Data Engineering - II
Course code: 1TD075 62033

M1 - Data Stream Processing (Part-I)

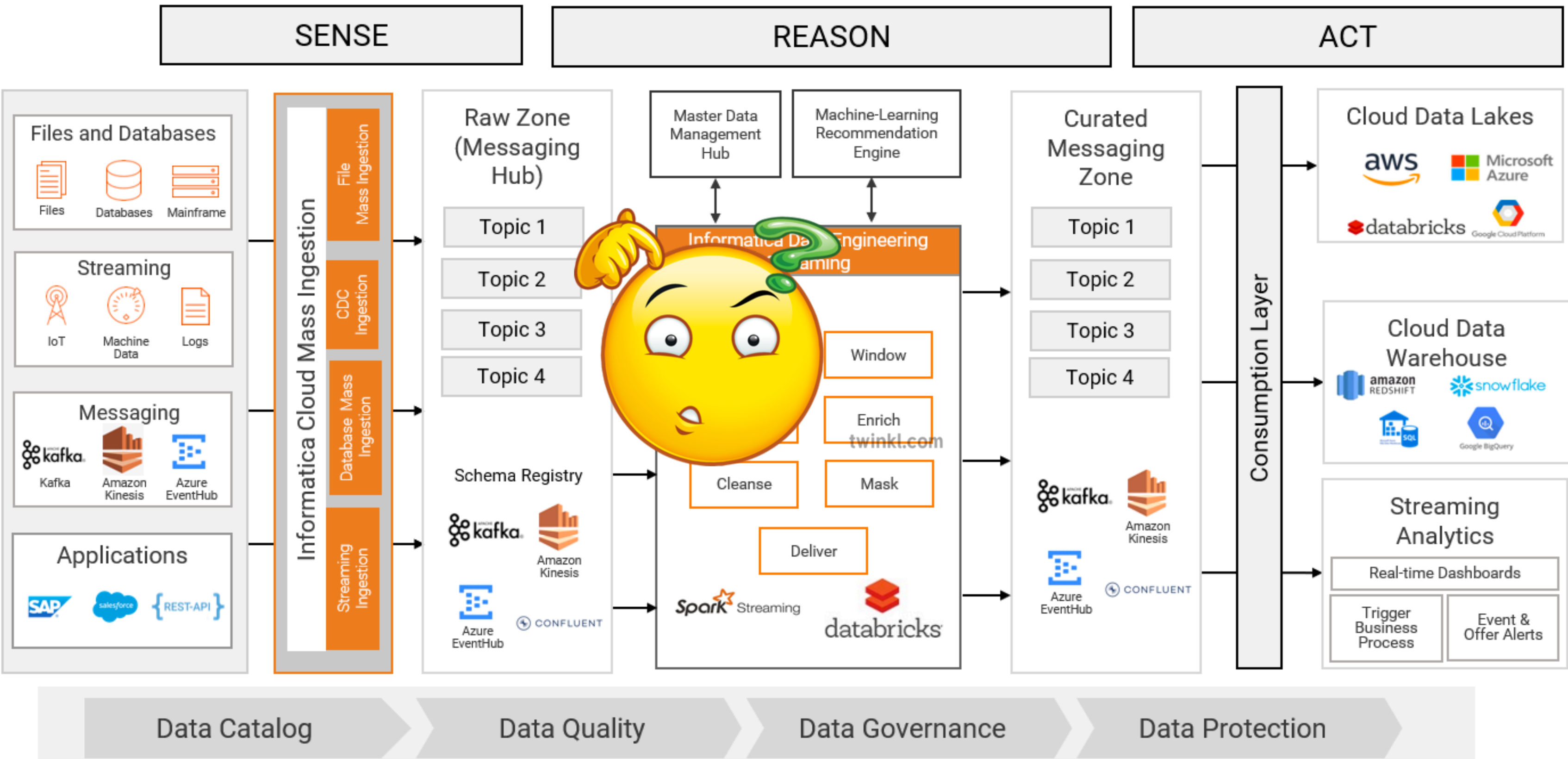
Salman Toor
salman.toor@it.uu.se

UPPSALA UNIVERSITET

Introduction

- Data stream processing
- Differences between batch, micro-batch and stream processing
- Components and conventional architectures for data stream processing
- Streaming frameworks
- Performance metrics for streaming frameworks

Data stream processing



https://blogs.informatica.com/2020/05/13/adopt-a-kappa-architecture-for-streaming-and-ingesting-data/

Why we need stream processing?

- We do not just have people who are entering information into a computer. Instead, we have computers entering data into each other
- Nowadays, number of applications generate continuous data streams instead of as persistent tables
- The traditional data management solutions are inadequate to address the requirements of continuous queries by modern applications

https://link.springer.com/referenceworkentry/10.1007%2F978-3-319-77525-8_188
https://link.springer.com/content/pdf/10.1007/3-540-73679-4_3.pdf

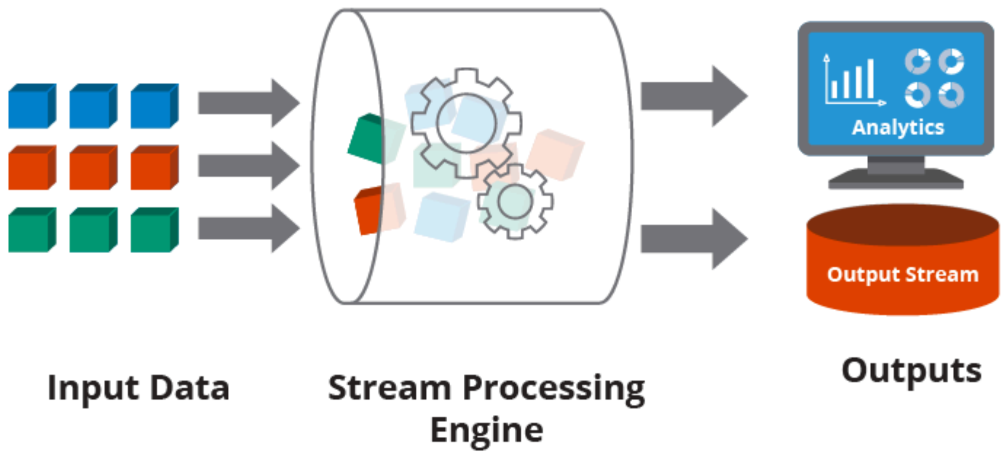
Data stream processing

- A **data stream** is an ordered sequence of instances that can be read only once or a small number of times using limited computing and storage capabilities

- **Stream processing** refers to analyzing data streams on-the-fly to produce new results as new input data becomes available

http://cloudbus.org/papers/C11-Stream-SA.pdf
https://link.springer.com/referenceworkentry/10.1007%2F978-3-319-77525-8_188
https://link.springer.com/content/pdf/10.1007/3-540-73679-4_3.pdf

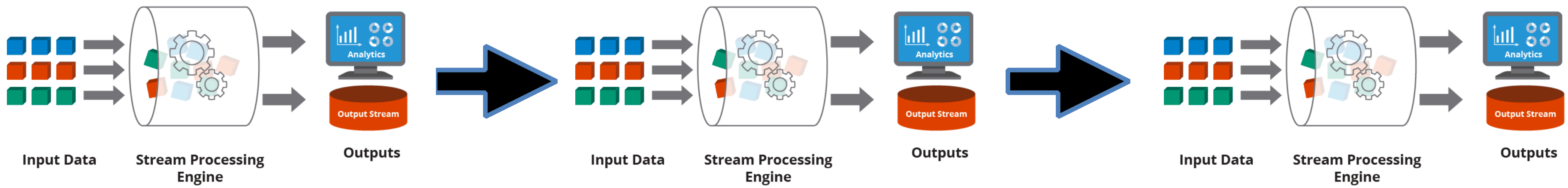
Data stream processing



- Data sources
  - sensor networks
  - wireless networks
  - customer click streams
  - multimedia data
  - scientific data
- Data source characteristics
  - open-ended
  - flowing at high-speed
  - non stationary distributions in dynamic environments

https://hazelcast.com/glossary/stream-processing/

Data stream processing



- Data sources
  - sensor networks
  - wireless networks
  - customer click streams
  - multimedia data
  - scientific data
- Data source characteristics
  - open-ended
  - flowing at high-speed
  - non stationary distributions in dynamic environments

https://hazelcast.com/glossary/stream-processing/

Data stream models

- Three basic models

  - **Insert Only Model**: once an element a_i is seen, it cannot be changed
  - **Insert-Delete Model**: elements a_i can be deleted or updated
  - **Accumulative Model**: each a_i is an increment to $A[j] = A[j-1] + a_i$. Where $a_1, a_2, \ldots, a_j, \ldots$ arrive sequentially, item by item

https://link.springer.com/referenceworkentry/10.1007%2F978-3-319-77525-8_188

https://link.springer.com/content/pdf/10.1007/3-540-73679-4_3.pdf

UPPSALA UNIVERSITET

Stream processing

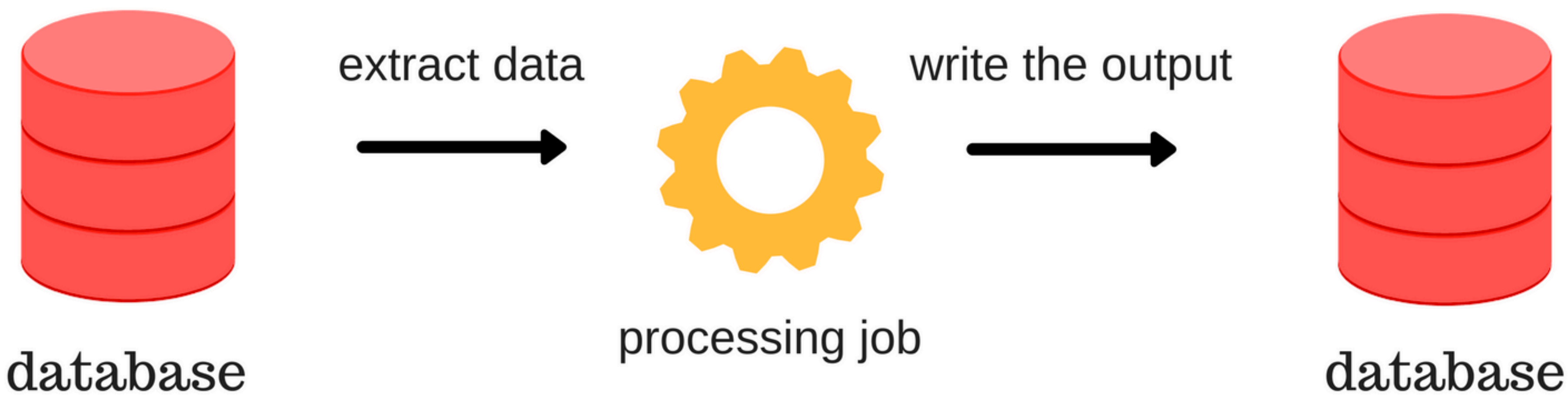
Stream processing refers to analyzing data streams on-the-fly to produce new results as new input data becomes available

- Challenges
  - It is not possible to find the exact solution for the functions such as min or max
  - All blocking operations are difficult to execute in stream processing

- Active research directions
  - Approximate query processing techniques
  - Novel architectures for blocking operators
  - Reliable processing and management of high coming data rate

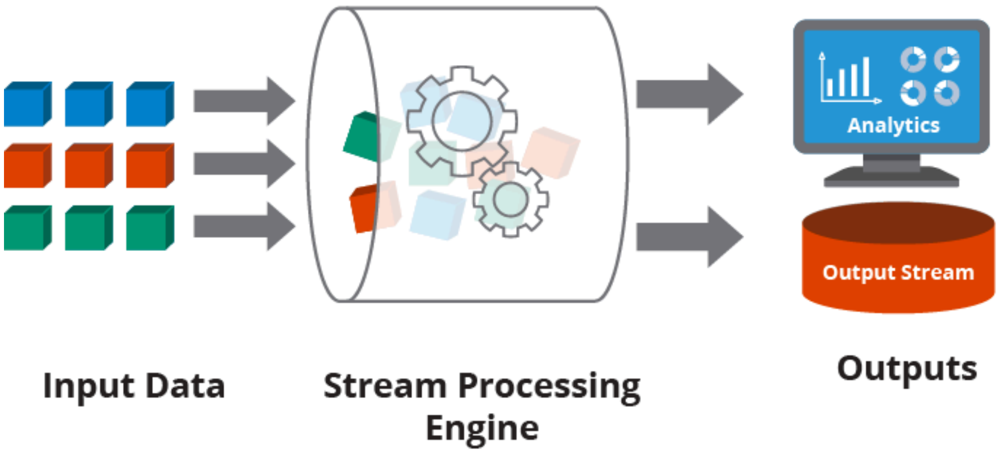
Difference between batch, micro-batch and stream processing

- Batch processing
  - Ad-hoc or scheduled processing
  - requires large datasets



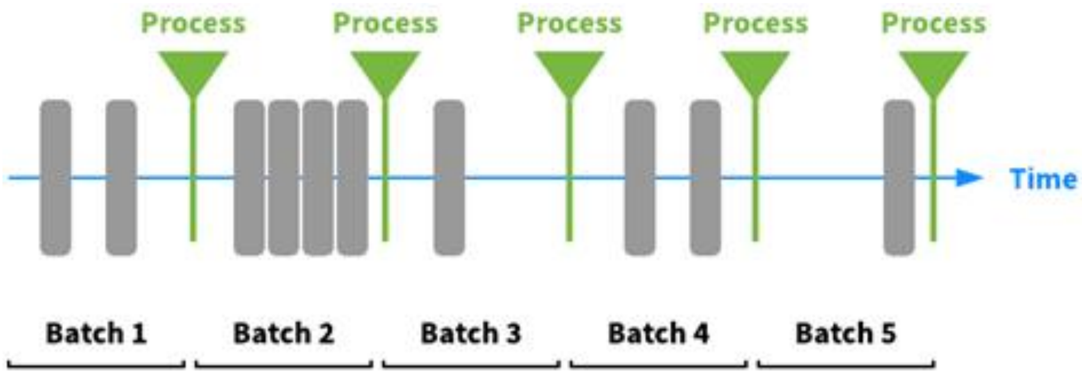
Difference between batch, micro-batch and stream processing

- Stream processing
  - Realtime processing
  - continuous data



Difference between batch, micro-batch and stream processing

- Mini or micro batch processing (Best of two worlds)
  - close to realtime processing on mini/micro batches
  - continuous collection of data in batches

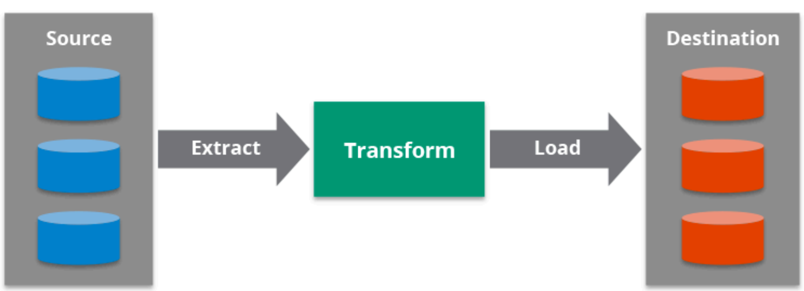


https://intellipaat.com/blog/a-guide-to-apache-spark-streaming-tutorial/
https://livebook.manning.com/book/pulsar-in-action/chapter-1/v-4/41

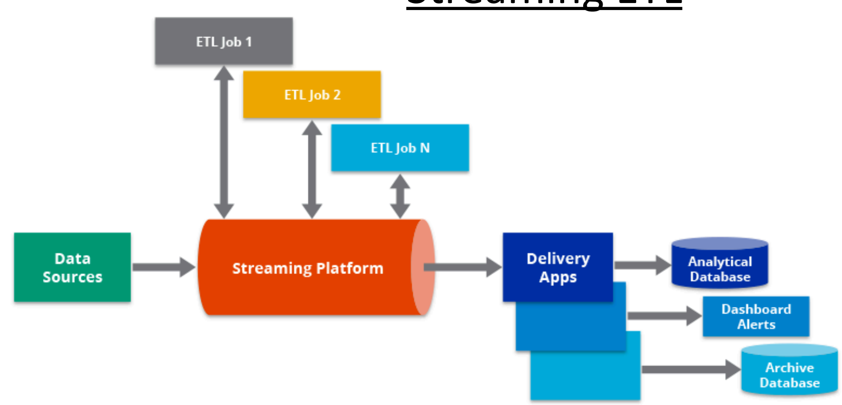
Streaming ETL

- ETL is an abbreviation of Extract, Transform and Load operations
    - **Extract** - collecting data from sources
    - **Transform** - any function/transformation performed on the data
    - **Load** - sending data to the next stage
- Equally valid for batch, micro-batch and stream processing

Batch ETL



Streaming ETL



https://hazelcast.com/glossary/streaming-etl/

Traditional vs stream processing engines

Table 3.1 Differences between traditional and stream data processing

	Traditional	Stream
Number of passes	Multiple	Single
Processing Time	Unlimited	Restricted
Memory Usage	Unlimited	Restricted
Type of Result	Accurate	Approximate
Distributed	No	Yes

https://link.springer.com/content/pdf/10.1007/3-540-73679-4_3.pdf

Types of stream processing

- Stateless stream processing

  Each object of the stream can process independently of other objects in the stream (preferable option to process data streams)

- Stateful stream processing

  The operation on each object in stream gets affected by some common state and that state is also updated in return by each operation (High overhead, only used in special cases)

UPPSALA UNIVERSITET

CHAPTER

11

A TAXONOMY AND SURVEY OF STREAM PROCESSING SYSTEMS

Xinwei Zhao*, **Saurabh Garg***, **Carlos Queiroz**[†], **Rajkumar Buyya**[‡]

**School of Engineering and ICT, University of Tasmania, Tasmania, Australia* [†]*The Association of Computing Machinery (ACM), Singapore* [‡]*Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia*

http://cloudbus.org/papers/C11-Stream-SA.pdf

Basic components for data stream processing

Data Source – Producer

One or more sources of data, also known as producers. Producers are applications that continuously transmit data to the streaming framework

Basic components for data stream processing

Data consumer – Processor

Processor receives data streams from one or more message brokers and applies user-defined queries/ transformations to the data to prepare it for consumption and analysis

Basic components for data stream processing

Message Broker

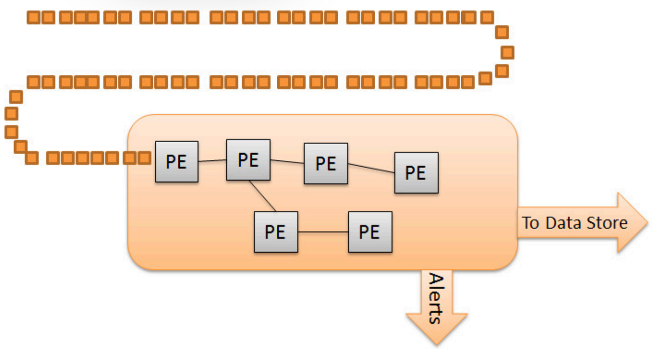
The message broker receives data from the producer and converts it into a standard message format and then publishes the messages in a continuous stream called *topics*

Basic components for data stream processing

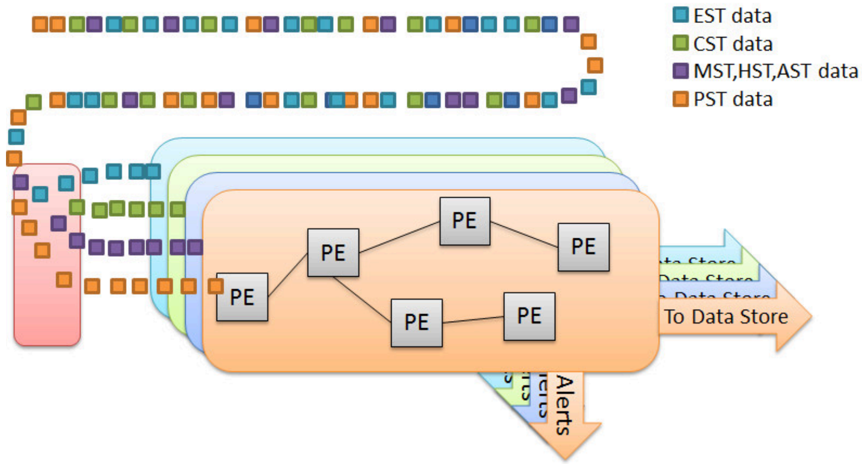
Topics

Topic is an abstraction provided to the incoming messages. It is a category or feed name to which data is published by producers and consumed by the processors

Stream processing



1. A simple stream processing setup



2. Partitioning Streams

- Important application design choices:
  - Avoid expensive external calls and dependencies
  - Avoid reprocessing historic events
  - Pull data immediately as they become available
  - Avoid complex time-consuming calculations as part of message processing
  - Discard unnecessary data
  - Carefully design topic partitions

https://www.linkedin.com/pulse/6-important-concepts-stream-processing-chandra-rajasekharaiah/

Role of timestamps in stream processing

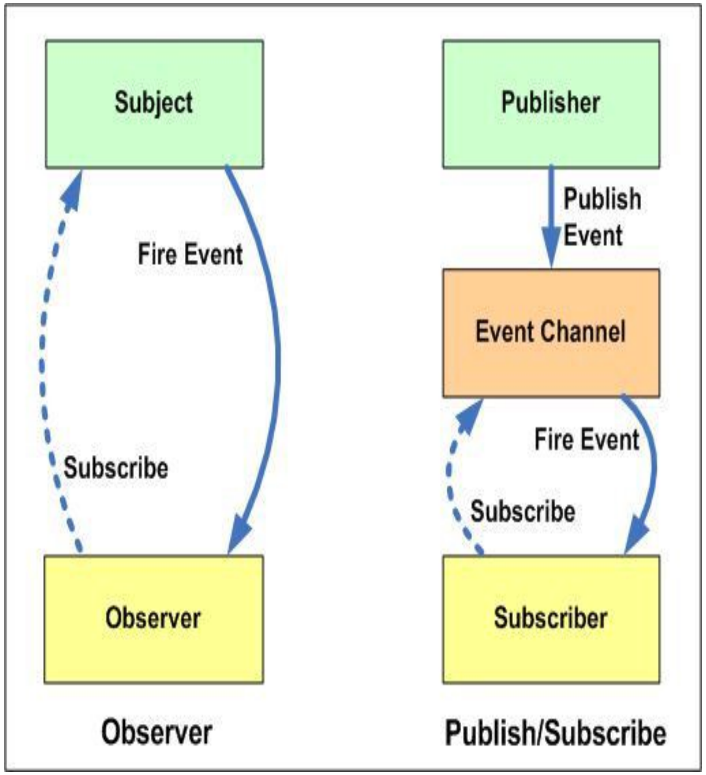
- **Event time** - Time when a message generated at the producing source
- **Processing time** - Time to process the incoming message from the producer
- **Ingestion time** - Time required for a message to reach the consumer

Different streaming frameworks realize timestamps differently

Design pattern for stream frameworks

Two types of subscriptions

- Ephemeral subscription
- Durable subscription



Streaming frameworks adhere to <u>publish-subscribe</u> design pattern

Log compaction

- Streaming frameworks heavily depend on logs

- Log compaction is a mechanism to reduce log size while guarantee availability of latest message state based on the primary key.

- Different frameworks adapt different strategies based on the formal guarantees and the application's needs

https://help.aiven.io/en/articles/3995429-topic-log-compaction

Partitioning and ordering for streaming frameworks

- Topic partitioning is one of the key features that provides robustness and scalability in streaming frameworks

  - Random partitioning

  - Partition by aggregate

  - Ordering partition

https://linuxhint.com/apache_kafka_partitions/

https://blog.newrelic.com/engineering/effective-strategies-kafka-topic-partitioning/

Replication strategies

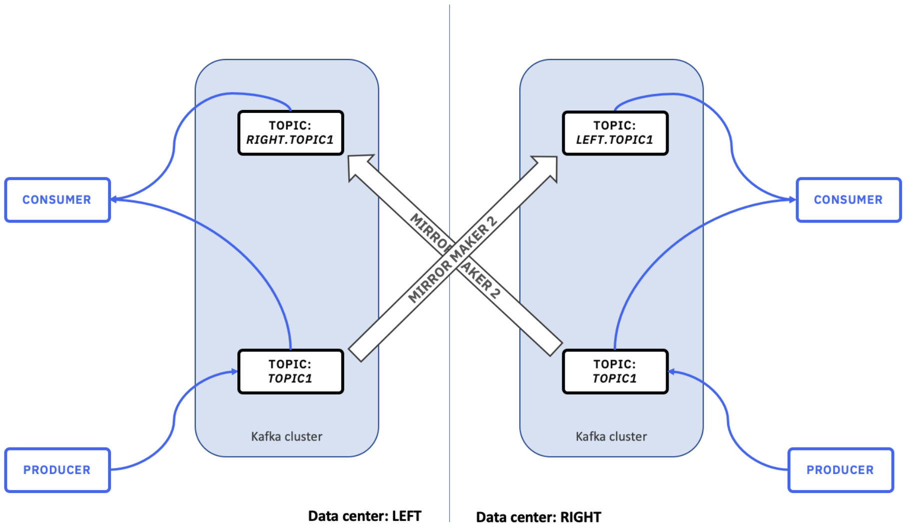
- Often in streaming frameworks, replication is implemented at partition levels

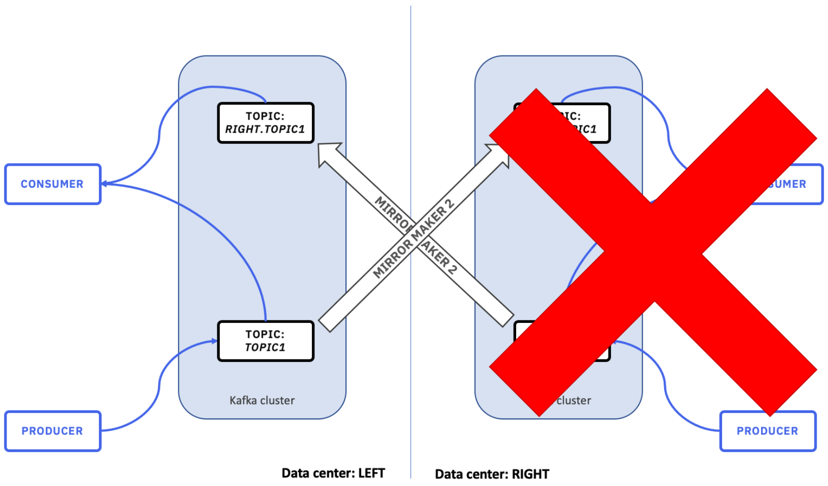
- Replication strategy realized by master-work model. Master holds the read/write permissions and workers follow the changes. In case master is unavailable, one of the workers becomes the leader

- Each new message generated by the producer only incremented in the partition by the master

https://www.tandfonline.com/doi/full/10.1080/17445760.2019.1585848

Geo-replication





- A mechanism to create high availability of data and services in a geographically distributed environment
- consists of two types:
  - Active-active geo-replication
  - Active-passive geo-replication

https://dalelane.co.uk/blog/?p=4074

Message brokers

APACHE kafka®

RabbitMQ

ØMQ

APACHE ACTIVEMQ™

redis

Data stream processing platforms

APACHE kafka®

samza

APACHE STORM

APACHE nifi

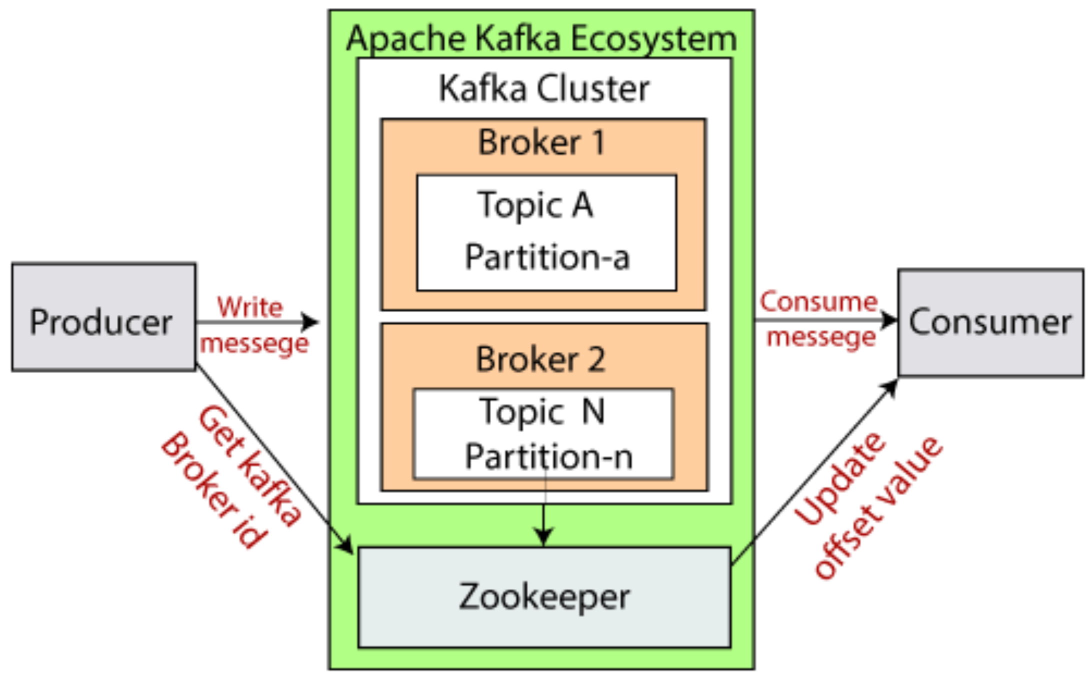
APACHE Spark™

Flink

PULSAR

Apache Kafka is an open-source distributed **event streaming** platform

- Components
  - Kafka Cluster
  - Producers
  - Consumers
  - Brokers
  - Topics
  - Partitions
  - ZooKeeper

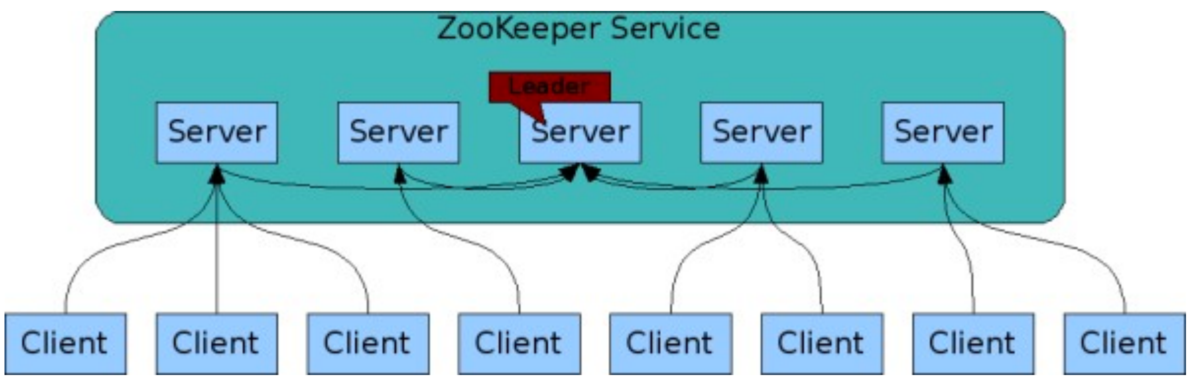


Apache Kafka Architecture

https://www.javatpoint.com/apache-kafka-architecture https://kafka.apache.org/

- Apache ZooKeeper - centralized service, used for maintaining configuration information, naming, and providing distributed synchronization

- Zookeeper data is kept in-memory, which means Zookeeper can achieve high throughput and low latency numbers



https://medium.com/@logeesan/zookeeper-in-kafka-ce31b3dd55b1

UPPSALA UNIVERSITET

APACHE kafka®

- Features
  - Low latency (upto 10 milliseconds)
  - High throughput (thousands of messages in a second)
  - Fault tolerance
  - Durability
  - Scalability

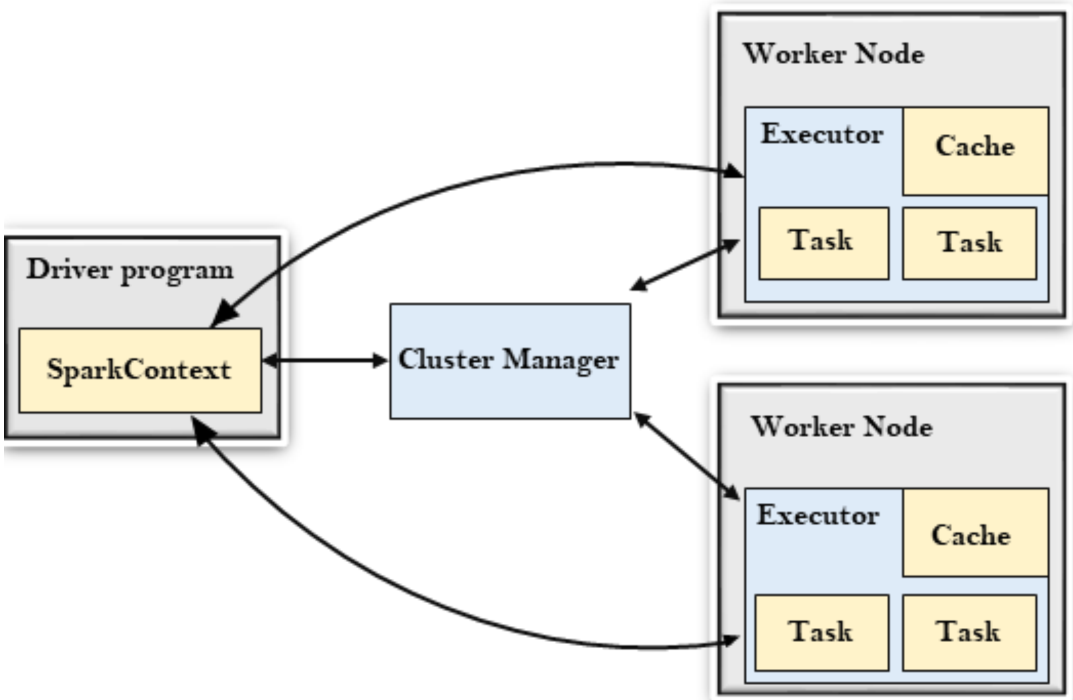
https://www.javatpoint.com/apache-kafka-advantages-and-disadvantages

- Apache Spark is a unified analytics engine for large-scale data processing
- Master-worker architecture
- Architecture depends on two abstractions:
  - Resilient Distributed Datasets (RDD)
  - Directed Acyclic Graph (DAG)
- Java based framework
- Support batch and stream processing

https://spark.apache.org/

UPPSALA UNIVERSITET

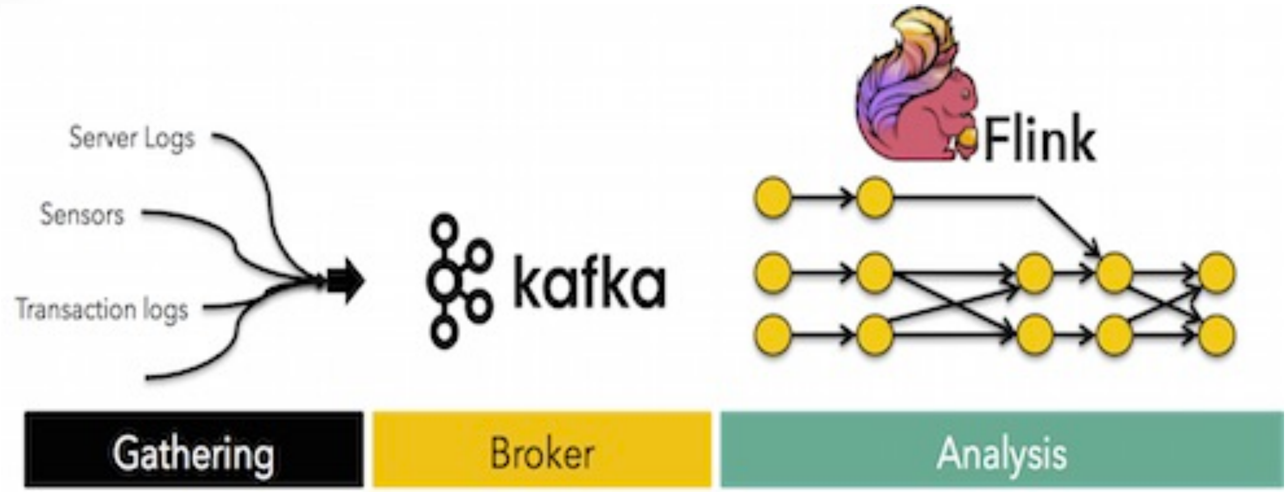
- Framework components
  - Driver program
  - Cluster Manager
  - Worker Nodes
  - Executors
  - Tasks



https://www.javatpoint.com/apache-spark-architecture

- Features
  - High performance setup both for batch and stream processing
  - Generality (SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming)
  - Fault tolerance
  - Lightweight
  - Scalability
  - Support both in-memory and HDFS based processing

Combination of different streaming frameworks



- Apache Flink supports batch analytics, continuous stream analytics, as well as machine learning and graph processing natively on top of a streaming engine
- Combination of different frameworks also allows to build highly scalable and robust processing platform

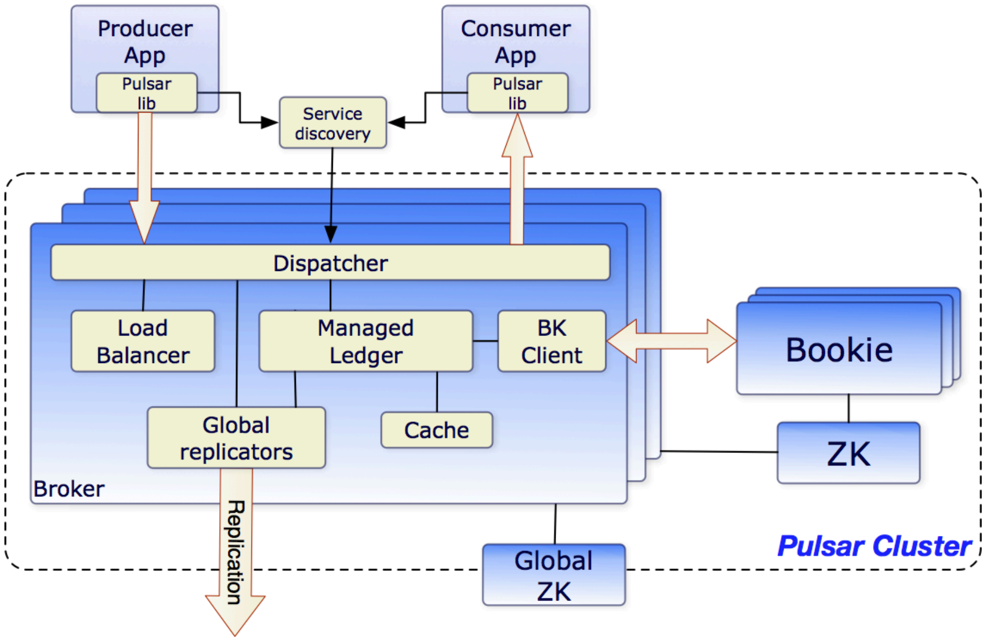
- Pulsar is a **multi-tenant**, **high-performance** solution for **server-to-server messaging**. Pulsar was originally developed by Yahoo, it is under the stewardship of the Apache Software Foundation

- Pulsar is built on the publish-subscribe pattern

- To support multi-tenancy, Pulsar has a concept of tenants. Tenants can be spread across clusters and can each have their own authentication and authorization scheme applied to them

https://pulsar.apache.org/en/

- Components
  - Clusters
    - Broker(s)
      - HTTP server
      - Dispatcher
    - Service discovery
    - Bookies (persistent store for messages)
    - ZooKeeper (metadata store)
    - BookKeeper instances (Bookies)



https://pulsar.apache.org/docs/en/concepts-architecture-overview/

Features

Pulsar Functions

Easy to deploy, lightweight compute process, developer-friendly APIs, no need to run your own stream processing engine.

Proven in production

Run in production at Yahoo! scale for over 5 years, with millions of messages per second across millions of topics.

Horizontally scalable

Expand capacity seamlessly to hundreds of nodes.

Low latency with durability

Low publish latency (< 5ms) at scale with strong durability guarantees.

Geo-replication

Configurable replication between data centers across multiple geographic regions.

Multi-tenancy

Built from the ground up as a multi-tenant system. Supports isolation, authentication, authorization and quotas.

Persistent storage

Persistent message storage based on Apache BookKeeper. IO-level isolation between write and read operations.

Client libraries

Flexible messaging models with high-level APIs for Java, Go, Python, C++, Node.js, WebSocket and C#.

Operability

REST Admin API for provisioning, administration, tools and monitoring. Can be deployed on bare metal, Kubernetes, Amazon Web Services(AWS), and DataCenter Operating System(DC/OS).

https://pulsar.apache.org/en/

Example

Producer.py

```
import pulsar

client = pulsar.Client('pulsar://localhost:6650')

producer = client.create_producer('my-topic')

for i in range(10):
    producer.send(('Hello-%d' % i).encode('utf-8'))

client.close()
```

https://pulsar.apache.org/docs/en/client-libraries-python/

UPPSALA UNIVERSITET

PULSAR

Consumer.py

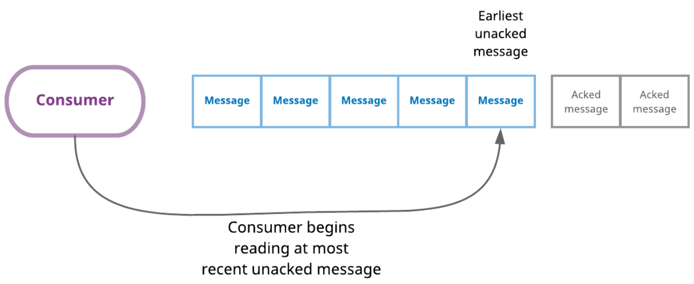
```python
consumer = client.subscribe('my-topic', 'my-subscription')

while True:
    msg = consumer.receive()
    try:
        print("Received message '{}' id='{}'".format(msg.data(), msg.message_id()))
        # Acknowledge successful processing of the message
        consumer.acknowledge(msg)
    except:
        # Message failed to be processed
        consumer.negative_acknowledge(msg)

client.close()
```

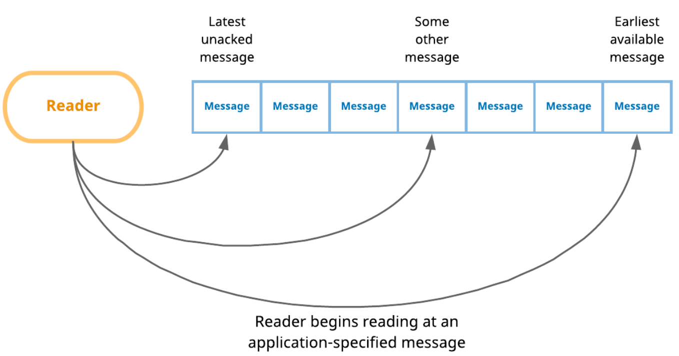
Consumer interface

Pulsar automatically manages topic cursors



Reader interface

Applications manually control topic cursors



Consumer_reader_interface.py

```python
# MessageId taken from a previously fetched message
msg_id = msg.message_id()

reader = client.create_reader('my-topic', msg_id)

while True:
    msg = reader.read_next()
    print("Received message '{}' id='{}'".format(msg.data(), msg.message_id()))
    # No acknowledgment
```

https://pulsar.apache.org/docs/en/client-libraries-python/

Performance metrics for stream processing

- Following are the important features that a production-grade data stream processing system should adhere to:

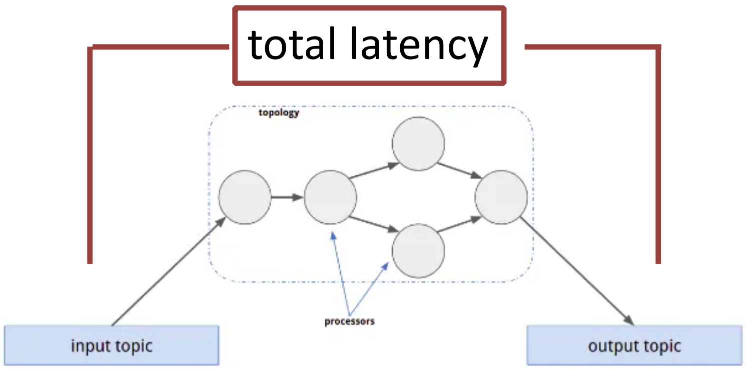
- Low latency
  - System latency
  - Information latency

- High throughput
- Efficient utilization of resources
- Robustness
- Scalability
- Security

https://www.diva-portal.org/smash/get/diva2:1440436/FULLTEXT01.pdf
https://arxiv.org/pdf/1802.08496.pdf https://cs.brown.edu/~ugur/8rulesSigRec.pdf

Performance metrics for stream processing

- **Latency** - the delay that occurs before a transfer of data begins following the stream toward its landing point
  - **Information latency** - the time it takes a system to traverse and process information
  - **System latency** - the delay cause by all the system components to ensure reliable and consistent end-to-end delivery of messages

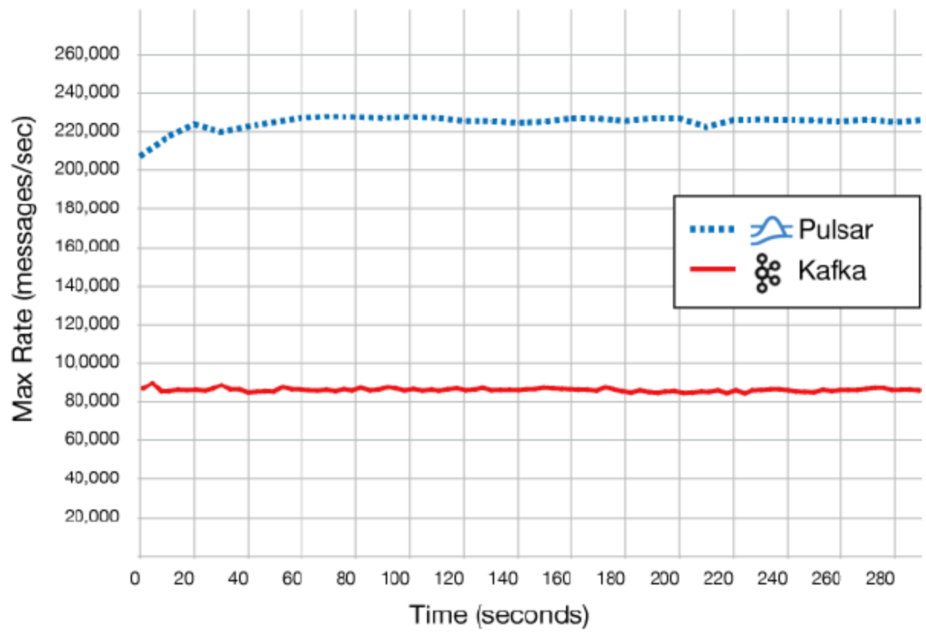


1. Number of processing units required by the system (system latency)
2. Amount of workload each process need to perform (information latency)

Low latency is an important feature of stream processing frameworks.

Performance metrics for stream processing

- **Throughput** - amount of data a framework can process



http://entradasoft.com/blogs/apache-pulsar-outperforms-apache-kafka-on-openMessaging-benchmark

https://engineering.linkedin.com/kafka/benchmarking-apache-kafka-2-million-writes-second-three-cheap-machines

Performance metrics for stream processing

- Efficient utilization of resources

- Maximum usage of hardware and network for the faster and reliable stream processing

https://www.confluent.io/blog/kafka-fastest-messaging-system/

Performance metrics for stream processing

- **Robustness** - the probability that the system does not experience many failures in a given time

- Mainly depends on two components:
  - System devices
  - System software components:
    - services
    - transfer protocols

- Often reliability adheres with the redundancy or replication strategies

Performance metrics for stream processing

- **Scalability** - the ability of a system to efficiently handle the growing amount of workload

- Horizontal scalability

  The system performs by add more resources
- Vertical scalability

  Here the focus is to increase the actual capacity of the available resources

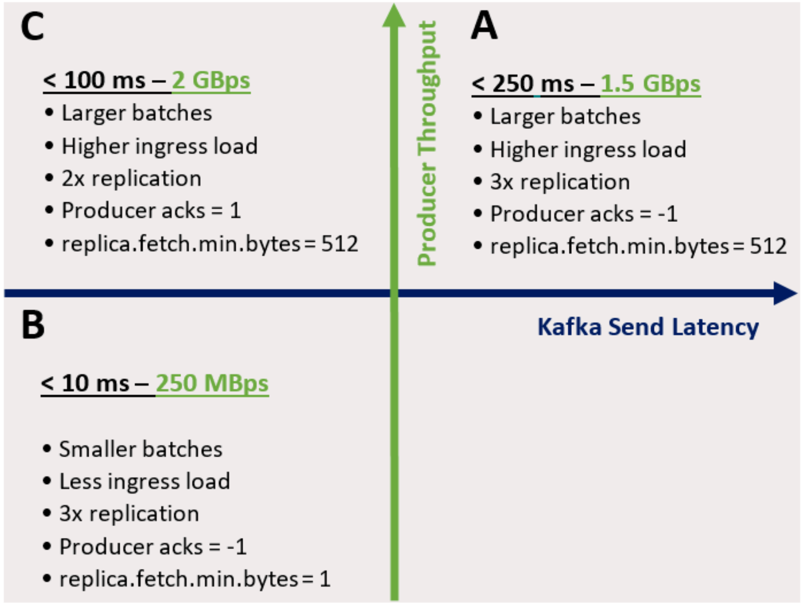
Performance metrics for stream processing

- Security
  - Based on four fundamental components:
    - Authentication
    - Authorization
    - Encryption
    - Access control
  - Requires at multiple level:
    - Client level
    - System level
    - Network level

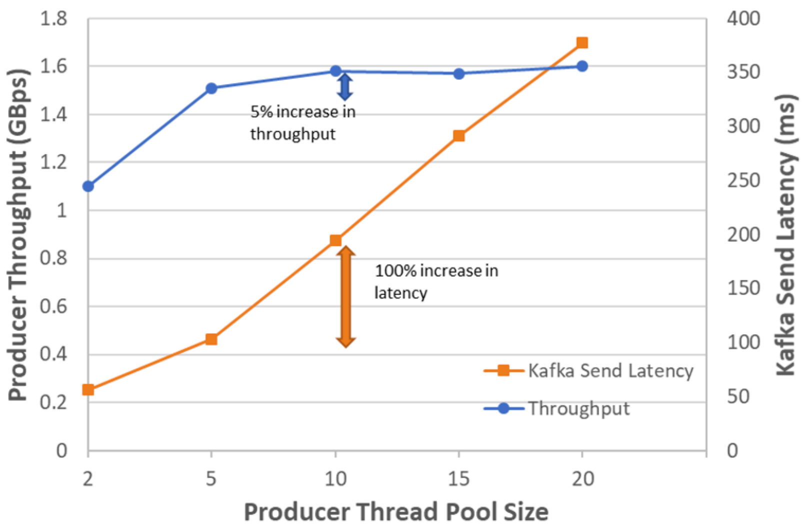
Performance of Kafka cluster at Microsoft Azure infrastructure

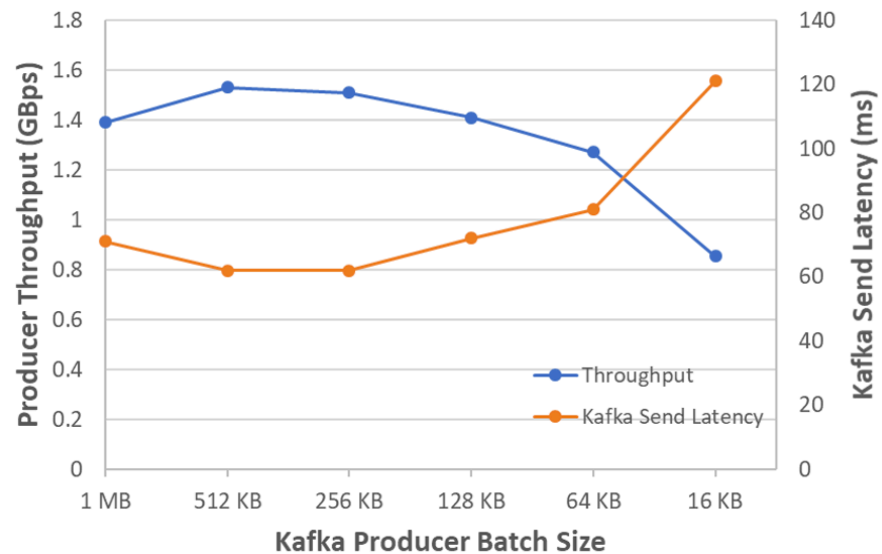
To build a compliant and cost-effective near real time publish-subscribe system that can ingest and process 3 trillion events per day from businesses like O365, Bing, Skype, SharePoint online, and more, we created a streaming platform called Siphon. Siphon is built for internal Microsoft customers on Azure cloud with Apache Kafka on HDInsight as its core component.

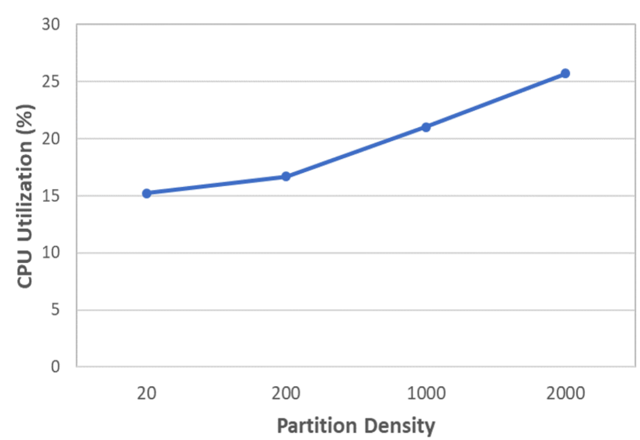
https://azure.microsoft.com/en-us/blog/processing-trillions-of-events-per-day-with-apache-kafka-on-azure/



Performance of Kafka cluster at Microsoft Azure infrastructure







https://azure.microsoft.com/en-us/blog/processing-trillions-of-events-per-day-with-apache-kafka-on-azure/