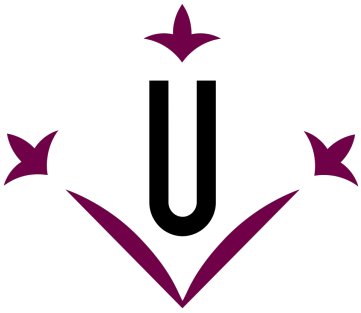
,,,,,,



Universitat de Lleida

Activity 2: Kafka
Components, benefits and applications.

Students
CASTRO SÁNCHEZ, DAVID
LLOBET ROQUÉ, PAU

Teachers
GABALDON PONSA, ELOI
GERVAS ARRUGA, JORGE
LERIDA MONSO, JOSEP LLUIS

Universitat de Lleida
Computer Science
Distributed Computing

22nd of October 2022

Índex

1 Kafka basics

In this section we will talk about the origins of kafka, what this technology is based on, what problem it solves and why is this so popular today.

1.1 Origins of Kafka

Kafka was developed around 2010 at LinkedIn by a team that included Jay Kreps, Jun Rao, and Neha Narkhede. The problem they originally set out to solve was **low-latency ingestion** of large amounts of event data from the LinkedIn website and infrastructure into a lambda architecture that harnessed Hadoop and real-time event processing systems.

The key was the "real-time"processing. At the time, there weren't any solutions for this type of ingress for real-time applications. Real-time systems such as the traditional messaging queues (ActiveMQ, RabbitMQ, etc.) have great **delivery guarantees** and support things such as **transactions**, and **message consumption tracking**, but they are overkill for the use case LinkedIn had in mind.

Everyone wants to build fancy machine-learning algorithms, but without the data, the algorithms are useless. Kafka was developed to be the **main data ingestion technology** for this type of use case. Recently, LinkedIn has reported ingestion rates of 1 trillion messages a day.

So, basically, kafka is the answer to the problems faced by the **distribution** and the **scaling** of messaging systems.

1.2 How it works?

The key to Kafka is the **log**. Developers often get confused when first hearing about this l·log,"because we're used to understanding l·logsïn terms of application logs. What we're talking about here, is the **log data structure**.

The log is simply a **time-ordered, append-only** sequence of data inserts where the data can be anything (in Kafka, it's just an array of bytes). If this sounds like the basic data structure upon which a database is built, it is.

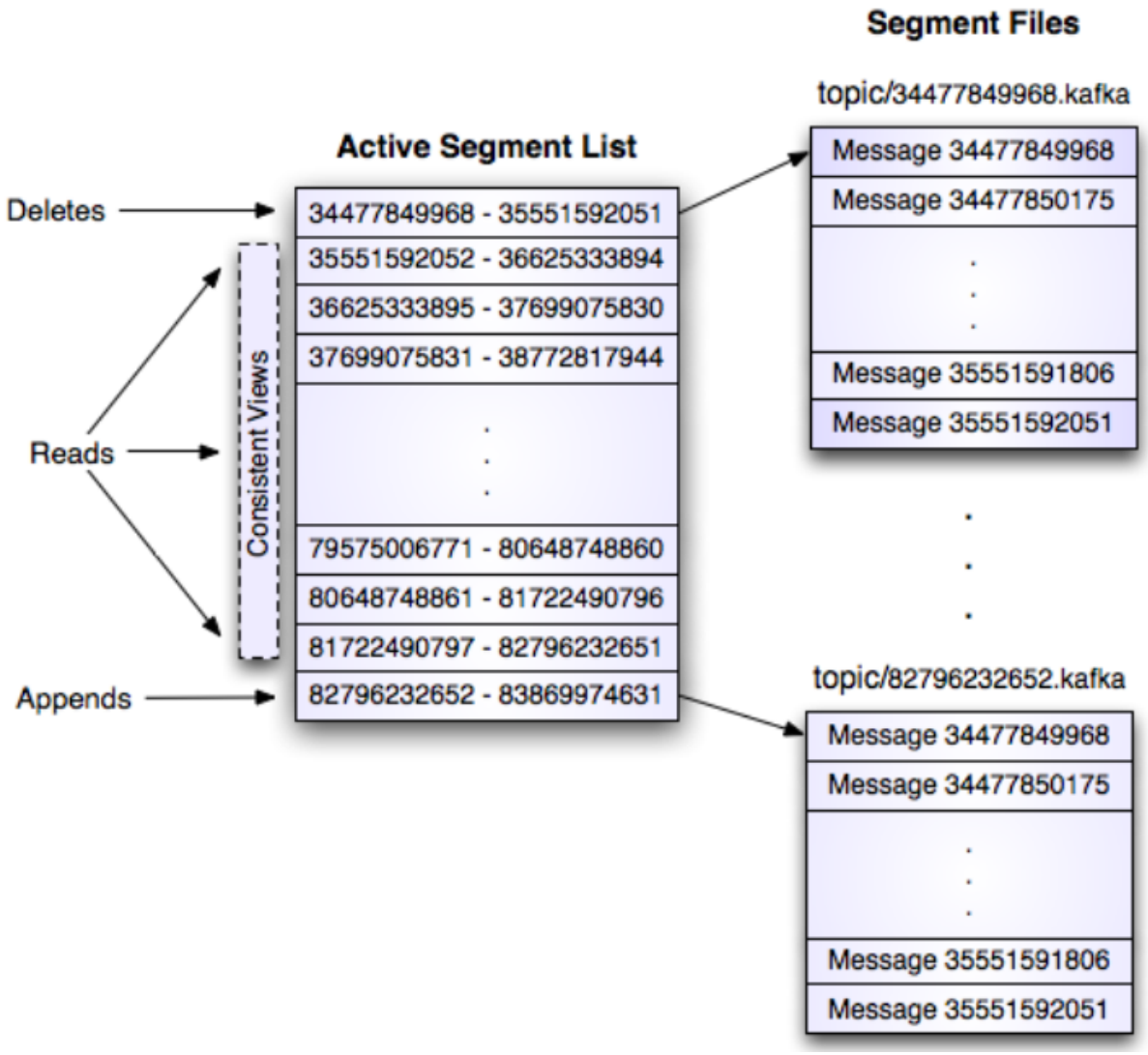


Figura 1: Kafka Log Implementation

Databases write change events to a log and derive the value of columns from that log. In Kafka, messages are written to a **topic**, which maintains this log from which subscribers can read and derive their own representations of the data.

Let's see a practical example. Imagine we want to design a system that listens to every basketball game updates from very sources such as game scoring, timing information, ...

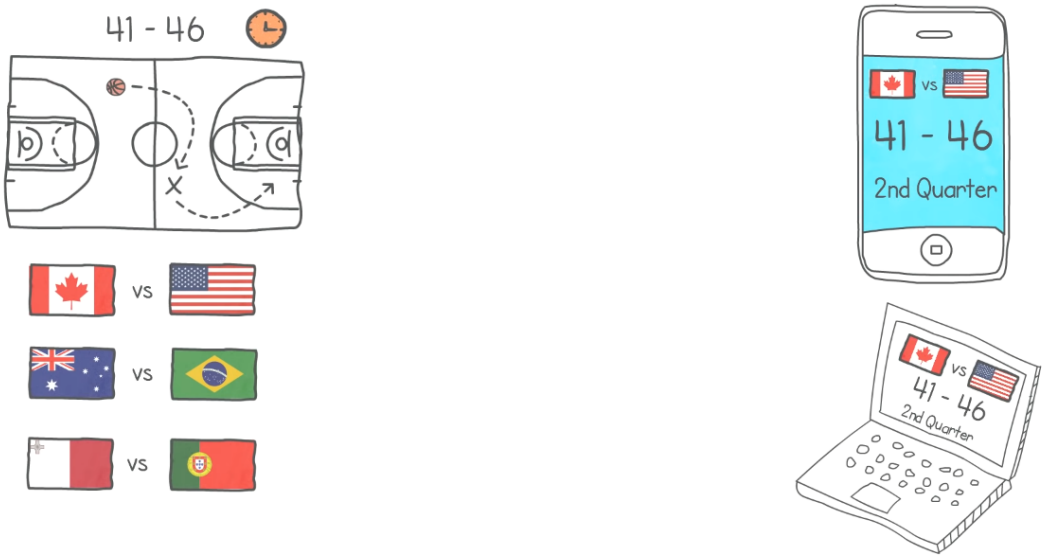


Figura 2: Visual representation of our system

Then, we want to display all of this information in different channels such as mobile devices and computers browsers. In our architecture, we have a process that reads this updates and writes them to a queue. We call this process a **Producer** since it's producing this updates into the queue.

At the head of this queue, a number of down scheme processes consume these updates to display them on the channels. We call this processes **Consumers**.

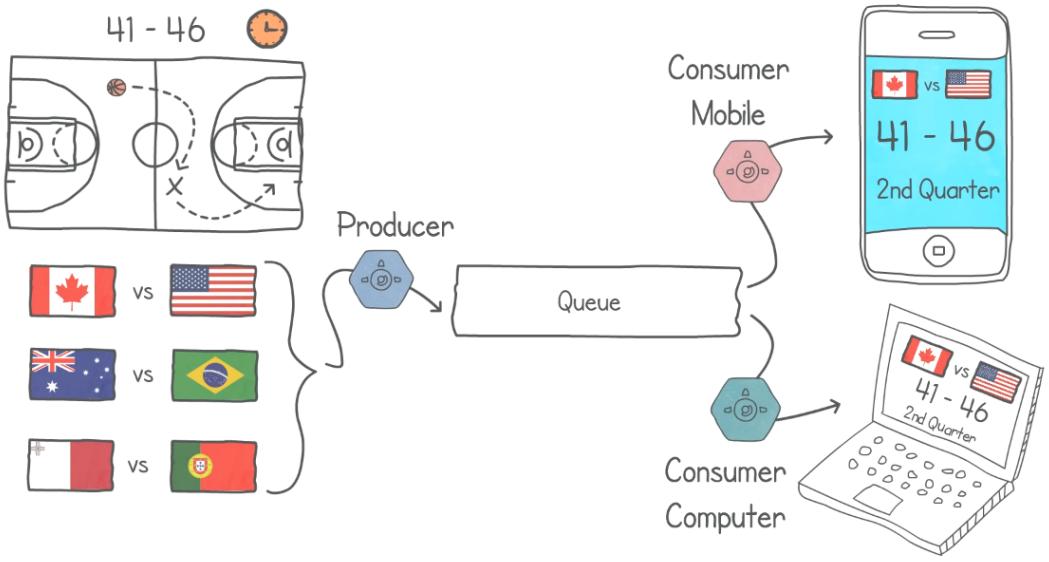


Figura 3: Producers and consumers representation

Imagine that we want to scale our system to track more and more games, the problem is that our servers are not able to process all this amount of data. The solution is to distribute the data processing in order to optimize resources.

Here is when kafka offers a solution. This solution is based on distributing the content in more than one queue, making partitions.

In this way, the information is processed in parallel in an optimal way. Kafka proposes an organization of information by attributes, in this way it ensures the correct distribution and avoids possible problems with erroneous messages.

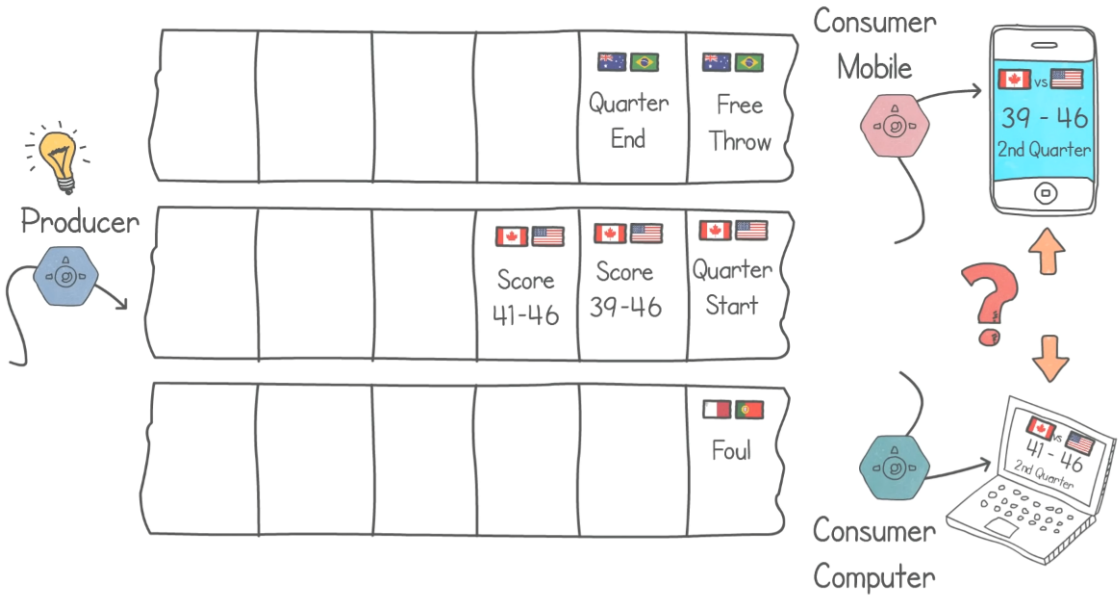


Figura 4: Kafka data structuring

2 Kafka benefits and drawbacks

Cosidering what we know of Kafka and it's purpose, in this section we will discuss it's benefits and drawbacks, just like any other system.

2.1 Main benefits

- **Scalability**: Kafka is a distributed system, which means it is able to be scaled quickly and easily without incurring any downtime. Apache Kafka is able to handle many terabytes of data without incurring much at all in the way of overhead.
- **Reliability/Availability**: Kafka replicates data and can support multiple subscribers. Additionally, it automatically balances consumers in the event of failure. That means that it is more reliable than similar messaging services available. This also ensures that kafka remains available to the consumers.
- **Latency**: Apache Kafka decouples the message which lets the consumer consume the message anytime. This leads to low latency value, up to 10 milliseconds.

- **Fault Tolerance**: Kafka has an essential feature to provide resistant to node/-machine failure within the cluster.
- **Bandwith/Performance**: Due to low latency, Kafka can handle a huge number of messages of high volume and high velocity. Delivers high throughput for both publishing and subscribing, utilizing disk structures that are offering constant levels of performance, even when dealing with many terabytes of stored messages.
- **Affordability**: Kafka is an open source software, so it's free to use by anyone.
- **Resource Sharing/Replication**: Kafka offers the replication feature, which makes data or messages to persist more on the cluster over a disk.
- **Consumer Friendly**: It is possible to integrate with the variety of consumers using Kafka. Kafka can behave or act differently according to the consumer that it integrates with, because each customer has a different ability to handle these messages coming out of Kafka. Moreover, Kafka can integrate well with a variety of consumers written in a variety of languages.

2.2 Main drawbacks

- **Complexity**:Kafka is an excellent platform for streamlining messages. However, in the case of migration projects which transforms data, Apache Kafka gets more complex. Hence, to interact both data producers and consumers you need to create data pipelines. That makes it significantly more complex.
- **Lack of complete set of monitoring tools**: Apache Kafka does not contain a complete set of monitoring as well as managing tools. Thus, new startups or enterprises fear to work with Kafka.
- **Reductions in Performance**: In general, there are no issues with the individual message size. However, the brokers and consumers start compressing these messages as the size increases. Due to this, when decompressed, the node memory gets slowly used. Also, compress happens when the data flow in the pipeline. It also affects throughput and performance.
- **Lack some message paradigms**: Certain message paradigms such as point-to-point queues, request/reply, etc. are missing in Kafka for some use cases.
- **Message Tweaking**: The broker uses certain system calls to deliver messages to the consumer. However, Kafka's performance reduces significantly if the message needs some tweaking. So, it can perform quite well if the message is unchanged because it uses the capabilities of the system.

2.3 Similar systems comparison

2.3.1 VS RabbitMQ

RabbitMQ is a general purpose message broker that supports protocols including MQTT, AMQP, and STOMP. It can deal with high-throughput use cases, such as online payment processing. It can handle background jobs or act as a message broker between microservices.

Some of the top differences are:

- **Performance**: RabbitMQ can process 4.000 to 10.000 messages per second. Whereas Kafka can process 1.000.000 messages per second which makes the performance of Kafka faster than RabbitMQ.
- **Topology**: RabbitMQ is Exchange queue topology based, in which your messages are sent to exchange after they are forwarded to different queues binding for the utilization of the consumers. Kafka is a publishing subscribe topology based, in which your messages are sent in the form of streams after they are forwarded to the users through various authorized groups.
- **Use Cases**: RabbitMQ is based on simple use cases, but on the other hand Kafka is based on massive data or high throughput cases.
- **Messaging Pattern**: RabbitMQ deletes the messages from queues when they are delivered and acknowledged. On the other hand, Kafka retains a record of the messages until its expiratory date and saves the messages in the queues.

2.3.2 VS ActiveMQ

ActiveMQ is a general-purpose message broker that supports several messaging protocols such as AMQP, STOMP, MQTT. It supports more complicated message routing patterns as well as the Enterprise Integration Patterns. In general, it is mainly used for integration between applications/services especially in a Service Oriented Architecture.

Some of the top differences are:

- **Performance**:With ActiveMQ performance slows down as the number of consumers starts increasing. Unlike with Kafka.
- **Push/Pull**: ActiveMQ is a push-type messaging platform where the providers push messages to the consumers. Kafka is a pull-type messaging platform where the consumers pull from the brokers.
- **Use Cases**: ActiveMQ is a traditional messaging system that deals with a small amount of data. And Kafka is a distributed system meant for processing a huge amount of data.

- **Scalability**: ActiveMQ has no chances of horizontal scalability and replication. While Kafka is scalable and available because of replication of partitions.

2.3.3 VS MQTT

MQTT is an open standard for a publish/subscribe messaging protocol. MQTT was built for IoT use cases, including constrained devices and unreliable networks. However, it was not built for data integration and data processing.

Although Kafka is also a messaging system based on publish/subscribe pattern, it is also called "distributed commit log" or "distributed streaming platform". Its main function is to achieve distributed persistent data preservation.

The only connection between the two is that they are both related to the publish/subscribe pattern. MQTT is a messaging protocol based on the publish/subscribe pattern, while the production and consumption processes of Apache Kafka are also part of the publish/subscribe pattern.

3 Practical uses of Kafka

Kafka can be extremely useful for some real cases, some examples are:

- **Activity tracking** : This was the original use case for Kafka. LinkedIn needed to rebuild its user activity tracking pipeline as a set of real-time publish-subscribe feeds.
- **Messaging** : Kafka works well as a replacement for traditional message brokers; Kafka has better throughput, built-in partitioning, replication, and fault-tolerance, as well as better scaling attributes.
- **Operational Metrics** : Kafka is often used for operational monitoring data. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.
- **Real-Time Data Processing** : Many systems require data to be processed as soon as it becomes available. Kafka transmits data from producers to consumers with very low latency (5 milliseconds, for instance). This is useful for: Financial organizations, Autonomous mobile devices, Logistical and supply chain businesses etc...

As an example, some services we use each day use Kafka, that is the case for:

- **Neftlix** : Uses Kafka to apply recommendations in real-time while you're watching TV shows
- **Uber** : Uses Kafka to gather user, taxi and trip data in real-time to compute and forecast demand, and compute surge pricing in real-time.
- **Linkedin** : Uses Kafka to prevent spam, collect user interactions to make better connection recommendations in real-time.

4 Conclusions

Analyzing the groups we previously saw in class, we believe that kafka belongs to the **Message-Oriented Middleware(MOM)**, since it follows the basic structure mentioned in class: a sender deposits a message in the message system, which forwards it to the message queue associated with each receiver.

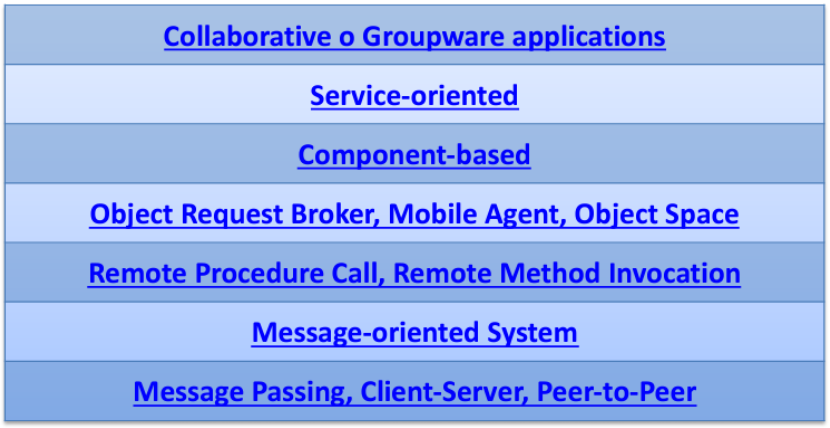


Figura 5: Different levels of abstraction

After the kafka analysis, we can understand why this technology has such a big impact today. Kafka provides a new form of messaging queues by **optimizing resources**, adapted to **scalability** and adding new functionalities such as **data tracking**.

And it does all this in an open way by providing **high affordability**, taking into account basic handicaps of distributed computing such as performance.Therefore, we can say that Kafaka is the best solution for a distributed computing system to **treat data as a continuous stream**.

5 Appendix

Bibliography consulted for the preparation of the report:

- Kafka explanation
- How kafka works (I)
- How kafka works (II)
- Kafka: principal concepts
- How kafka works
- Main functionalities
- Kafka highlights and drawbacks (I)
- Kafka highlights and drawbacks (II)
- Kafka highlights and drawbacks (III)
- Kafka highlights and drawbacks (IV)
- Class slides:
  - Chapter1: Cloud Computing
  - Chapter2: Programming Paradigms