find a modern business that does not have an app, online advertising, an e-commerce site, or products enabled by the Internet of Things. Each of these digital assets generates real-time event data streams. There is a growing appetite for implementing a streaming data infrastructure that enables complex, powerful, and real-time analytics.

Streaming data architecture helps to develop applications that use both bound and unbound data in new ways. For example, Netflix also uses Kafka streams to support its recommendation engines, combining streamed data and machine learning.

PRO TIP: Data streaming architecture is also referred to as Kappa architecture. It is an alternative to Lambda architecture which separates slow batch processing from fast real-time data access. Data streaming technologies like Apache Kafka or Apache Flink enable near-real-time processing of incoming data events - this approach allows Kappa architecture to combine batch and streaming processing into the same data flow.

Streaming Data Architecture Use Cases

At smaller scales, traditional batch architectures may suffice. However, streaming sources such as sensors, server and security logs, real-time advertising, and clickstream data from apps and websites can generate up to a Gb of events per second.


Batch Processing
Processes large volume of data all at once.
May require dedicated staff to handle issues
Stream Processing
Analyzes streaming cross-device data in near real time
The data output rate must be just as fast as the data input rate

Stream processing is becoming a vital part of many enterprise data infrastructures. For example, companies can use clickstream analytics to track web visitor behavior and tailor their content, while ecommerce historical data analytics can help retailers prevent shopping cart abandonment and show customers more relevant offers. Another common use case is Internet of Things (IoT) data analysis, which involves analyzing large streams of data from sensors and connected devices.

Benefits of data stream processing

Stream processing provides several benefits that other data platforms cannot:

- **Handling never-ending streams of events natively**, reducing the overhead and delay associated with batching events. Batch processing tools need pausing the stream of events, gathering batches of data, and integrating the batches to get a conclusion. While it is difficult to aggregate and capture data from numerous streams in stream processing, it enables you to gain instant insights from massive amounts of streaming data.
- **Processing in real-time or near-real-time for up-to-the-minute data analytics and insight.** For example, dashboards that indicate machine performance, or just-in-time delivery of micro-targeted adverts or support, or detection of fraud or cybersecurity breaches.
- **Detecting patterns in time-series data.** Detection of patterns over time, such as trends in website traffic statistics, requires data to be processed and analyzed continually. This is made more complex by batch processing, which divides data into batches, resulting in certain occurrences being split across two or more batches.
- **Simplified data scalability.** Growing data volumes might overwhelm a batch processing system, necessitating the addition of additional resources or a redesign of the architecture. Modern stream processing infrastructure is hyper-scalable, with a single stream processor capable of processing Gigabytes of data per second.

Developing a streaming architecture is a difficult task that is best accomplished by the addition of software components specific to each use case – hence necessitating the need to "architect" a common solution capable of handling the majority, if not all, envisioned use cases.

Streaming data challenges

Real-time data streaming systems require new technologies and process bottlenecks. The increased complexity of these systems can lead to failure when seemingly innocuous components or processes become slow or stall. Here are the streaming data challenges today's organizations face, and solutions:

Reliance on centralized storage and compute clusters

Modern real-time data streaming technologies, such as Apache Kafka, are designed to support distributed processing and to minimize coupling between producers and consumers. Deployment too tightly tied to one central cluster (i.e. traditional Hadoop stack) can suffocate project and domain autonomy. As a result, streaming adoption and data consumption will be constrained.

Solution

Containerization, which allows for greater flexibility and domain independence, is used in a distributed deployment architecture to achieve this.

Scalability bottlenecks

As data sets grow bigger, operations naturally become a more significant problem. For example, backups take significantly longer and consume a significant amount of resources. Rebuilding indexes, defragmenting storage, and reorganizing historical data are all time-consuming and resource-intensive operations that require significant resources.

Solution

Check the production environment loads. If you test run the expected load of the previous 3 months of the data, you can find and fix problems before going live.

Business integration hiccups

In many cases, the enterprise consists of many lines of business and application teams, each focused on its own mission and challenges. This works for a while until each group needs to integrate and exchange real-time event data streams.

Solution

To federate the events, multiple integration points may be required.

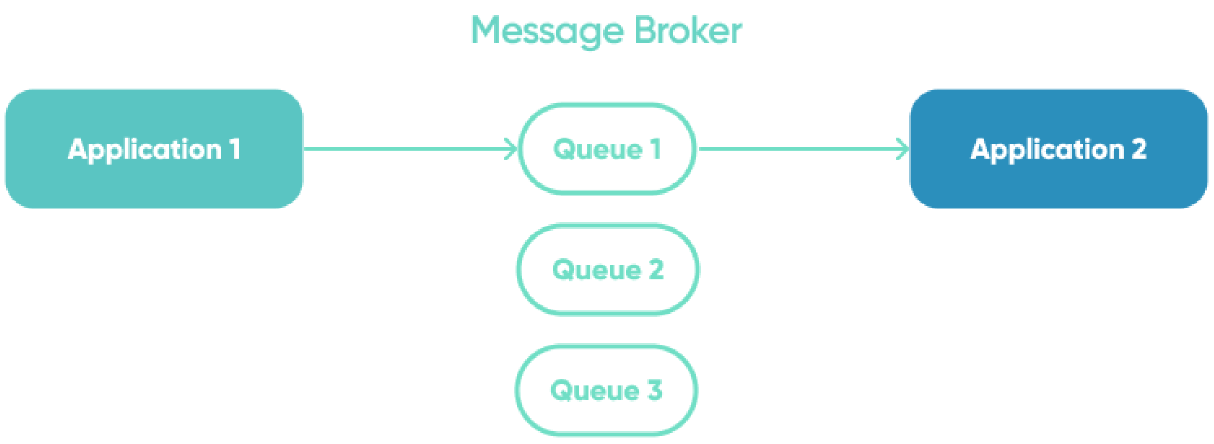
4 Key Components of a Streaming Data Architecture

A streaming data architecture is a set of software components designed to handle large streams of raw data from various sources:

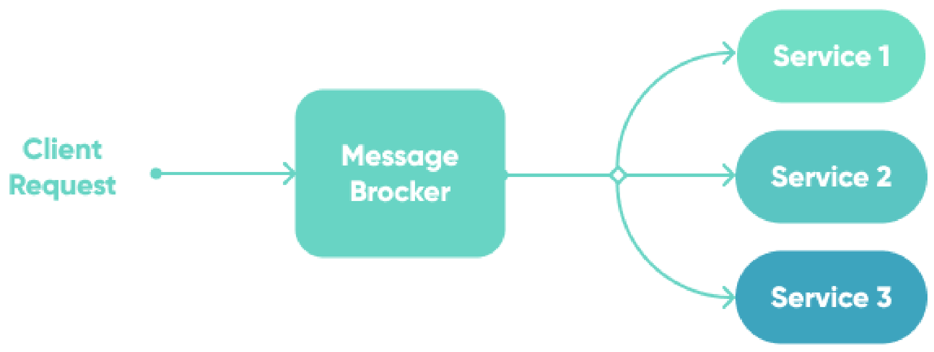
Message Broker (Stream Processor)

The stream processor collects data from its source, converts it to a standard message format, and then streams it continuously for consumption by other components. A storing streaming data component, such as a data warehouse/data lake, an ETL tool, or another type of component are examples of such components. Stream processors have a high throughput, but they don't do any data transformation or task scheduling.

Message Broker can act as a proxy between two applications where communication between them is achieved using queues. In such case we refer to it as point-to-point broker.



If one application is broadcasting a single message to multiple applications, we say that broker acts in Publish/Subscribe model



Popular stream processing tools:

- Apache Kafka
- RabbitMQ

Batch processing and real-time ETL tools

In data-intensive organizations, process streaming data is an essential component of the big data architecture. There are many fully managed frameworks to choose from that all set up an end-to-end streaming data pipeline in the cloud to enable real-time analytics.

Example managed tools:

- Amazon Kinesis Data Streams
- Azure Event Hub
- Google Cloud PubSub

Streaming Data Storage

Organizations typically store their streaming event data in cloud object stores to serve as operational data lake due to the sheer volume and multi-structured nature of event streams. They offer a low-cost and long-term solution for storing large amounts of event data. They're also a flexible integration point, allowing tools from outside your streaming ecosystem to access data.

To learn more, check out Data Lake Consulting services.

Examples:

- Amazon Redshift
- Microsoft Azure Data Lake Storage
- Google Cloud Storage

Data Analytics / Serverless Query Engine

With data processed and stored in a data warehouse/data lake, you will now need data analytics tools.

Examples (not exhaustive):

- **Query engines** – Athena, Presto, Hive, Redshift Spectrum, Pig
- **Text search engines** – Elasticsearch, OpenSearch, Solr, Kusto

- **Streaming data analytics** – Amazon Kinesis, Google Cloud DataFlow, Azure Stream Analytics

Streaming architectures patterns

Streaming architectures patterns help build reliable, scalable, secure applications in the cloud:

Idempotent producer

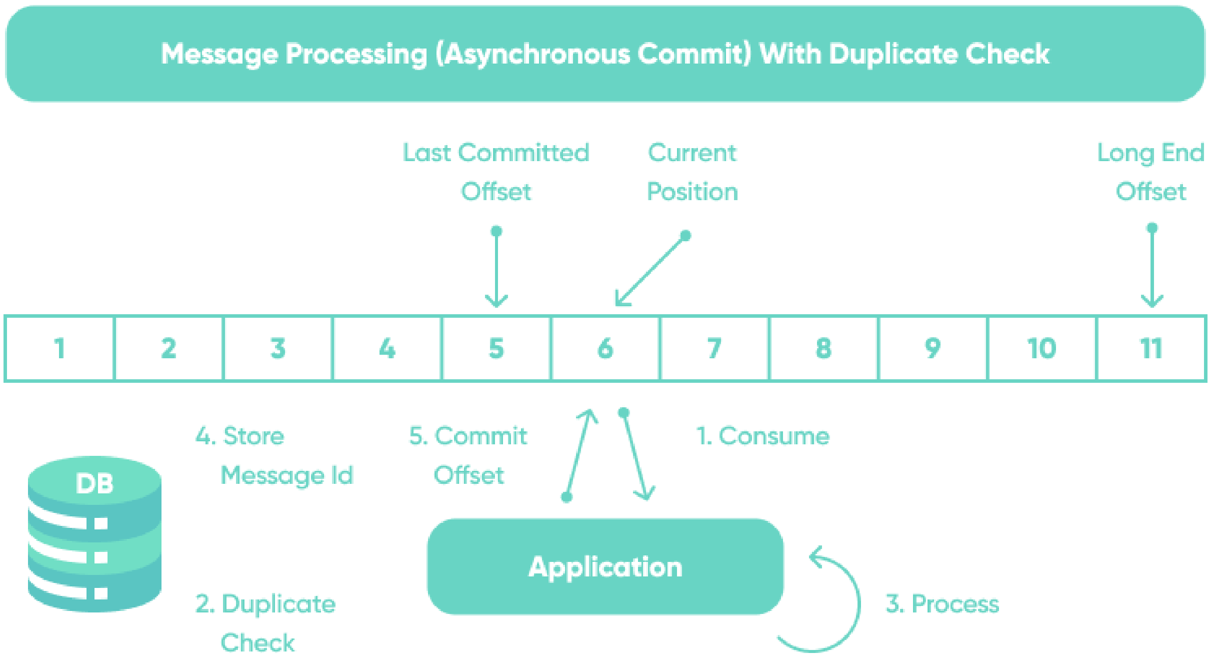
The event streaming platform knows nothing of the business logic so how can you deal with duplicate events when reading from an event stream?

The idempotent producer pattern is most commonly used to deal with duplicated events in an input data stream.

For example in Apache Kafka, with ProducerConfig configuration:

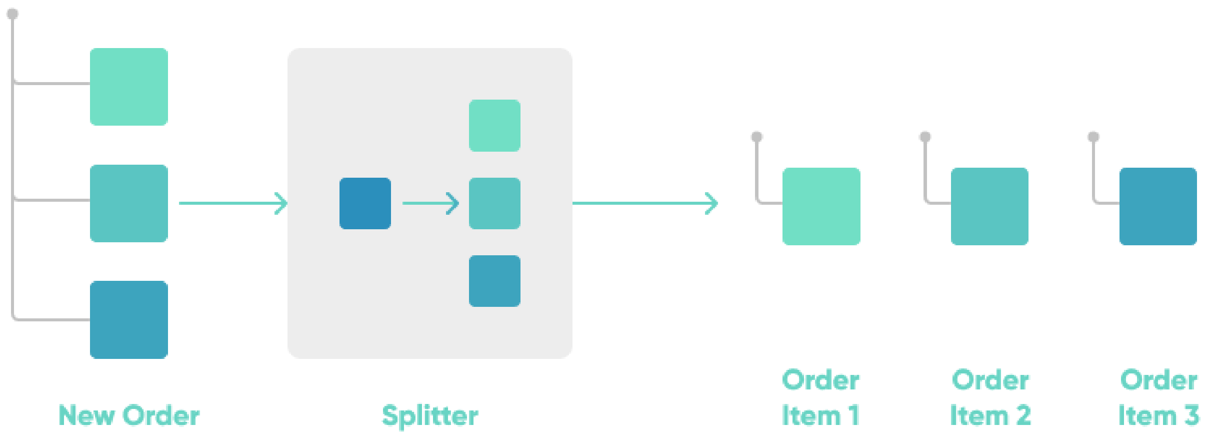
```
enable.idempotence=true

acks=all
```

Each producer gets assigned a Producer Id (PID) and it includes its PID every time it sends messages to a broker. Additionally, each message gets a monotonically increasing sequence number.



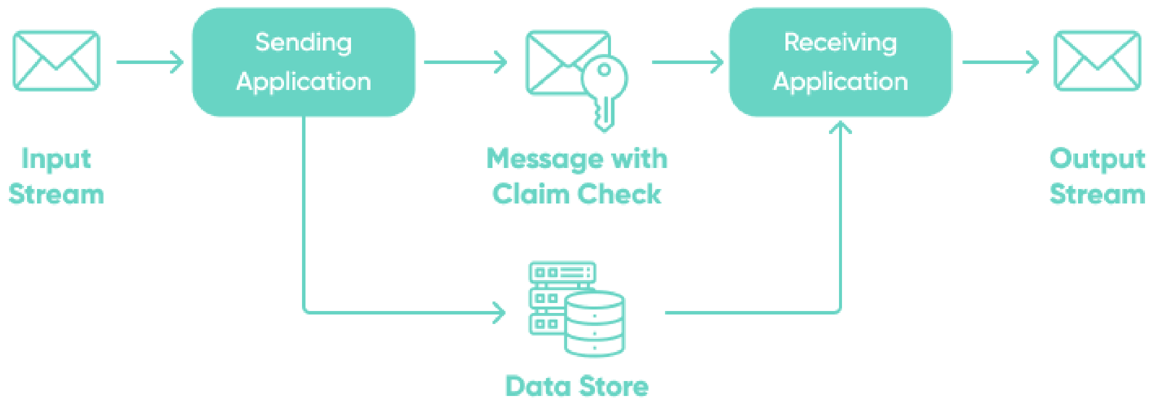
Event splitter

Many data sources produce messages that consist of multiple elements. The event splitter pattern can be used used to split a business event into multiple events. For example, e-commerce order events can be split into multiple events per order item (for analytics purposes).



Claim-check pattern

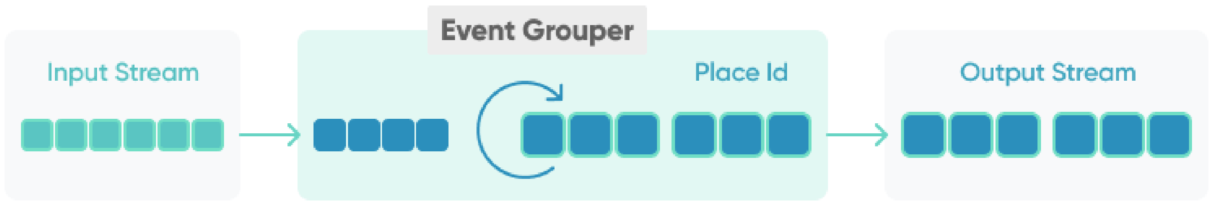
Often a messaging-based architecture must be capable of sending, receiving, and manipulating large messages. These use cases can be related to image recognition, video processing, etc. Sending such large messages to the message bus directly is not recommended. The solution is to send the claim check to the messaging platform and store the payload to an external service.



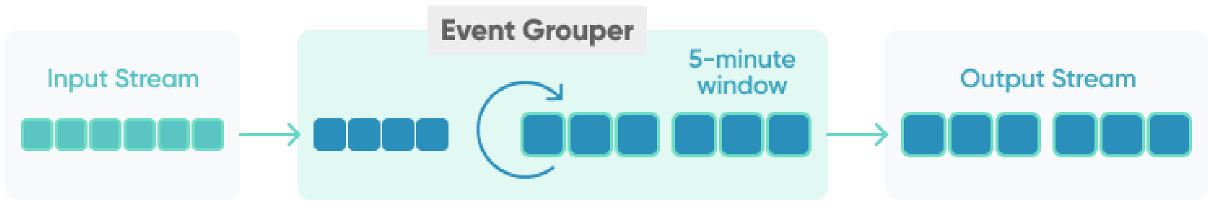
Event grouper

Sometimes Events become significant only after they've happened several times. For example, parcel delivery will be attempted three times before we ask the customer to collect it from the depot. How can we wait for N logically similar events?

The solution is to consider related events as a group. For this, we need to group them by a given key, and then count the occurrences of that key.



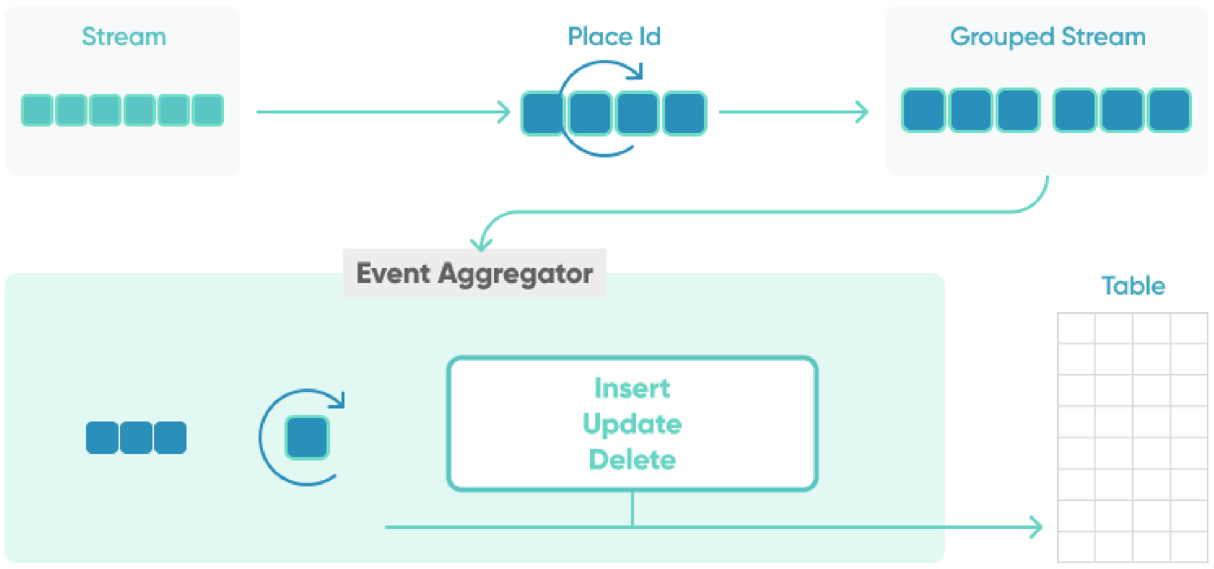
For time-based grouping, we can group related events into created automatically time windows, for example, 5 minutes or 24 hours.



Event aggregator

Combining multiple events into a single encompassing event, calculate the average, median, or percentile on the incoming business data, a common task in event streaming and real-time analytics. How can multiple related events be aggregated to produce a new event?

We can combine event grouper and event aggregator. The grouper prepares the input events as needed for the subsequent aggregation step, e.g. by grouping the events based on the data field by which the aggregation is computed (such as order ID) and/or by grouping the events into time windows (such as 5-minute windows). The aggregator then computes the desired aggregation for each group of events, e.g., by computing the average or sum of each 5-minute window.

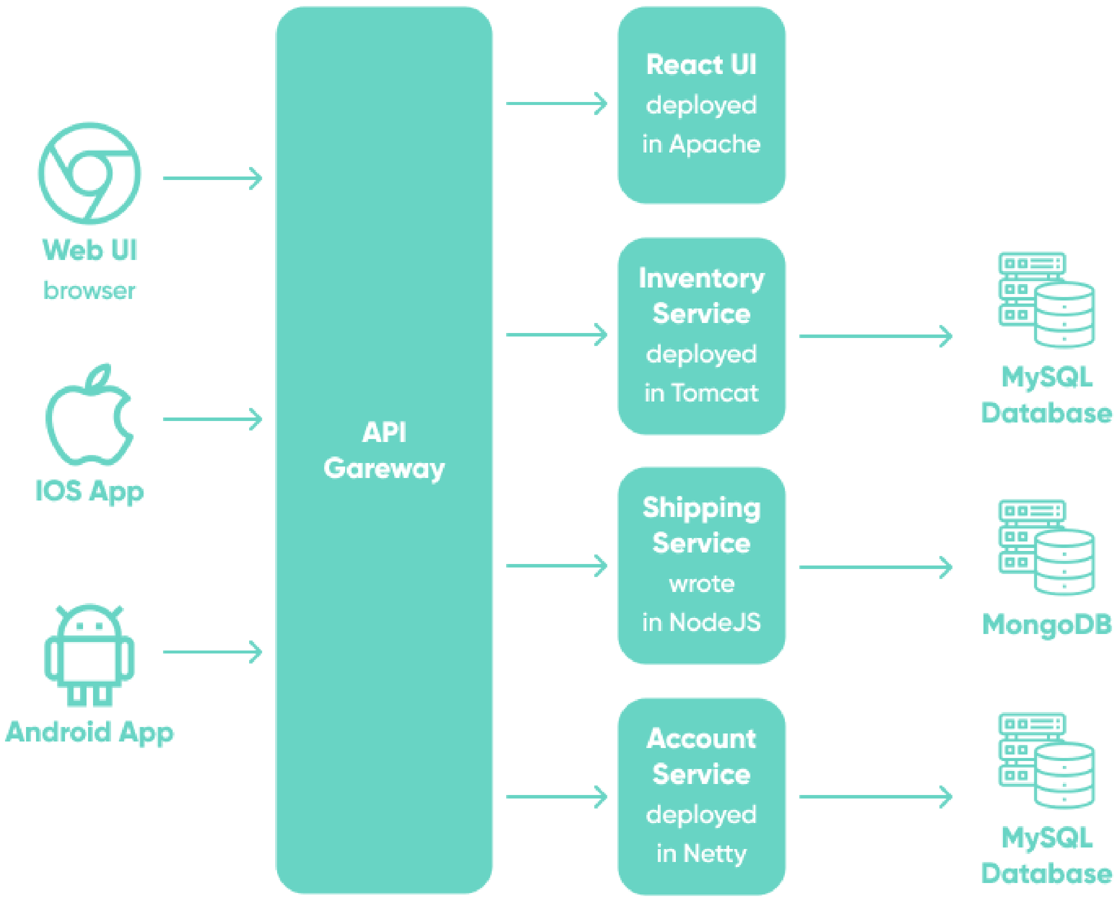


Gateway routing

When a client requires the consumption of multiple services, setting up a separate endpoint for each service and having the client manage each endpoint can be challenging. For example, an e-commerce application might provide services such as search, reviews, cart, checkout, and order history. Each service has a unique API with which the client must communicate, and the client must be aware of all endpoints in order to connect to the services.

If an API is changed, the client must also be updated. When a service is refactored into two or more distinct services, the code in both the service and the client must change.

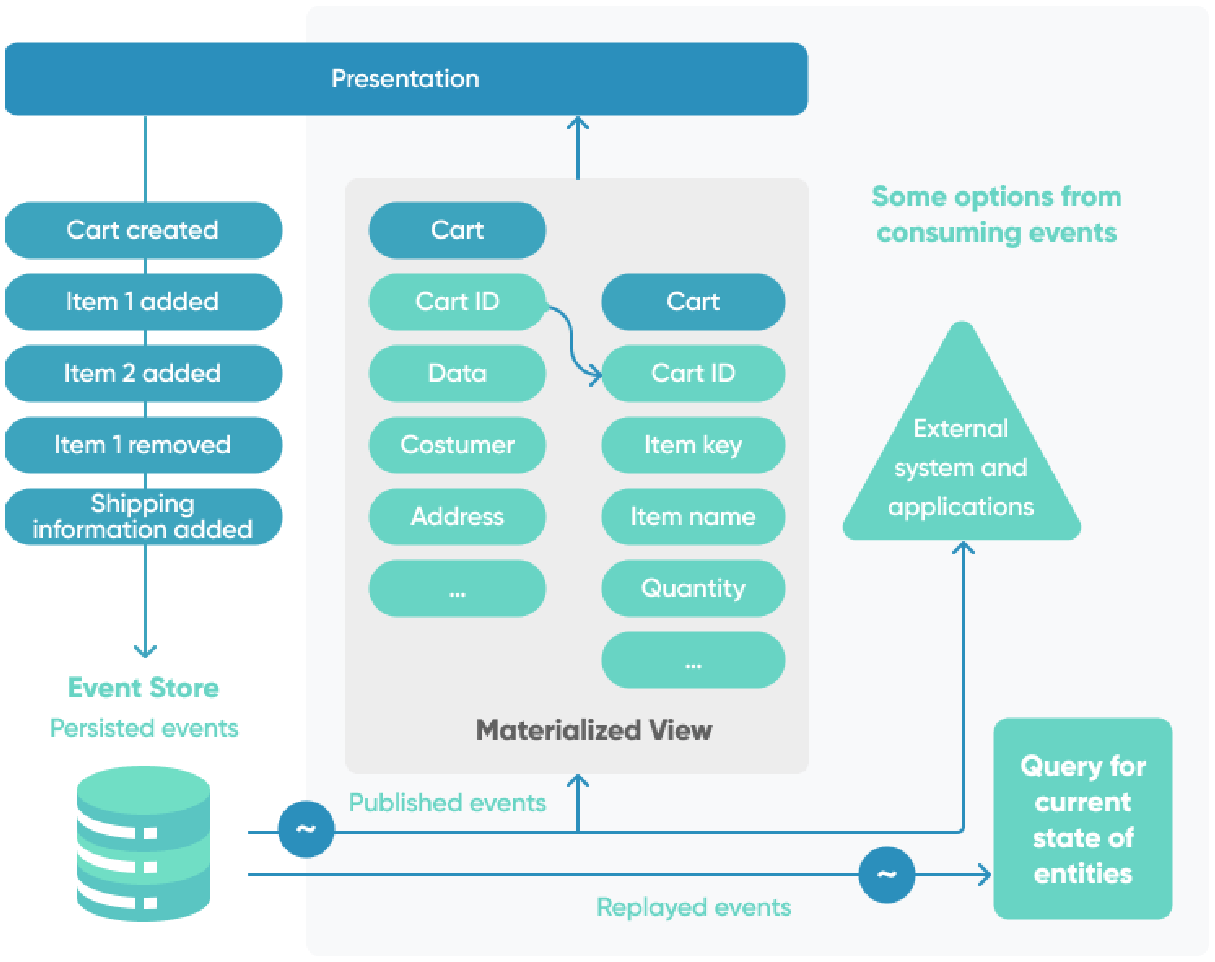
Gateway routing allows the client applications to know about and communicate with a single endpoint.


Web UI
browser
IOS App
Android App
API Gareway
React UI deployed in Apache
Inventory Service deployed in Tomcat
Shipping Service wrote in NodeJS
Account Service deployed in Netty
MySQL Database
MongoDB
MySQL Database

CQRS

It is common in traditional architectures to use the same data model for both querying and updating databases. That's straightforward and effective for basic CRUD operations. However, in more complex applications, this approach can become cumbersome. For example, the application may run a variety of queries, returning data transfer objects of various shapes.

Command and Query Responsibility Segregation (CQRS) is a pattern for separating read and update operations in a data store. CQRS can improve the performance, scalability, and security of your application. Migrating to CQRS gives a system more flexibility over time, and it prevents update commands from causing merge conflicts at the domain level.

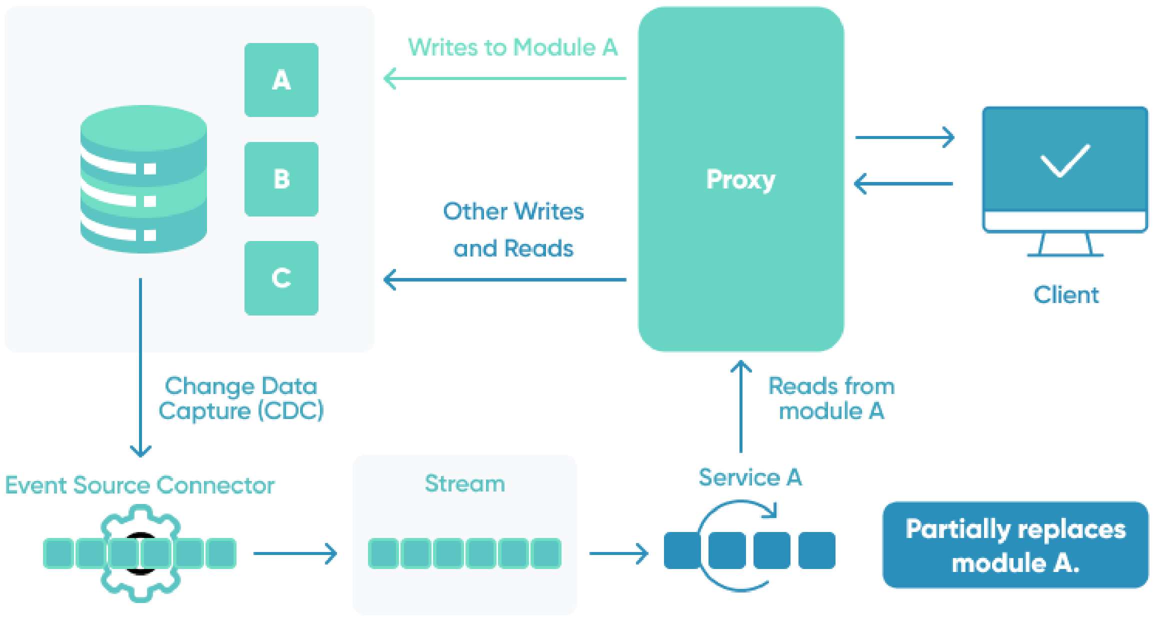


Strangler Fig pattern

As systems age, the development tools, hosting technology, and even system architectures on which they were built can all become obsolete as systems age. The complexity of these applications can increase dramatically as new features and functionality are added, making them more difficult to maintain and add new features to.

Replacing a complex system from the ground up can be a huge undertaking. Often, you will need a gradual migration to a new system, with the old system remaining in place to handle features that have not yet been migrated.

You can use the Strangler Fig pattern to incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services. Eventually, the old system's functions will be completely replaced by the new system, making it possible to decommission the old system.



Conclusion

The growing popularity of streaming data architectures reflects a shift in the modern data platforms development from a monolithic architecture to a decentralized one built with microservices.

This type of architecture is more flexible and scalable than a classic DB-centric application architecture and it factors the time an event occurs into account, which makes it easier for an application's state and processing to be partitioned and distributed across many instances.

STREAM COMPUTING:

What is streaming analytics?

Streaming analytics is the processing and analyzing of data records continuously rather than in batches. Generally, streaming analytics is useful for the types of data sources that send data in small sizes (often in kilobytes) in a continuous flow as the data is generated.

Streaming analytics overview

Streaming analytics may include a wide variety of data sources, such as telemetry from connected devices, log files generated by customers using web applications, ecommerce transactions, or information from social networks or geospatial services. It's often used for real-time aggregation and correlation, filtering, or sampling.

Data traditionally is moved in batches. Batch processing often processes large volumes of data at the same time, with long periods of latency. For example, a process may be run every 24 hours. While this can be an efficient way to handle large volumes of data, it doesn't work with time-sensitive data that's meant to be streamed, because that data can be stale by the time it's processed.

How to optimize streaming analytics

When companies are collecting data to the tune of hundreds of thousands or even millions of events per second, absolutely massive datasets are the result. Traditional systems can take days to deliver insights from data at this scale.

To generate real-time actions, you need real-time data processing and analysis. This can be accomplished with the right data-streaming platform and infrastructure. Stream analytics built on Google Cloud products and services, for example, enable companies to ingest, process, and analyze data streams in real time.

Streaming analytics use cases

Companies use streaming analytics to analyze data in real time and provide insights into a wide range of activities, such as metering, server activity, geolocation of devices, or website clicks. Likely use cases include:

Ecommerce

Analyze user clickstreams to optimize the shopping experience with real-time pricing, promotions, and inventory management.

Financial services

Analyze account activity to detect anomalous behavior in the data stream and generate a security alert for abnormal behavior.

Investment services

Track market changes and adjust settings to customer portfolios based on configured constraints, such as selling when a certain stock value is reached.

News media

Stream user click records from various news source platforms and enrich the data with demographic information to better serve articles that are relevant to the targeted audience.

Utilities

Monitor throughput across a power grid and generate alerts or initiate workflows when established thresholds are reached.

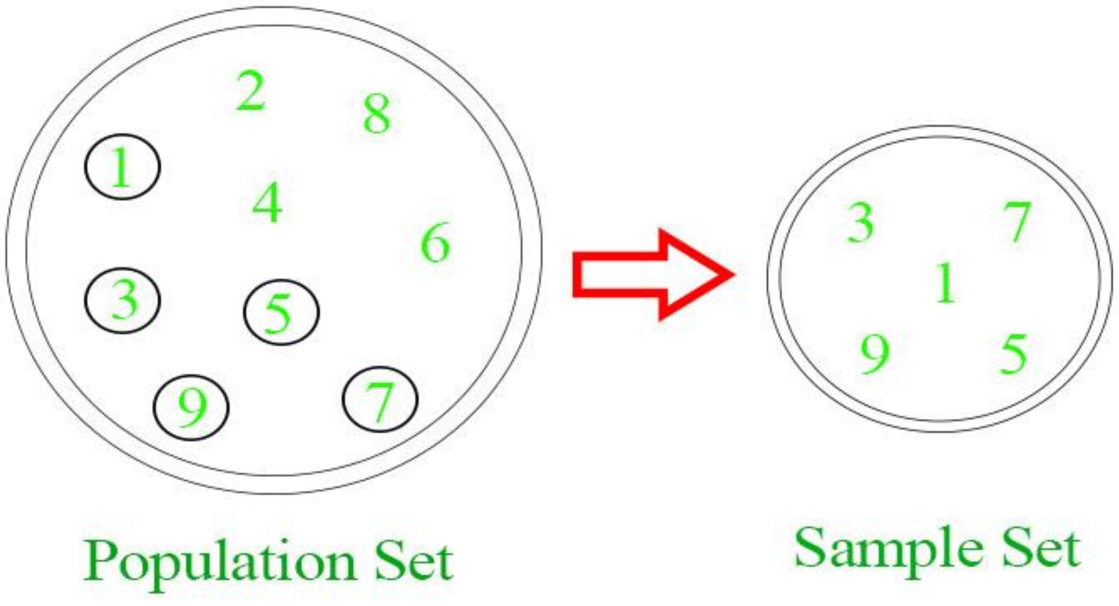
Related products and services

Stream analytics from Google Cloud makes data more organized, useful, and accessible from the instant it's generated. Built on the autoscaling infrastructure of Pub/Sub, Dataflow, and BigQuery, stream analytics from Google Cloud provisions the resources you need to ingest, process, and analyze fluctuating volumes of real-time data for real-time business insights and actions. This abstracted provisioning reduces complexity and makes stream analytics accessible to both data analysts and data engineers.

SAMPLING DATA IN A STREAM:

In the world of Statistics, the very first thing to be done before any estimation is to create a Sample set from the entire Population Set. The Population set can be seen as the entire tree from where data is collected whereas the Sample Set can be seen as the branch in which the actual study of observations and estimation is done. Population tree is a very large set and making the study of observations on it can be very exhausting, both time and money-wise alike. Thus to cut down on the amount of time and as well as resources, a Sample Set is created from the Population set. **Process of Sampling**:

1. *Identifying the Population set.*
2. *Determination of the size of our sample set.*
3. *Providing a medium for the basis of selection of samples from the Population medium.*
4. *Picking out samples from the medium using one of many Sampling techniques like Simple Random, Systematic or Stratified Sampling.*
5. *Checking whether the formed sample set, contains elements actually matches the different attributes of population set, without large variations in between.*
6. *Checking for errors or inaccurate estimations in the formed sample set, that may or may not have occurred*
7. *The set which we get after performing the above steps actually contributes to the Sample Set.*



A simple illustration of how sampling is done at it's basic stages.

Population

Population is the whole set of variables, elements, entities which are considered for a statistical study. It is also known as the universal set from where actual inferences are drawn. Population set consists of all the attributes of individuals or elements under consideration, but doing estimations on a Population is very exhausting resources as well as time-wise alike. *Example*: Consider the mean weight of all men on Earth. This here, is considered a hypothetical population because it includes all men that have ever lived on earth which includes people who will exist in the future and also people who have lived earlier before us. But there comes an anomaly, while doing such measurement which is not all men in the population tray are observable (consider men, who will exist in the future and also men, who have lived before and doesn't exist right now). Also, performing statistics on the population sample (if hypothetically possible) would require a great deal of time as well as resources, which will be exhaustive and inefficient as well. Thus what is perform instead is to take a subset from the available population and perform statistics on them and interpolate inferences about the entire population. Taking out a subset, makes the task easier as the time required to scrutinize the subset is lesser than the time required to scrutinize the whole set of Population. Statistics is performed on the sample set to draw conclusions about the entire population tray. Calculations are considered to be a conclusion of the population set because it doesn't measure with the actual data of the population set and is not free from errors. This is obvious as sample set is used as a medium frame, having fewer members and thus some information is lost. (which results in errors).

Sampling Frame

Sampling Frame is the basis of the sample medium. It is a collection of all the sample elements taken into observation. Sometimes it might even happen that all elements in the sampling frame, didn't even take part in the actual statistics. In that case, the elements that took part in the study are called **Samples** and potential elements that could have been in the study but didn't take part forms the **Sampling Frame.** Thus, Sampling Frame is the potential list of elements on which we will perform our statistics. Coming up with a good sampling frame is very essential because it will help in predicting the reaction of the statistics result with the population set. A sampling frame is not just a random set of handpicked elements rather it even consists of identifiers which help to identify each and every element in the set. *Example:* GeeksForGeeks organized a meetup of all the Geek Interns all over India at Delhi to perform a statistical study on their performances. GfG sent an invitational email to all 500 of those Interns, but since all the Interns are scattered all over India, out of 500 people of 200 show up in actual. And thus, GfG had to perform their study on 200 students only (Sample Set). But the remaining 300 people who could have been the potential candidates in the study, but decided not to show up forms the Sampling Frame. **Methods and Types of sampling**:

1. Simple Random Sampling
2. Systematic Sampling
3. Stratified Sampling

These are the most widely used Sampling Processes with each having their both advantages as well as disadvantages. Let us look at each of these sampling methods in details:

1. **Simple Random Sampling**: Simple Random Sampling is the most elementary form of sampling. In this method, all the elements in populations are first divided into random sets of equal sizes. Random sets have no defining property among themselves, i.e one set cannot be identified from another set based on some specific identifiers. Thus every element has an equal property of being selected. **P(of getting selected)**

   = The basic methods for employing SRS are:
   - Choose the Population Set
   - Identify the basis of Sampling
   - Use of random number/session generators to pick an element from each set.
   - Less exhaustive with respect to time as it is the most elementary form of sampling
   - Very useful for population set with very less number of elements
   - SRS can be employed anywhere, anytime even without the use of special random generators
   - Not efficient for large population sets
   - Causes the most number of errors out of the three mentioned methods of sampling
   - There are chances of bias and then SRS won't be able to provide a correct result
   - Does not provide a specific identifier to separate statistically similar samples
2. **Systematic Sampling**: Systematic Sampling is also known as a type of probability sampling. It is much more accurate than SRS and also the standard error formation percentage is very low but not error-free. In this method, first, the population tray elements are arranged based on a specific order or scheme properly known as being sorted. It can be of any order, which totally depends upon the person performing the statistics. The elements are first arranged either ascendingly, descending, lexicographically or any other known methods deemed fit by the tester. Although the start point needs to be random every time. After being arranged, then the sample elements are picked based on a pre-defined interval set or function. **Example**: In a random set of numbers with elements ranging from 1 to 100. The elements are first sorted either in ascending or descending order. Then let's say every 4th element is picked to be a part of the sampling frame. This kind of sampling is known as Systematic Sampling. **P(of getting selected) = [depends upon the ordered population tray after it has been sorted]** The basic methods of employing Systematic Random Sampling are :-
   - Choosing the Population Set wisely
   - Checking whether Systematic Sampling will be the efficient method or not.
   - If Yes, then Application of an sorting method to get an ordered pair of population elements.
   - Choosing a periodicity to crawl out elements.
   - Accuracy is higher than SRS.

- Standard probability of error is lesser .
- No problem for bias to creep in during creation of sample frame.
- Not much efficient when comes to the time wise
- Periodicity in population tray elements can lead to absurd results.
- Systematic sampling can either provide the most accurate result or an impossible one.

3. **Stratified Sampling**: Stratified Sampling is the most complex type of Sampling Method out of all the three methods mentioned above. It is a hybrid method concerning both simple random sampling as well as systematic sampling. It is one of the most advanced types of sampling method available, providing near accurate result to the tester. In this method, the population tray is divided into sub-segments also known as stratum(singular). Each stratum can have their own unique property. After being divided into different sub-stratum, SRS or Systematic Sampling can be used to create and pick out samples for performing statistics. The elementary methods for Stratified Sampling are :
    - Choosing the population tray wisely.
    - Checking for periodicity or any other features, so that they can be divided into different strata
    - Dividing the population tray into sub-sets and sub-groups on the basis of selective property.
    - Using SRS or Systematic Sampling of each individual strata to form the sample frame.
    - We can even apply different sampling methods to different sub-sets.
    - Provide results with high accuracy measurements.
    - Different results can be desired just by changing the Sampling method.
    - This method also compares different strata when samples are being drawn.
    - Inefficient and Expensive when comes to resources as well as money.
    - This method will fail only in rare cases where homogeneity in elements is present.

These three are the widely used methods of Sampling which are being done nowadays. Each of them has their own advantages as well as disadvantages. So, the sampling method must be chosen wisely, because a wrong choice can lead to erroneous answers.

Methods of Sampling

1. Random Sampling

As the name suggests, in this method of sampling, the data is collected at random. It means that every item of the universe has an equal chance of getting selected for the investigation purpose. In other words, each item has an equal probability of being in the sample, which makes the method impartial. As there is no control of the investigator in selecting the sample, the random sampling method is used for homogeneous items. As there is no tool or a large number of people required for collecting data through random sampling, this method is economical. There are two ways of collecting data through the random sampling method. These are the Lottery Method and Tables of Random Numbers.

- **Lottery Method:** In Lottery Method, the investigator prepares paper slips for each of the items of the universe and shuffles these slips in a box. After that, some slips are impartially drawn out from the box to obtain a sample.
- **Table of Random Numbers:** A Table of Random Numbers has been prepared by a group of statisticians. In this method of collecting data through random sampling, this table is referred by the investigator to frame a sample. There are many Tables of Random Numbers available from which Tippet's Table is used by most of the investigators. In this Table, Tippet has used 41,600 figures and has involved 10,400 numbers with four units in each of the numbers. Now, through this method, the items available in the universe are first arranged in an order, and then using Tippet's Table, the investigator selects the required number of items to form a sample.

Merits of Random Sampling Method

1. Random Sampling method is economical as the items are selected randomly, which can be done by fewer people and with fewer resources.

2. Random Sampling method is impartial and free from personal biases, as it randomly selects the numbers, and each of the items has an equal probability of getting selected.

3. This method fairly represents the universe through samples.

4. It is a straightforward and simple method of collecting data.

Demerits of Random Sampling Method

1. Despite its various advantages, the random sampling method does not give proper weightage to some important items of the universe.

2. Also, there is no guarantee that different items of the universe are proportionately represented.

Random Sampling is sometimes confused with Haphazard Sampling. But, it is not Haphazard Sampling. There is a difference between these two sampling methods.

Random Sampling works with the rules of sampling. However, Haphazard Sampling does not work with the rules of sampling.

Also, the random sampling method gives equal chance to each item being selected. However, the haphazard sampling method does not provide an equal chance for each item.

2. Purposive or Deliberate Sampling

The method in which the investigator himself selects the sample of his choice, which in his opinion is best to represent the universe, is known as **Purposive or Deliberate Sampling.** It means that the probability of an item getting selected is not equal as the sample is selected by choice. This method is suitable under situations when there are some items in the universe whose involvement or selection in the sample is important. **For example,** If an investigation is about FMCG Companies, then the inclusion of companies like Nestle, Hindustan Unilever Ltd., etc., is essential in the sample. However, the chances of personal biases in this method of sampling are more, which reduces its credibility.

Merits of Purposive or Deliberate Sampling

1. The Purposive or Deliberate Sampling Method is flexible, as it allows an investigator to include items with special significance in the sample.

2. The investigator can easily tune the selection of items based on the purpose of the investigation, making it easy for him to perform the analysis.

3. It is a very simple technique of collecting data, as the investigator can select the significant items in the sample by his choice.

Demerits of Purposive or Deliberate Sampling

1. As the investigator can select an item in the sample for the investigation, the probability of personal biases increases.

2. An increase in the probability of personal biases makes the method less reliable for collecting data, and the results become doubtful.

3. Stratified or Mixed Sampling

A sampling method which is suitable at times when the population has different groups with different characteristics and an investigation is to be performed on them is known as Stratified or Mixed Sampling. In other words, **Stratified or Mixed Sampling** is a method in which the population is divided into different groups, also known as strata with different characteristics, and from those strata some of the items are selected to represent the population. The investigator while forming strata has to ensure that each of the stratum is represented in a correct proportion. **For example,** there are 60 students in Class 10th. Out of these 60 students, 10 opted for Arts and Humanities, 30 opted for Commerce, and 20 opted for Science in Class 11th. It means that the population of 60 students is divided into three strata; viz., Arts and Humanities, Commerce, and Science, containing 10, 30, and 20 students, respectively. Now, for investigation purpose, some of the items will be proportionately selected from each of the strata in a way that those items forming a sample represents the entire population. Besides, an investigator can even select the items from different strata, unproportionately.

Merits of Stratified or Mixed Sampling

1. As different groups of a population with different characteristics are selected in this method of sampling, it covers a large portion of the characteristics of the population.

2. Selection of the diverse characteristics of the population makes the comparative analysis of the data possible.

3. The Stratified Method of Sampling offers meaningful and reliable results to the investigator.

Demerits of Stratified or Mixed Sampling

1. The Stratified Sampling Method has a limited scope because it is suitable only when the investigator has complete knowledge of the diverse characteristics of the entire population.

2. As the population is divided into different strata by the investigator himself, there are chances of biasness in this step.

3. In the case of a small population, it may be difficult for the investigator to divide the population into small strata.

*Stratified Sampling Method is also known as **Mixed Sampling Method** because it is a mixture of both **Purposive Sampling** and **Random Sampling Method.** The population is divided into different strata on purpose; however, the items are selected from different strata, randomly.*

4. Systematic Sampling

According to the **Systematic Sampling Method** of collecting data, different units of the population are systematically arranged in numerical, alphabetical, and geographical order. To form a sample, every nth term or item of the numbered items is selected. This method is a short-cut method of collecting data through the Random Sampling method. **For example,** if 10 out of 200 people are to be selected for investigation, then these are first arranged in a systematic order. After that one of the first 10 people would be randomly selected. In the same way, every 10th person from the selected item will be taken under the sample. In other words, if the first selected person is the 8th person in the order, then the subsequent person selected in the sample would be 18th, 28th, 38th, 48th, ……….198th.

Merits of Systematic Sampling

1. Systematic Sampling Method is a simple method of collecting data as the investigator can easily determine the sample.

2. As the items are arranged in a systematic order, the chances of personal biases are less.

Demerits of Systematic Sampling

1. As the first item of the given population is selected randomly, and then further items are selected on the basis of the first item, every item of the population does not get an equal chance of getting selected.

2. In case the population has homogeneous items, the method of Systematic Sampling does not serve any specific purpose.

5. Quota Sampling

In the **Quota Sampling Method** of collecting data, the entire population is divided into different classes or groups. It is done on the basis of the different characteristics of the given population. The investigator fixes some percentages of the different groups with different characteristics of the total population. After that, he fixes some quota of the items for each of the selected segregated groups. At last, to form a sample, the investigator has to select a fixed number of items from each of the segregated groups.

Merits of Quota Sampling

1. The Quota Sampling Method of collecting data is affordable.

Demerits of Quota Sampling

1. The chances of personal biases while selecting the items in a sample are high.

2. Personal biases during the selection of items in a sample make the reliability of the results through investigation questionable.

6. Convenience Sampling

As the name suggests, **Convenience Sampling** is a method of collecting data in which the investigator selects the items from the population that suits his convenience. **For example,** an investigator who wants to collect data on the average number of females using inductions in the kitchen goes to a shopping mall and collects information from each of the females visiting there. By doing so, the investigator is neglecting other females who were not present in the mall that day or did not go to the mall. This reduces the reliability of the result, as there are other females in the universe who uses inductions in the kitchen, but were not present in the mall at that time.

Merits of Convenience Sampling

1. The Convenience Sampling Method is the least expensive method of collecting data.

2. It is also the simplest method of collecting data from the population.

Demerits of Convenience Sampling

1. This method is highly unreliable, as the investigator selects the items that suit him, and it is not possible that every investigator has reliable thinking or purpose of investigation. Besides, different investigators have different perspectives.

FILTERING STREAMS:

Suppose you are creating an account on Geekbook, you want to enter a cool username, you entered it and got a message, “Username is already taken”. You added your birth date along username, still no luck. Now you have added your university roll number also, still got “Username is already taken”. It’s really frustrating, isn’t it?
But have you ever thought about how quickly Geekbook checks availability of username by searching millions of username registered with it. There are many ways to do this job –

- **Linear search** : Bad idea!
- **Binary Search** : Store all username alphabetically and compare entered username with middle one in list, If it matched, then username is taken otherwise figure out, whether entered username will come before or after middle one and if it will come after, neglect all the usernames before middle one(inclusive). Now search after middle one and repeat this process until you got a match or search end with no match. This technique is better and promising but still it requires multiple steps.
  But, there must be something better!!

Bloom Filter is a data structure that can do this job.
For understanding bloom filters, you must know what is hashing. A hash function takes input and outputs a unique identifier of fixed length which is used for identification of input.

What is Bloom Filter?

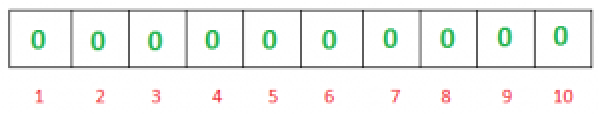
A Bloom filter is a **space-efficient probabilistic** data structure that is used to test whether an element is a member of a set. For example, checking availability of username is set membership problem, where the set is the list of all registered username. The price we pay for efficiency is that it is probabilistic in nature that means, there might be some False Positive results. **False positive means**, it might tell that given username is already taken but actually it’s not.

Interesting Properties of Bloom Filters

- Unlike a standard hash table, a Bloom filter of a fixed size can represent a set with an arbitrarily large number of elements.
- Adding an element never fails. However, the false positive rate increases steadily as elements are added until all bits in the filter are set to 1, at which point all queries yield a positive result.
- Bloom filters never generate **false negative** result, i.e., telling you that a username doesn’t exist when it actually exists.
- Deleting elements from filter is not possible because, if we delete a single element by clearing bits at indices generated by k hash functions, it might cause deletion of few other elements. Example – if we delete “geeks” (in given example below) by clearing bit at 1, 4 and 7, we might end up deleting “nerd” also Because bit at index 4 becomes 0 and bloom filter claims that “nerd” is not present.

Working of Bloom Filter

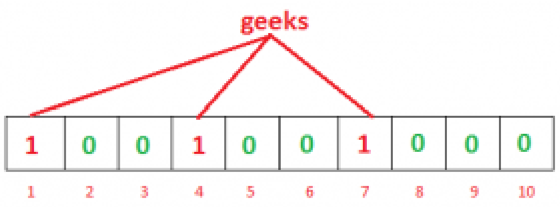
A empty bloom filter is a **bit array** of **m** bits, all set to zero, like this –



We need **k** number of **hash functions** to calculate the hashes for a given input. When we want to add an item in the filter, the bits at k indices h1(x), h2(x), … hk(x) are set, where indices are calculated using hash functions.
Example – Suppose we want to enter “geeks” in the filter, we are using 3 hash functions and a bit array of length 10, all set to 0 initially. First we’ll calculate the hashes as follows:

```
h1(“geeks”) % 10 = 1

h2(“geeks”) % 10 = 4

h3(“geeks”) % 10 = 7
```

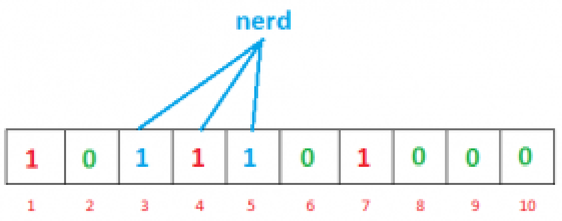
Note: These outputs are random for explanation only.
Now we will set the bits at indices 1, 4 and 7 to 1



Again we want to enter “nerd”, similarly, we’ll calculate hashes

```
h1(“nerd”) % 10 = 3

h2(“nerd”) % 10 = 5

h3(“nerd”) % 10 = 4
```

Set the bits at indices 3, 5 and 4 to 1



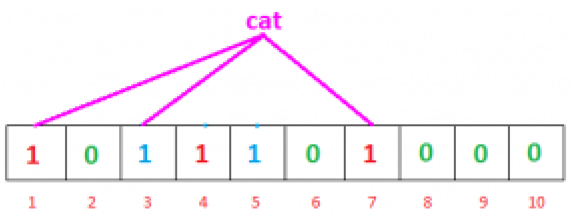
Now if we want to check “geeks” is present in filter or not. We’ll do the same process but this time in reverse order. We calculate respective hashes using h1, h2 and h3 and check if all these indices are set to 1 in the bit array. If all the bits are set then we can say that “geeks” is **probably present**. If any of the bit at these indices are 0 then “geeks” is **definitely not present**.

False Positive in Bloom Filters

The question is why we said **“probably present”**, why this uncertainty. Let’s understand this with an example. Suppose we want to check whether “cat” is present or not. We’ll calculate hashes using h1, h2 and h3

```
h1(“cat”) % 10 = 1

h2(“cat”) % 10 = 3

h3(“cat”) % 10 = 7
```

If we check the bit array, bits at these indices are set to 1 but we know that “cat” was never added to the filter. Bit at index 1 and 7 was set when we added “geeks” and bit 3 was set we added “nerd”.



So, because bits at calculated indices are already set by some other item, bloom filter erroneously claims that “cat” is present and generating a false positive result. Depending on the application, it could be huge downside or relatively okay.

We can control the probability of getting a false positive by controlling the size of the Bloom filter. More space means fewer false positives. If we want to decrease probability of false positive result, we have to use more number of hash functions and larger bit array. This would add latency in addition to the item and checking membership.

Operations that a Bloom Filter supports

- insert(x) : To insert an element in the Bloom Filter.
- lookup(x) : to check whether an element is already present in Bloom Filter with a positive false probability.

NOTE : We cannot delete an element in Bloom Filter.

Probability of False positivity: Let **m** be the size of bit array, k be the number of hash functions and **n** be the number of expected elements to be inserted in the filter, then the probability of false positive **p** can be calculated as:

Size of Bit Array: If expected number of elements **n** is known and desired false positive probability is **p** then the size of bit array **m** can be calculated as :

Optimum number of hash functions: The number of hash functions **k** must be a positive integer. If **m** is size of bit array and **n** is number of elements to be inserted, then k can be calculated as :

Space Efficiency

If we want to store large list of items in a set for purpose of set membership, we can store it in hashmap, tries or simple array or linked list. All these methods require storing item itself, which is not very memory efficient. For example, if we want to store "geeks" in hashmap we have to store actual string " geeks" as a key value pair {some_key : "geeks"}.

Bloom filters do not store the data item at all. As we have seen they use bit array which allow hash collision. Without hash collision, it would not be compact.

Choice of Hash Function

The hash function used in bloom filters should be independent and uniformly distributed. They should be fast as possible. Fast simple non cryptographic hashes which are independent enough include murmur, FNV series of hash functions and Jenkins hashes.

Generating hash is major operation in bloom filters. Cryptographic hash functions provide stability and guarantee but are expensive in calculation. With increase in number of hash functions k, bloom filter become slow. All though non-cryptographic hash functions do not provide guarantee but provide major performance improvement.

COUNTING DISTINCT ELEMENTS IN A STREAM:

Counting distinct elements is a problem that frequently arises in distributed systems. In general, the size of the set under consideration (which we will henceforth call the *universe*) is enormous. For example, if we build a system to identify denial of service attacks, the set could consist of all IP V4 and V6 addresses. Another common use case is to count the number of unique visitors on popular websites like Twitter or Facebook.

An obvious approach if the number of elements is not very large would be to maintain a *Set*. We can check if the set contains the element when a new element arrives. If not, we add the element to the set. The size of the set would give the number of distinct elements. However, if the number of elements is vast or we are maintaining counts for multiple streams, it would be infeasible to maintain the set in memory. Storing the data on disk would be an option if we are only interested in offline computation using batch processing frameworks like *Map Reduce*.

Like the previous algorithms we looked at, the algorithms for counting distinct elements are also approximate, with an error threshold that can be tweaked by changing the algorithm's parameters.

Flajolet — Martin Algorithm

The first algorithm for counting distinct elements is the Flajolet-Martin algorithm, named after the algorithm's creators. The Flajolet-Martin algorithm is a single pass algorithm. If there are `m` distinct elements in a *universe* comprising of `n` elements, the algorithm runs in *O(n)* time and *O(log(m))* space complexity. The following steps define the algorithm.

- First, we pick a hash function `h` that takes stream elements as input and outputs a bit string. The length of the bit strings is large enough such that the result of the hash function is much larger than the size of the *universe*. We require at least `log n`bits if there are `n` elements in the *universe*.
- `r(a)` is used to denote the number of trailing zeros in the binary representation of `h(a)` for an element `a` in the stream.
- *R* denotes the maximum value of r seen in the stream so far.
- The estimate of the number of distinct elements in the stream is *(*`2^R`*)*.

To intuitively understand why the algorithm works consider the following.

The probability that `h(a)` ends in at least `i` zeros is exactly *(*`2^-i`*)*. For example, for `i=0`, there is a probability `1`that the tail has at least `0`zeros. For `i=1`, there is a probability of `1/2` that the last bit is zero; for `i=2`, the probability is `1/4` that the last two bits are zero's, and so on. The probability of the rightmost set bit drops by a factor of `1/2` with every position from the Least Significant Bit to the Most Significant Bit.

This probability should become `0` when bit position `R >> log m` while it should be non-zero when `R <= log m.` Hence, if we find the rightmost unset bit position `R` such that the probability is `0`, we can say that the number of unique elements will approximately be `2 ^ R`.

The Flajolet-Martin uses a multiplicative hash function to transform the non-uniform set space to a uniform distribution. The general form of the hash function is

$(ax + b) \bmod c$ where a and b are odd numbers and c is the length of the hash range.

The Flajolet-Martin algorithm is sensitive to the hash function used, and results vary widely based on the data set and the hash function. Hence there are better algorithms that utilize more than one hash function. These algorithms use the average and median values to reduce skew and increase the predictability of the result.

Flajolet-Martin Psuedocode and Explanation

```
1. L = 64 (size of the bitset), B= bitset of size L
2. hash_func = (ax + b) mod 2^L
3. for each item x in stream
     • y = hash(x)
     • r = get_righmost_set_bit(y)
     • set_bit(B, r)
4. R = get_righmost_unset_bit(B)
5. return 2 ^ R
```

We define a hash range, big enough to hold the maximum number of possible unique values, something as big as `2 ^ 64`. Every stream element is passed through a hash function that transforms the elements into a uniform distribution.

For each hash value, we find the position of the rightmost set bit and mark the corresponding position in the bitset to 1. Once all elements are processed, the bit vector will have `1`s at all the positions corresponding to the position of every rightmost set bit for all elements in the stream.

Now we find`R` , the rightmost `0` in this bit vector. This position `R` corresponds to the rightmost set bit that we have not seen while processing the elements. This corresponds to the probability `0` and will help in approximating the cardinality of unique elements as `2 ^ R`.

ESTIMATING MOMENTS:

- Estimating moments is a generalization of the problem of counting distinct elements in a stream. The problem, called computing "moments," involves the distribution of frequencies of different elements in the stream.

- Suppose a stream consists of elements chosen from a universal set. Assume the universal set is ordered so we can speak of the ith��h element for any i.
- Let mi�� be the number of occurrences of the ith��h element for any i. Then the kth��h-order moment of the stream is the sum over all i of (mi)k(��)�

. **For example :-**

- The 0th0�h moment is the sum of 1 of each mi that is greater than 0 i.e., 0th0�h moment is a count of the number of distinct element in the stream.
- The 1st moment is the sum of the mi�� 's, which must be the length of the stream. Thus, first moments are especially easy to compute i.e., just count the length of the stream seen so far.
- The second moment is the sum of the squares of the mi��'s. It is sometimes called the surprise number, since it measures how uneven the distribution of elements in the stream is.
- To see the distinction, suppose we have a stream of length 100, in which eleven different elements appear. The most even distribution of these eleven elements would have one appearing 10 times and the other ten appearing 9 times each.

- In this case, the surprise number is 102102 + 10 × 9292 = 910. At the other extreme, one of the eleven elements could appear 90 times and the other ten appear 1 time each. Then, the surprise number would be 902902 + 10 × 12 = 8110.

- Moments of order k:

- ● If a stream has A distinct elements, and each element has frequency mi
- ● The kth order moment of the stream is :
- ● The 0th order moment is the number of distinct elements in the stream
- ● The 1st order moment is the length of the stream 5/17 Moments of order k (cont.)
- ● The kth order moment of the stream is:
- ● The 2nd order moment is also known as the "surprise number" of a stream (large values = more uneven distribution) mi i=1 i=2 i=3 i=4 i=5 i=6 i=7 i=8 i=9 i=10 i=11 2 nd moment Seq1 10 9 9 9 9 9 9 9 9 9 9 910 Seq2 90 1 1 1 1 1 1 1 1 1 1 8110 6/17 Method for second moment
- ● Assume (for now) that we know n, the length of the stream
- ● We will sample s positions

- ● For each sample we will have X.element and X.count
- ● We sample s random positions in the stream X.element = element in that position, X.count ← 1 When we see X.element again, X.count ← X.count + 1
- ● Estimate second moment as n(2 × X.count - 1) Alon, N., Matias, Y., & Szegedy, M. (1999). The space complexity of approximating the frequency moments. Journal of Computer and system sciences, 58(1), 137-147. 7/17 Method for second moment (cont.)
- ● Example: a,b,c,b,d,a,c,d,a,b,d,c,a,a,b ma = 5, mb = 4, mc = 3, md = 3 second moment = 52+42+32+32 = 59
- ● Suppose we sample s=3 variables X1 , X2 , X3
- ● Suppose we pick the 3rd, 8th, and 13th position at random
- ● X1 .element=c, X2 .element=d, X3 .element=a
- ● X1 .count=3, X2 .count=2, X3 .count=2 (we count forwards only!)
- ● Estimate n(2 × X.count – 1), first estimate = 15(6-1) = 75, second estimate 15(4-1) = 45, third estimate 15(4-1) = 45, average of estimates = 55≃59 Alon, N., Matias, Y., & Szegedy, M. (1999). The space complexity of approximating the frequency moments. Journal of Computer and system sciences, 58(1), 137-147. 8/17 Method for second moment (cont.)
- ● Example: a,b,c,b,d,a,c,d,a,b,d,c,a,a,b
- ● Suppose we pick the 3rd, 8th, and 13th position at random
- ● X1 .element=c, X2 .element=d, X3 .element=a
- ● X1 .count=3, X2 .count=2, X3 .count=2 Alon, N., Matias, Y., & Szegedy, M. (1999). The space complexity of approximating the frequency moments. Journal of Computer and system sciences, 58(1), 137-147. 9/17 Why this method works?
- ● Let e(i) be the element in position i of the stream
- ● Let c(i) be the number of times e(i) appears in positions i, i+1, i+2, …, n
- ● Example: a,b,c,b,d,a,c,d,a,b,d,c,a,a,b c(6) = ? 10/17 Why this method works?
- ● Let e(i) be the element in position i of the stream
- ● Let c(i) be the number of times e(i) appears in positions i, i+1, i+2, …, n
- ● Example: a,b,c,b,d,a,c,d,a,b,d,c,a,a,b c(6) = 4 (remember: we count forwards only!) 11/17 Why this method works? (cont.)
- ● c(i) is the number of times e(i) appears in positions i, i+1, i+2, …, n
- ● E[n (2 × X.count – 1)] is the average of n (2 c(i) – 1) over all positions i=1...n 12/17 Why this method works? (cont.)
- ● Now focus on element a that appears ma times in the stream – The last time a appears this term is 2c(i) – 1 = 2x1-1 = 1 – Just before that, 2c(i)-1 = 2x2-1 = 3 – … – Until 2ma – 1 for the first time a appears
- ● Hence 13/17 For higher order moments (v = X.count)
- ● For second order moment – We use n(2v-1) = n(v2 – (v-1)2)
- ● For third order moment – We use n(3v2 – 3v + 1) = n(v3 – (v-1)3)
- ● For k th order moment – We use n(vk - (v-1)k) 14/17 For infinite streams
- ● Use a reservoir sampling strategy
- ● If we want s samples – Pick the first s elements of the stream setting Xi .element ← e(i) and Xi .count ← 1 for i=1...s – When element n+1 arrives
- ● Pick Xn+1.element with probability s/(n+1), evicting one of the existing elements at random and setting X.count ← 1
- ● As before, probability of an element is s/n

COUNTING ONENESS IN A WINDOW:

Given an array of size **N** and an integer **K**, return the count of distinct numbers in all windows of size K.
Examples:
Input: *arr[] = {1, 2, 1, 3, 4, 2, 3}, K = 4*
Output: *3 4 4 3*
Explanation*: First window is {1, 2, 1, 3}, count of distinct numbers is 3*
Second window is {2, 1, 3, 4} count of distinct numbers is 4
Third window is {1, 3, 4, 2} count of distinct numbers is 4
Fourth window is {3, 4, 2, 3} count of distinct numbers is 3
Input*: arr[] = {1, 2, 4, 4}, K = 2*
Output*: 2 2 1*

***Explanation**: First window is {1, 2}, count of distinct numbers is 2*
First window is {2, 4}, count of distinct numbers is 2
First window is {4, 4}, count of distinct numbers is 1

Count distinct elements in every window

Follow the given steps to solve the problem:

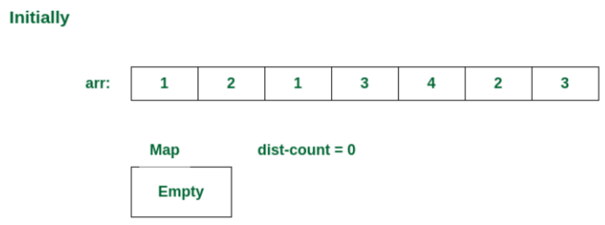
- For every index, i from 0 to N – K, traverse the array from *i* to *i + k using another loop*. This is the window
- Traverse the window, from *i* to that index and check if the element is present or not
- If the element is not present in the prefix of the array, i.e no duplicate element is present from *i* to *index-1*, then increase the count, else ignore it
- Print the count

Time complexity: O(N * K^2)
Auxiliary Space: O(1)
Count distinct numbers in all windows of size K using hashing:
So, there is an efficient solution using hashing, though hashing requires extra O(n) space but the time complexity will improve. The trick is to use the count of the previous window while sliding the window. To do this a hash map can be used that stores elements of the current window. The hash-map is also operated on by simultaneous addition and removal of an element while keeping track of distinct elements. The problem deals with finding the count of distinct elements in a window of length k, at any step while shifting the window and discarding all the computation done in the previous step, even though k – 1 elements are same from the previous adjacent window. For example, assume that elements from index i to i + k – 1 are stored in a Hash Map as an element-frequency pair. So, while updating the Hash Map in range i + 1 to i + k, reduce the frequency of the i-th element by 1 and increase the frequency of (i + k)-th element by 1.
Insertion and deletion from the HashMap takes constant time.
Follow the given steps to solve the problem:

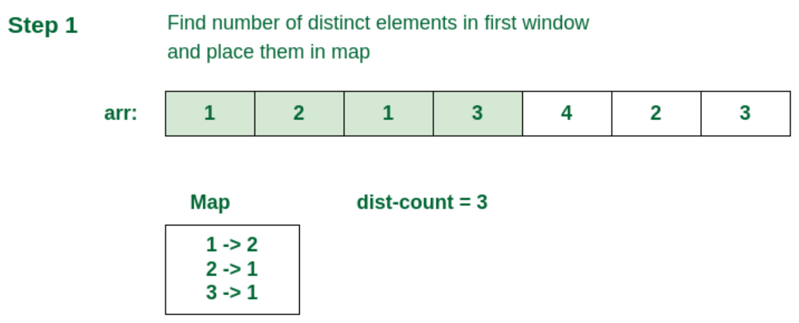
- Create an empty hash map. Let the hash map be *hm*.
- Initialize the count of distinct elements as *dist_count* to 0.
- Traverse through the first window and insert elements of the first window to *hm*. The elements are used as key and their counts as the value in hm. Also, keep updating *dist_count*
- Print distinct count for the first window.
- Traverse through the remaining array (or other windows).
- Remove the first element of the previous window.
    - If the removed element appeared only once, remove it from *hm* and decrease the distinct count, i.e. do *"dist_count–"*
    - else (appeared multiple times in hm), then decrement its count in hm
- Add the current element (last element of the new window)
    - If the added element is not present in hm, add it to *hm* and increase the distinct count, i.e. do *"dist_count++"*
    - Else (the added element appeared multiple times), increment its count in *hm*

Below is the illustration of the above approach:

Consider the array arr[] = {1, 2, 1, 3, 4, 2, 3} and K = 4

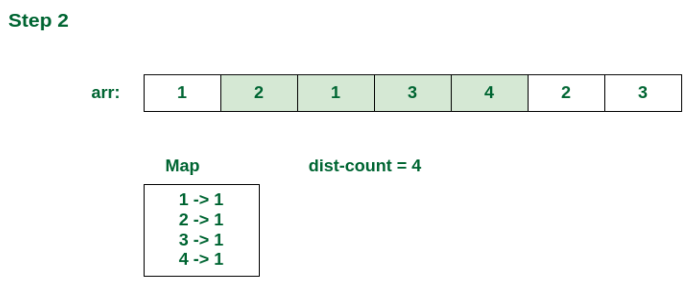


Initial State

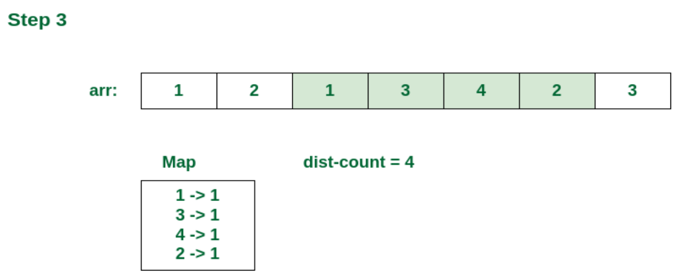


Checking in the first window

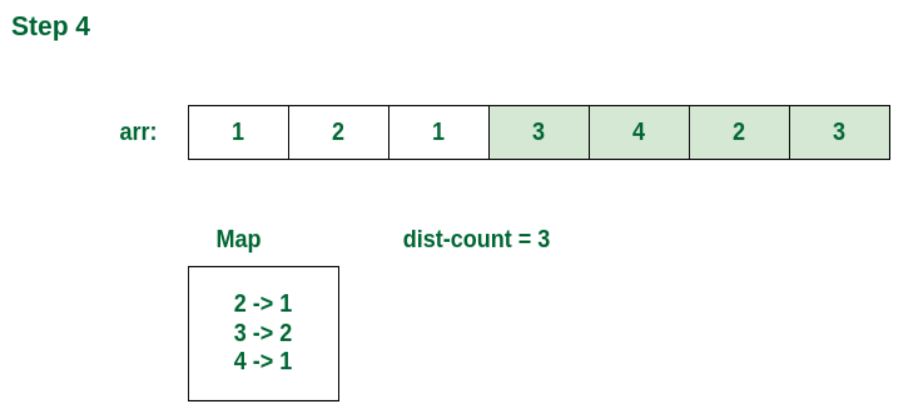
In all the remaining steps, remove the first element of the previous window and add new element of current window.



Checking in the second window



Checking in the third window



DECAYING WINDOW:

This algorithm allows you to identify the most popular elements (trending, in other words) in an incoming data stream.

The decaying window algorithm not only tracks the most recurring elements in an incoming data stream, but also discounts any random spikes or spam requests that might have boosted an element's frequency. In a decaying window, you assign a score or weight to every element of the incoming data stream. Further, you need to 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.

1. Assign each element with a weight/score.
2. Calculate aggregate sum for each distinct element by adding all the weights assigned to that element.

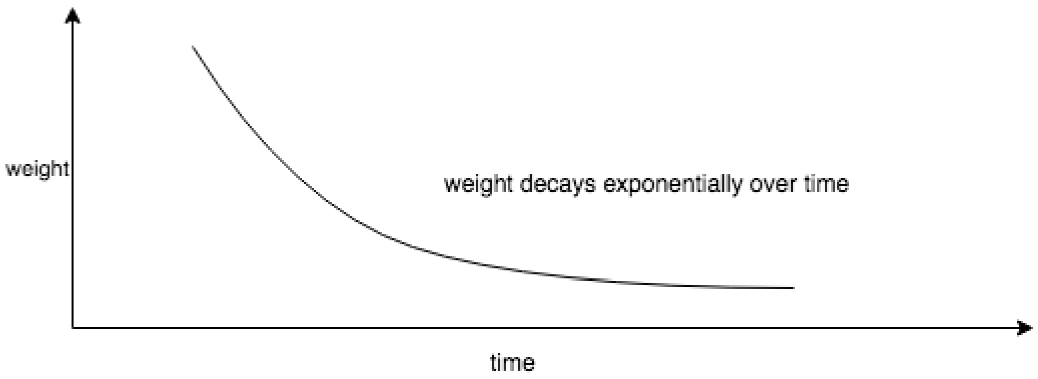
In a decaying window algorithm, ***you assign more weight to newer elements***. For a new element, you first reduce the weight of all the existing elements by a constant factor k and then assign the new element with a specific weight. The aggregate sum of the decaying exponential weights can be calculated using the following formula:

$\sum t-1i=0at-i(1-c)i$

Here, c is usually a small constant of the order
10^{-6} or 10^{-9}. Whenever a new element, say a_{t+1} , arrives in the data stream you perform the following steps to achieve an updated sum:

1. Multiply the current sum/score by the value (1−c).

2. Add the weight corresponding to the new element.



Weight decays exponentially over time

In a data stream consisting of various elements, you maintain a separate sum for each distinct element. For every incoming element, you multiply the sum of all the existing elements by a value of (1−c). Further, you add the weight of the incoming element to its corresponding aggregate sum.

A threshold can be kept to, ignore elements of weight lesser than that.

Finally, the element with the highest aggregate score is listed as the most popular element.

Example

For example, consider a sequence of twitter tags below:
fifa, ipl, fifa, ipl, ipl, ipl, fifa

Also, let's say each element in sequence has weight of 1.
Let's c be 0.1
The aggregate sum of each tag in the end of above stream will be calculated as below:

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 = ***2.135***

ipl

fifa - 0 * (1-0.1) = 0

ipl - 0 * (1-0.1) + 1 = 1

fifa - 1 * (1-0.1) + 0 = 0.9 (adding 0 because current tag is different than ipl)

ipl - 0.9 * (1-0.01) + 1 = 1.81

ipl - 1.81 * (1-0.01) + 1 = 2.7919

ipl - 2.7919 * (1-0.01) + 1 = 3.764

fifa - 3.764 * (1-0.01) + 0 = ***3.7264***

In the end of the sequence, we can see the score of ***fifa is 2.135*** but ***ipl is 3.7264***

So, ***ipl*** is more trending then ***fifa***

Even though both of them occurred same number of times in input there score is still different.

Advantages of Decaying Window Algorithm:

1. Sudden spikes or spam data is taken care.
2. New element is given more weight by this mechanism, to achieve right trending output.

REAL TIME ANALYTICS PLATFORM(RTAP) APPLICATIONS:

Real-time analytics permits businesses to get awareness and take action on data immediately or soon after the data enters their system. Real-time app analytics response queries within seconds. They grasp a large amount of data with high velocity and low reaction time. For example, real-time big data analytics uses data in financial databases to notify trading decisions. Analytics can be on-demand or uninterrupted. On-demand notifies results when the user requests it. Continuous renovation users as events happen and can be programmed to answer automatically to certain events. For example, real-time web analytics might refurbish an administrator if the page load presentation goes out of the present boundary.

Examples –
Examples of real-time customers analytics include as follows.

1. Viewing orders as they happen for better tracing and to identify fashion.
2. Continually modernize customer activity like page views and shopping cart use to understand user etiquette.
3. Choose customers with advancement as they shop for items in a store, affecting real-time decisions.

The functioning of real-time analytics :
Real-time data analytics tools can either push or pull. Streaming requires the faculty to shove gigantic amounts of brisk-moving data. When streaming takes too many assets and isn't empirical, data can be hauled at interlude that can range from seconds to hours. The tow can happen in between business needs which require figuring funds so as not to disrupt functioning. The reaction times for real-time analytics can differ from nearly immediate to a few seconds or minutes. The components of real-time data analytics include as follows.

- Aggregator
- Broker
- Analytics engine
- Stream processor

Benefits of using real-time analytics :

1. Momentum is the main welfare of real-time data analytics. The less time a business must wait to access data between the time it arrives and is processed, the business can use data insights to make changes and act on a critical decision.
2. Similarly, real-time data analytics gadgets let companies see how users link with a product upon liberating, which means there is no detain in understanding user conduct for making the needed adaptation.

Advantages of Real-time analytics :
Real-time analytics offers the following advantages over traditional analytics as follows.

- Create custom interactive analytics tools.
- Share information through transparent dashboards.
- Customize monitoring of behavior.
- Make immediate changes when needed.
- Apply machine learning.

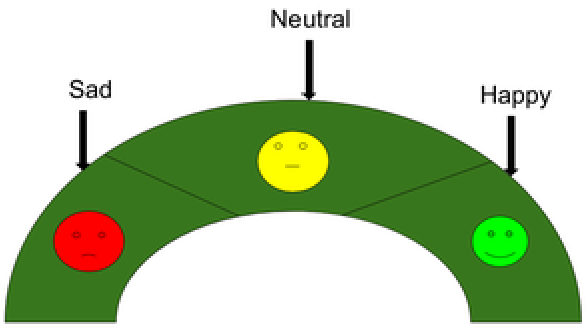
OtherBenefits:
Other benefits and uses include Managing location data, Detecting anomalies, and Better marketing, etc. are following benefits which are also useful.

Real Time System is a system that is put through real time which means response is obtained within a specified timing constraint or system meets the specified deadline.Real time system is of two types – Hard and Soft. Both are used in different cases. Hard real time systems are used where even the delay of some nano or micro seconds are not allowed. Soft real time systems provide some relaxation in time expression.
Applications of Real-time System:
Real-time System has applications in various fields of the technology. Here we will discuss the important applications of real-time system.
1. Industrial application:
Real-time system has a vast and prominent role in modern industries. Systems are made real time based so that maximum and accurate output can be obtained. In order to such things real -time systems are used in maximum industrial organizations. These system somehow lead to the better performance and high productivity in less time. Some of the examples of industrial applications are: Automated Car Assembly Plant, Chemical Plant etc.
2. Medical Science application:
In the field of medical science, real-time system has a huge impact on the human health and treatment. Due to the introduction of real-time system in medical science, many lives are saved and treatment of complex diseases has been turned down to easier ways. People specially related to medical, now feel more relaxed due to these systems. Some of the examples of medical science applications are: Robot, MRI Scan, Radiation therapy etc.
3. Peripheral Equipment applications:
Real-time system has made the printing of large banners and such things very easier. Once these systems came into use, the technology world became more strong. Peripheral equipment are used for various purposes. These systems

are embedded with micro chips and perform accurately in order to get the desired response. Some of the examples of peripheral equipment applications are: Laser printer, fax machine, digital camera etc.
4. Telecommunication applications:
Real-time system map the world in such a way that it can be connected within a short time. Real-time systems have enabled the whole world to connect via a medium across internet. These systems make the people connect with each other in no time and feel the real environment of togetherness. Some examples of telecommunication applications of real-time systems are: Video Conferencing, Cellular system etc.
5. Defense applications:
In the new era of atomic world, defense is able to produce the missiles which have the dangerous powers and have the great destroying ability. All these systems are real-time system and it provides the system to attack and also a system to defend. Some of the applications of defense using real time systems are: Missile guidance system, anti-missile system, Satellite missile system etc.
6. Aerospace applications:
The most powerful use of real time system is in aerospace applications. Basically hard real time systems are used in aerospace applications. here the delay of even some nano second is not allowed and if it happens, system fails. Some of the applications of real-time systems in aerospace are: Satellite tracking system, Avionics, Flight simulation etc.

CASE STUDIES IN SENTIMENTAL ANALYSIS:

Sentiment analysis is the process of classifying whether a block of text is positive, negative, or, neutral. Sentiment analysis is contextual mining of words which indicates the social sentiment of a brand and also helps the business to determine whether the product which they are manufacturing is going to make a demand in the market or not. The goal which Sentiment analysis tries to gain is to analyze people's opinion in a way that it can help the businesses expand. It focuses not only on polarity (positive, negative & neutral) but also on emotions (happy, sad, angry, etc.). It uses various Natural Language Processing algorithms such as Rule-based, Automatic, and Hybrid.
For example, if we want to analyze whether a product is satisfying customer requirements, or is there a need for this product in the market? We can use sentiment analysis to monitor that product's reviews. Sentiment analysis is also efficient to use when there is a large set of unstructured data, and we want to classify that data by automatically tagging it.Net Promoter Score (NPS) surveys are used extensively to gain knowledge of how a customer perceives a product or service. Sentiment analysis also gained its popularity due to its feature to process large volumes of NPS responses and obtain consistent results quickly.



Why perform Sentiment Analysis?
According to the survey,80% of the world's data is unstructured. The data needs to be analyzed and be in a structured manner whether it is in the form of emails, texts, documents, articles, and many more.

1. Sentiment Analysis is required as it stores data in an efficient, cost-friendly.
2. Sentiment analysis solves real-time issues and can help you solve all the real-time scenarios.

Types of Sentiment Analysis

1. **Fine-grained sentiment analysis:** This depends on the polarity based. This category can be designed as very positive, positive, neutral, negative, very negative. The rating is done on the scale 1 to 5. If the rating is 5 then it is very positive, 2 then negative and 3 then neutral.
2. **Emotion detection:** The sentiment happy, sad, anger, upset, jolly, pleasant, and so on come under emotion detection. It is also known as a lexicon method of sentiment analysis.

3. **Aspect based sentiment analysis:** It focuses on a particular aspect like for instance, if a person wants to check the feature of the cell phone then it checks the aspect such as battery, screen, camera quality then aspect based is used.
4. **Multilingual sentiment analysis:** Multilingual consists of different languages where the classification needs to be done as positive, negative, and neutral. This is highly challenging and comparatively difficult.

How does Sentiment Analysis work?

There are three approaches used:

1. **Rule-based approach:** Over here, the lexicon method, tokenization, parsing comes in the rule-based. The approach is that counts the number of positive and negative words in the given dataset. If the number of positive words is greater than the negative words then the sentiment is positive else vice-versa.
2. **Automatic Approach:** This approach works on the machine learning technique. Firstly, the datasets are trained and predictive analysis is done. The next process is the extraction of words from the text is done. This text extraction can be done using different techniques such as Naive Bayes, Linear Regression, Support Vector, Deep Learning like this machine learning techniques are used.
3. **Hybrid Approach:** It is the combination of both the above approaches i.e. rule-based and automatic approach. The surplus is that the accuracy is high compared to the other two approaches.

Applications

Sentiment Analysis has a wide range of applications as:

1. Social Media: If for instance the comments on social media side as Instagram, over here all the reviews are analyzed and categorized as positive, negative, and neutral.
2. Customer Service: In the play store, all the comments in the form of 1 to 5 are done with the help of sentiment analysis approaches.
3. Marketing Sector: In the marketing area where a particular product needs to be reviewed as good or bad.
4. Reviewer side: All the reviewers will have a look at the comments and will check and give the overall review of the product.

Challenges of Sentiment Analysis

There are major challenges in sentiment analysis approach:

1. If the data is in the form of a tone, then it becomes really difficult to detect whether the comment is pessimist or optimist.
2. If the data is in the form of emoji, then you need to detect whether it is good or bad.
3. Even the ironic, sarcastic, comparing comments detection is really hard.
4. Comparing a neutral statement is a big task.

STOCK MARKET PREDICTIONS:

Big data has proven to be a valuable tool for predicting stock market trends by analyzing large datasets, such as financial news reports, macroeconomic data, and market sentiment indicators. Big data can help identify patterns and correlations between various data points to help investors make more informed decisions about their investments.

By leveraging machine learning algorithms, big data can provide insights into market trends and help predict future stock prices. Additionally, big data can help investors identify potential risks and opportunities in the markets.

With the right combination of data points and analysis, investors can develop models that can accurately predict stock market trends.

What Is Big Data?

Big data is a term used to describe the large amount of data that is collected, stored, and analyzed to gain insights and make better decisions. Big data is typically collected from multiple sources such as social media, web analytics, user behavior, and machine-generated data.

This data is then analyzed to extract useful insights about trends, customer preferences, and other valuable information.

How Can Big Data Be Used to Predict Stock Market Trends?

Big data can be used to predict stock market trends by leveraging data from a variety of sources to identify patterns and correlations. By analyzing the data, it can be determined which stocks are likely to outperform or underperform in the market.

This data can then be used to make informed decisions about which stocks to buy, sell or hold.

What Types of Data Can Be Used?

When using big data to predict stock market trends, a variety of data sources can be used. This includes data from financial news and reports, market analysis, macroeconomic data, company financials, and social media.

Using Algorithms to Analyze Data

Once the data is collected, algorithms can be used to analyze the data and identify patterns. These algorithms can be used to identify correlations between different factors such as market performance, economic conditions, and company performance.

By analyzing the data, it can be determined which stocks are likely to outperform or underperform in the market.

Using Machine Learning for Predictive Analytics

Machine learning is used to develop predictive models that can be used to forecast future stock prices. By leveraging data from multiple sources, machine learning can be used to identify patterns and correlations that can be used to make informed decisions.

The Benefits of Using Big Data to Predict Stock Market Trends

Using big data to predict stock market trends can provide a number of benefits. By leveraging data from multiple sources, it can be used to identify patterns and correlations that can provide valuable insights. Additionally, it can be used to develop predictive models that can be used to make more informed decisions about which stocks to buy, sell or hold.

Conclusion

Big data can be used to predict stock market trends by leveraging data from a variety of sources to identify patterns and correlations. Algorithms and machine learning can be used to analyze the data and develop predictive models that can be used to make more informed decisions.

By using big data to predict stock market trends, investors can gain valuable insights that can help them make more informed decisions.