Lab 4: JMS vs Kafka

1. Overview

Nowadays, message queues are ubiquitous in data-intensive applications. Message queues allow for reactive programming where a service can submit its message for transmission and still maintain reactivity. Two very popular message queueing options are JMS and Kafka.

The Java Message Service (JMS) API is a messaging standard that allows application components based on the Java Platform Enterprise Edition (Java EE) to create, send, receive, and read messages. It enables distributed communication that is loosely coupled, reliable, and asynchronous.

Apache Kafka is a distributed event store and stream-processing platform. It is an open-source system developed by the Apache Software Foundation, written in Java and Scala. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds.

To understand the different paradigms presented by these two tools, you are required to act as if you are taking the decision for your organization on which tool you should use. You should try these tools and test different aspects to make this decision.

2. Installation & Prerequisites

For this lab, we need to install Kafka. To set up this cluster, you only need to:

- Install Kafka

Since this time we are conducting experiments to assess the performance of two tools, we should avoid introducing too many moving pieces. Therefore, we should fix the following:

- Message size is limited to a reasonable 1KB message
- Programming API to Java

3. Requirements

You are required to test several aspects to make an educated decision. You should study the following elements:

- Performance aspects
    - Response time for both the produce and consume API calls
    - Maximum Throughput for both produce and consume API calls.
    - Median latency (time between production and consumption of messages)
- Usability
    - Number of steps needed to set up the tool
    - Degree of code cluttering to call produce and consume APIs
- Integrations
    - Number of programming languages supported
    - Number of out-of-the-box external data sources integrations

A) Performance Comparison

You are required to report the following metrics for both tools:

- Response time for both the produce and consume API calls
- Maximum Throughput for both produce and consume API calls.
- Median time between production and consumption of messages

Response Time

You should test and report the response time for both the produce and consume API calls. The following code snippet in JMS can easily achieve this.

```
long start = System.currentTimeInMillis();
Message msg = consumer.receive();
long responseTimeInMillis = System.currentTimeInMillis() - start;
```

This could be similarly achieved in Kafka.

Response times reported should be the median of at least 1000 runs. You will need to produce 1000 messages using the produce API (1K messages per call) and measure the produce response time median. Then, you should delete all messages in the queue except 1K messages, and then call the consume API 1000 times to consume the 1K messages and report the consume median response time.

Maximum Throughput

To get the maximum throughput possible in either the produce or consume API, you will have to submit a fixed number of requests in 1 second and check whether all messages were correctly delivered by consuming all messages from the message queue.

However, this is not as straightforward as it seems. For example, you can submit 1000 requests per second, but all of the 1000 requests are submitted by 1000 threads in the last millisecond of the second. This is okay as a skewed test, but not what we are testing here. A typical guideline is as follows:

1. Let the throughput you are testing be X
2. Divide 1 second by X to get the period time (T), the interval after which you submit a request.
3. Loop till requests are all sent and sleep after sending each request for T-0.2 * T.

This 0.2 T we subtract is to account for the thread switching overhead. This **may fail** to compensate for it, but better than nothing.

Then, you should test increasing throughputs as long as the current tested throughput is successful until you hit the maximum possible throughput and report it. The increase should be exponential to reach the maximum throughput quickly. A failed test is denoted by any non-zero number of failed requests.

Luckily, in Kafka's case, we have ready scripts to do that for us. In Kafka's bin directory, there is a utility for this. You will just have to start Kafka and Zookeeper first.

```
./kafka-[producer/consumer]-perf-test.sh \
    --topic test-topic \
    --num-records 1000000 \
    --record-size 1000 \
    --throughput 100000 \
    --producer-props bootstrap.servers=localhost:9092
```

A similar script is available for the consumer API.

Hint: Feel free to use JMeter to test JMS. But mention clearly with simple justifications the configurations used (ramp-up time, number of threads, etc.)

Median Latency

This is an important metric since it measures the delay the message queue introduces to the data stream pipeline you are operating.

In both Kafka and JMS, this could be achieved by the following:

1. Open a consumer that is actively consuming all messages available in the topic
2. Start producing messages to the message queue while attaching the timestamp of sending

3. Upon message consumption, subtract the current timestamp from the message timestamp.
4. Do this for 10K messages and get the median difference.

B) Usability

Report all usability issues you feel bothered you while using each tool:

- Tool set up overhead
    - This means how much time/number of steps you used to set up the tool before writing code to use it. This includes the following
        - Time for the setup of the tool itself,
        - Time to set up project dependencies and get a “Hello World” project working
- Degree of code cluttering to call produce and consume APIs. This includes the following
    - The number of lines used (end to end) to perform each operation (consume or produce)
    - The number of API calls used to perform each operation (configuration, connection initialization, actual operation call, etc.)

Feel free to list other attributes.

C) Integrations

Report all integrations you feel are helpful to Data Intensive Applications use cases. For example, mention whether it is easy to integrate with Hadoop Ecosystem data sources, columnar databases like Cassandra, etc..

This is considered a research task rather than a technical one. As in any research task, this should be done thoroughly and with proper references to make sure you have the correct and most recent information.

5. Deliverables

You are required to deliver the following:

- Report containing:
    - Sections for each requirement and sub-requirement
        - Performance Comparison
            - Sample code used in measuring the required metric in both Kafka and JMS produce and consume APIs
            - Median value for each metric in both Kafka and JMS produce and consume APIs
        - Usability

        - All metrics chosen
        - Table showing metric values for both JMS and Kafka
    - Integrations
        - A clear description of how you approached this research task
        - The output of the research methodology for both JMS and Kafka
- Summary section for each tool with subsections for Advantages and Disadvantages
- Conclusion section with your recommendation for which tool you should use.

6. Notes

- You should work in groups of **four**
- All team members should be ready to answer any questions during the discussion
- Feel free to use ChatGPT or similar Chatbots for help. **(You could lose grades if you fail to justify the answers of Chat GPT → do not use blindly without being convinced of its output)**
- Any cheating from other teams will be severely penalized.