Module-2

Introduction to Stream Computing

Introduction

Refers to a sequence of data elements or symbols made available over time

• Data stream transmits from a source and receives at the processing end in a network

• A continuous stream of data flows between the source and receiver ends, and which is processed in real-time

Examples of Data Stream Applications

1. Making data-driven marketing decisions in real time. It requires the use of data from trends analyses of real-time sales, and analysis of social media, and the sales distribution.

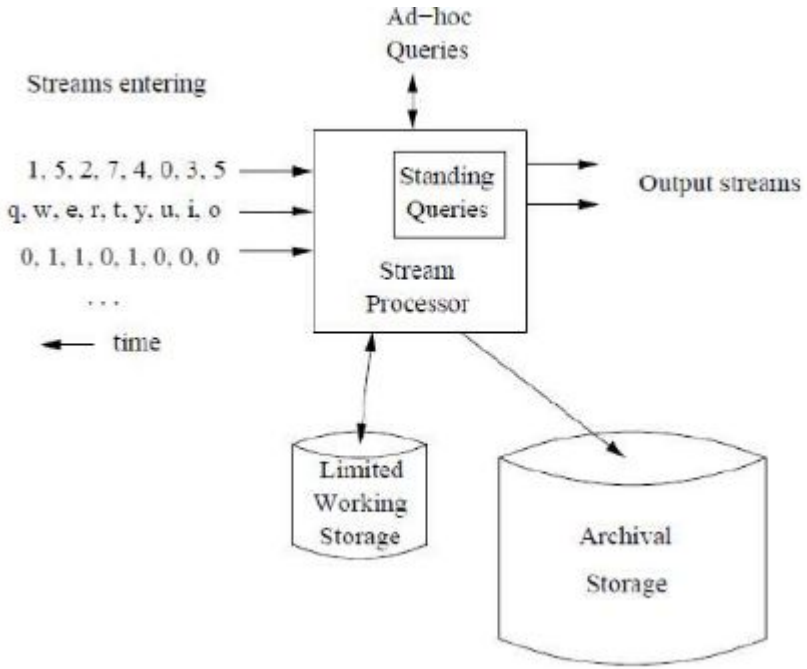
2. Monitoring and detection of potential failures of system using network management tool

3. Monitoring of industrial or manufacturing machinery in real time

4. A sensor network or IoT controlled by another entity, or a set of entities

5. Watching online video lectures, and rewinding or forwarding them

STREAM DATA MODEL



Examples of Stream Sources:

1. Sensor data
2. Image Data
3. Internet and web traffic

Stream Queries

Standing queries

Eg:

1. Maximum temperature of ocean

2. Is temperature greater than 25

Ad-hoc queries

Storing the data in a sliding window

- Most recent n elements

- Arrived with in t time.

Eg:

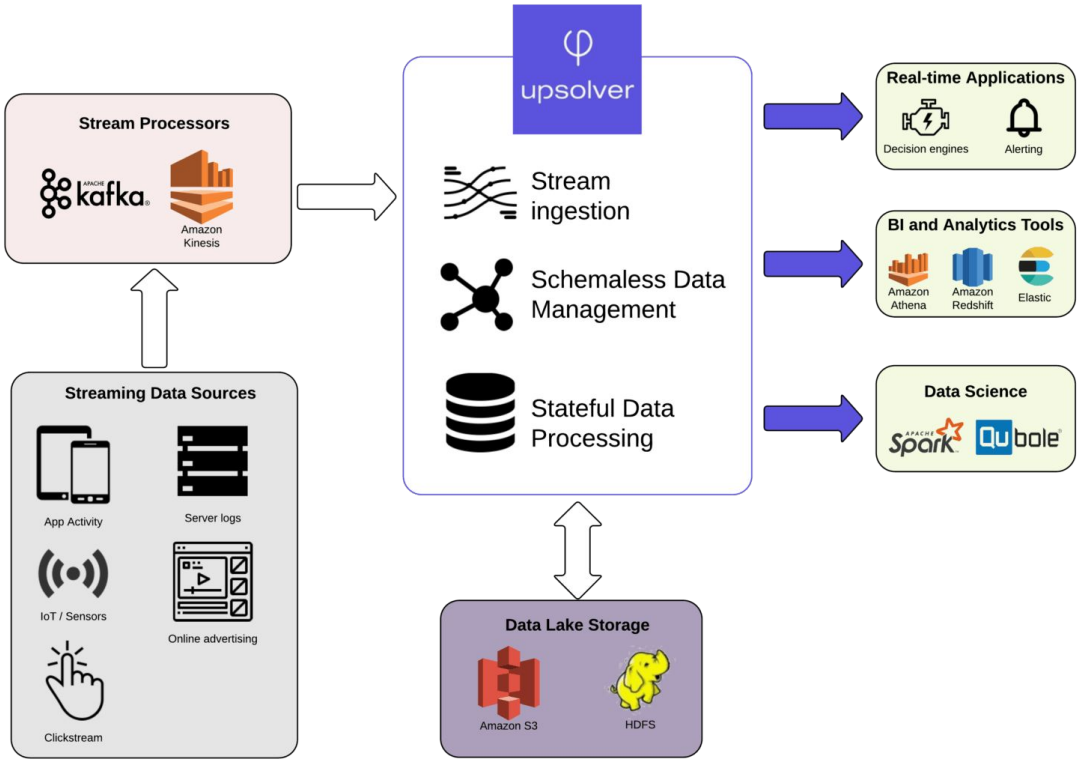
1. Log-in details of last one month

   ```
   SELECT COUNT (DISTINCT(name))
   FROM logins
   WHERE time>=t;
   ```

Issues in stream Processing

- Stream often deliver elements very rapidly.
- Efficient algorithm is needed.
- Important to store the data.
- Data can be complex types- Documents, Images, time series.

Stream Data Architecture



Components

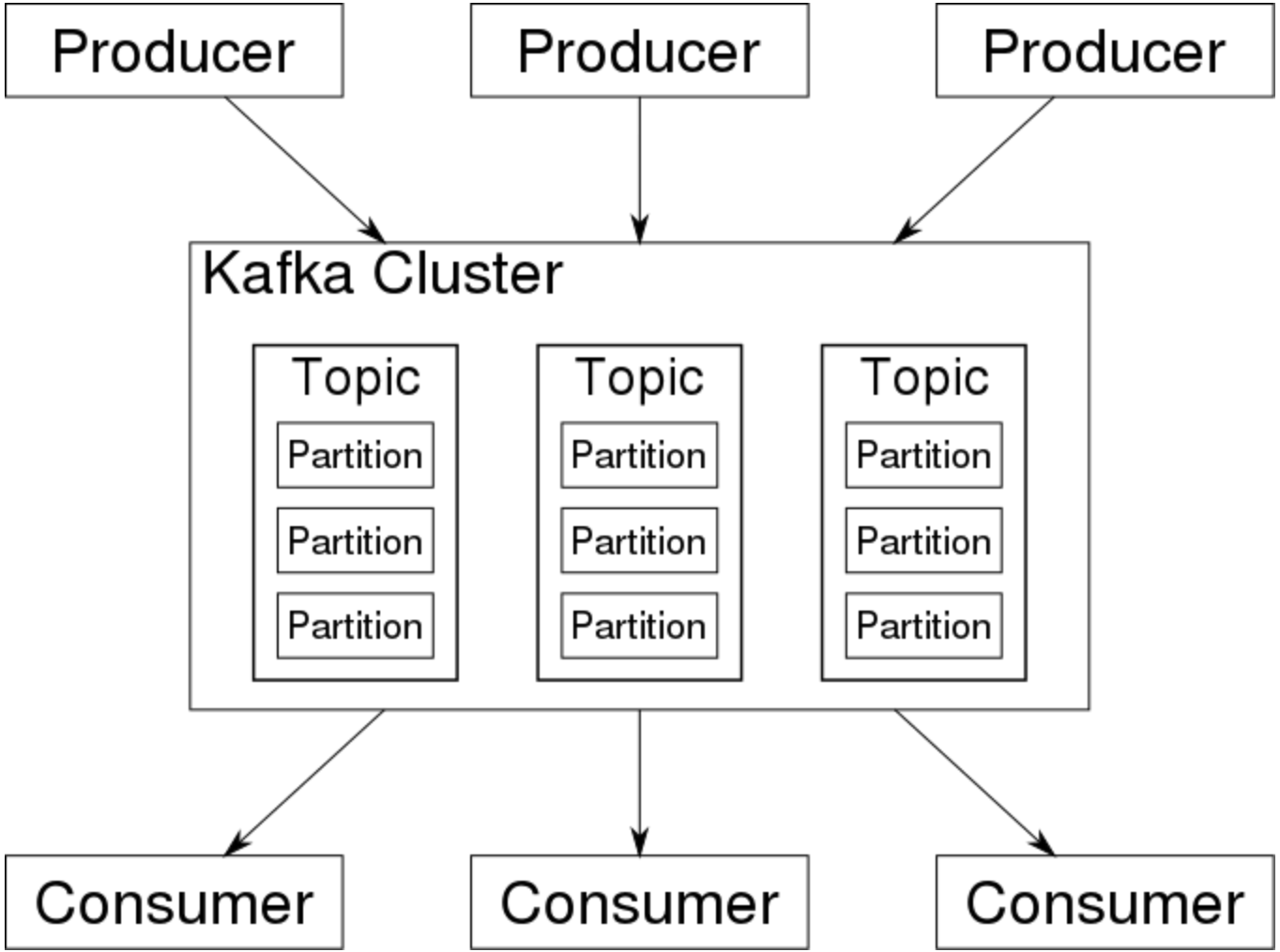
components includes tools for real-time processing, data manipulation, and analytics.

1. The Message Broker / Stream Processor

This is the element that takes data from a source, called a *producer*, translates it into a standard message format, and streams it on an ongoing basis. Other components can then listen in and consume the messages passed on by the broker.

The first generation of message brokers, such as RabbitMQ and Apache ActiveMQ, relied on the Message Oriented Middleware (MOM) paradigm.

Two popular stream processing tools are Apache Kafka and Amazon Kinesis Data Streams.


Producer
Producer
Producer
Kafka Cluster
Topic
Partition
Partition
Partition
Topic
Partition
Partition
Partition
Topic
Partition
Partition
Partition
Consumer
Consumer
Consumer

2. Batch and Real-time ETL Tools

Data streams from one or more message brokers must be aggregated, transformed, and structured before data can be analyzed with SQL-based analytics tools.

This would be done by an ETL tool or platform that receives queries from users, fetches events from message queues, then applies the query to generate a result

Extract- Transform- Load

the process often performing additional joins, transformations, or aggregations on the data. The result may be an API call, an action, a visualization, an alert, or in some cases a new data stream.

3. Data Analytics / Serverless Query Engine

There are many different approaches to streaming data analytics.

Analytics Tool	Streaming Use Case	Example Setup
Cassandra	Low latency serving of streaming events to apps	Kafka streams can be processed and persisted to a Cassandra cluster. You can implement another Kafka instance that receives a stream of changes from Cassandra and serves them to applications for real-time decision making.

Analytics Tool	Streaming Use Case	Example Setup
Amazon Athena	Distributed SQL engine	Streaming data is saved to S3. You can set up ad hoc SQL queries via the AWS Management Console, Athena runs them as serverless functions and returns results.
Amazon Redshift	Data warehouse	Amazon Kinesis Streaming Data Firehose can be used to save streaming data to Redshift. This enables near real-time analytics with BI tools and dashboards you have already integrated with Redshift.
Elasticsearch	Text search	Kafka Connect can be used to stream topics directly into Elasticsearch. If you use the Avro data format and a schema registry, Elasticsearch mappings with correct data types are created automatically. You can then perform rapid text search or analytics within Elasticsearch.

Streaming Data Storage

Streaming Data Storage Option	Pros	Cons
In a database or data warehouse – for example, PostgreSQL or Amazon Redshift	Easy SQL-based data analysis.	Hard to scale and manage. If cloud-based, storage is expensive.
In the message broker – for example, using Kafka persistent storage	Agile, no need to structure data into tables. Easy to set up, no additional components.	Data retention is an issue since Kafka storage is up to 10x more expensive compared to data lake storage. Kafka performance is best for reading recent (cached) data.
In a data lake – for example, Amazon S3	Agile, no need to structure data into tables. Low cost storage.	High latency, makes real time analysis difficult. Difficult to perform SQL analytics.

Stream Filtering.

- Accept those tuples in the stream that meet a criterion.
- Other tuples are dropped.

Filtering Streams - Examples

1. Web crawling - Based on user interest.
2. Spam Mail settings.
3. Only storing required data

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 bloom filter consists of

1. An array of N bits initially all 0s
2. Collection of hash functions h1,h2,h3… Each maps “key” values to N buckets.
3. A set S of M key values.

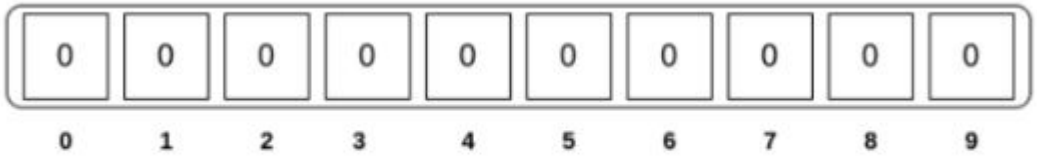
Operation

Initialize bit array with 0s, Take each key value in S and perform hi(k) and set that bit to 1

Check that all h1(k), h2(k), h3(k)

Suppose we use k hash functions, h1, h2, ..., hk, pass two strings through these hash functions, and compute their hashes. If all the hashes are identical for both strings, there is a very high probability that both strings are the same. On the other hand, even if one hash does not match, we can say that the strings are different.

Bloom filter is an array that stores 0s and 1s. In the image below, each cell represents a bit. The number below the bit is its index for a bloom filter of size 10.



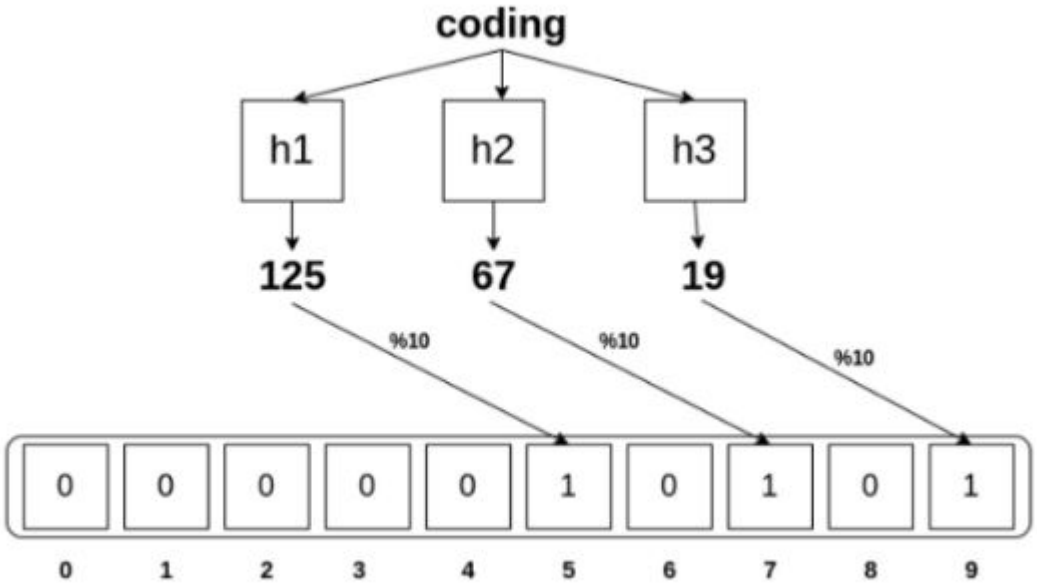
Empty Bloom Filter

In order to add an element, we need to pass it through k hash functions. Bits are set at the index of the hashes in array. For example, we want to add the word “coding”. After passing it through three hash functions, we get the following results.

h1(“coding”) = 125

h2(“coding”) = 67

h3(“coding”) = 19



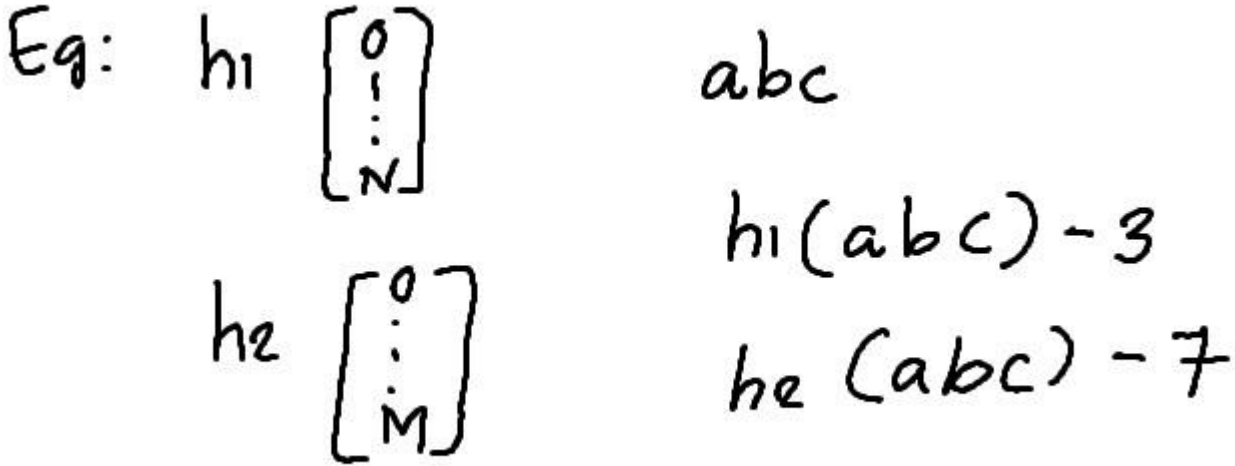
Inserting in bloom filter

Testing membership of an item in Bloom filter

If we want to test the membership of an element, we need to pass it through same hash functions. If bits are already set for all these indexes, then this element might exist in the set. However, even if one index is not set, we are sure that this element is not present in the set.

Working

Hash Map



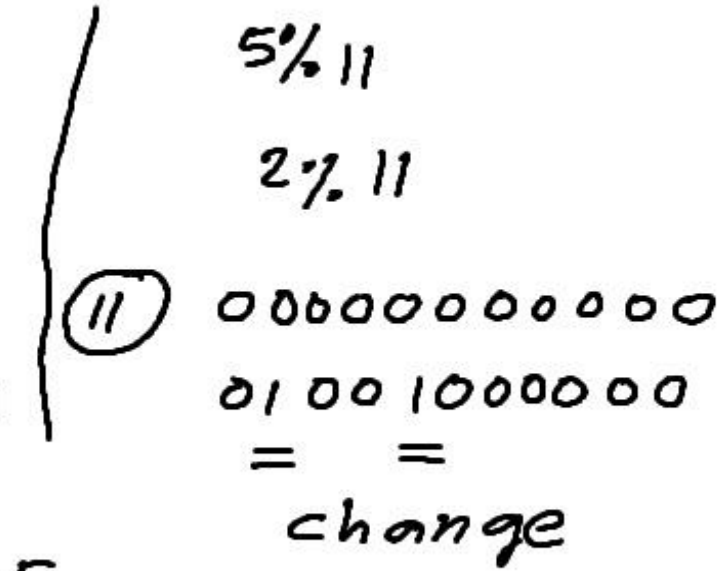
Bloom Filter

N - Array of 11 in size

25 → Input

h1 → Count odd position

h2 → Count even position



11001 → 25

h1 → 101 → 5

h2 → 10 → 2

Count Distinct Problem.

Measuring the number of distinct elements from a stream of values is one of the most common utilities that finds its application in the field of Database Query Optimizations, Network Topology, Internet Routing, Big Data Analytics, and Data Mining.

Application

- How many different words are found in a web page.
- How many different pages does a customer request.
- Recent logins
- Occurrence of same data in a stream.

The Flajolet-Martin Algorithm

-It is possible to estimate the number of distinct elements by hashing the elements of universal set to a bit string that is sufficiently long.

-Important properties of has function is that, when applied to the same element it always produces the same result.

Procedure

Pick a hash function H that maps each of N elements to log2 (N) bits.

- For each stream element a, let r(a) be the number of trailing 0s in H(a).
- r(a) <- position of first one counting from right.
- Record R as maximum r(a) seen.
- Estimated number of distinct elements = 2^R

Stream: 4, 2, 5 ,9, 1, 6, 3, 7

h(x) = x + 6 mod 32

h(4) = (4) + 6 mod 32 = 10 mod 32 = 10 = (01010)

h(2) = (2) + 6 mod 32 = 8 mod 32 = 8 = (01000)

h(5) = (5) + 6 mod 32 = 11 mod 32 = 11 = (01011)

h(9) = (9) + 6 mod 32 = 15 mod 32 = 15 = (01111)

h(1) = (1) + 6 mod 32 = 7 mod 32 = 7 = (00111)

h(6) = (6) + 6 mod 32 = 12 mod 32 = 12 = (01110)

h(3) = (3) + 6 mod 32 = 9 mod 32 = 9 = (01001)

h(7) = (7) + 6 mod 32 = 13 mod 32 = 13 = (01101)

Trailing zero's {1, 3, 0, 0, 0, 1, 0, 0}

R = max [Trailing Zero] = 3 Output = 2^R = 2^3= 8

Estimating Moments

Counting Ones in a window

- How many 1s are there in the last k bits of window with size N, where k<=N
-