

University of St.Gallen - Exercise Submission

Course Information

- **Course:** Event-driven and Process-oriented Architectures FS2024
- **Instructors:** B. Weber, R. Seiger, A. Abbad-Andalousi

Deadline

- **Submission Date:** 05.03.2024; 23:59

E01 - Outage of Zookeeper

E01-01: During Runtime

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Run the lab as provided and then shut down the Docker Container of Zookeeper.

Observations

- **Outcome:** Both Producer and Consumer continue running as before.

Conclusion

- Zookeeper is primarily responsible for setup. Therefore, shutting it down after the system is already running does not impact the currently running system.

E01-02: Before Runtime

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Shut down the Docker Container of Zookeeper, then run the lab as provided.

Observations

Producers

- creating topic: click-events
- `[kafka-admin-client-thread | adminclient-1] INFO org.apache.kafka.clients.NetworkClient - [AdminClient clientId=adminclient-1] Disconnecting from node 1001 due to request timeout.`
- `Cancelled createTopics request with correlation id 4 due to node 1001 being disconnected`
- Exited the program

- Similar for both producers


Consumer

- Loaded the last state if records were present in Kafka from previous runs.
- `[main] INFO org.apache.kafka.clients.consumer.KafkaConsumer - [Consumer clientId=consumer-grp1-1, groupId=grp1] Subscribed to topic(s): gaze-events, click-events`
 - Subscription to topics seems to be successful since Kafka Container is up and subscription is possible.
- `Id=grp1] Error while fetching metadata with correlation id 34524 : {gaze-events=UNKNOWN_TOPIC_OR_PARTITION, click-events=UNKNOWN_TOPIC_OR_PARTITION}`
 - Continuously fails within the While-loop to poll for records
- Program keeps running

Kafka

- Kafka Container Logs:
 - `2024-03-02 14:01:47 [2024-03-02 13:01:47,928] WARN [Controller id=1001, targetBrokerId=1001] Connection to node 1001 (localhost/127.0.0.1:9092) could not be established. Broker may not be available. (org.apache.kafka.clients.NetworkClient)`
 - `2024-03-02 14:01:47 java.io.IOException: Connection to localhost:9092 (id: 1001 rack: null) failed. 2024-03-02 14:01:47 at org.apache.kafka.clients.NetworkClientUtils.awaitReady(NetworkClientUtils.java:70) 2024-03-02 14:01:47 at kafka.controller.RequestSendThread.brokerReady(ControllerChannelManager.scala:298) 2024-03-02 14:01:47 at kafka.controller.RequestSendThread.doWork(ControllerChannelManager.scala:251) 2024-03-02 14:01:47 at org.apache.kafka.server.util.ShutdownableThread.run(ShutdownableThread.java:130) 2024-03-02 14:01:47 [2024-03-02 13:01:47,932] DEBUG [PartitionStateMachine controllerId=1001] Started partition state machine with initial state -> Map() (kafka.controller.ZkPartitionStateMachine)`

Conclusion

- The observations indicate that shutting down Zookeeper before running the system affects the connectivity and functionality of the producers and consumer, highlighting Zookeeper's critical role in initial system setup and maintenance. Zookeeper serves in managing the Kafka cluster, including broker health checks, leader election, and maintaining overall cluster stability.
- The Kafka Container Logs indicate that the system is unable to establish a connection with the broker, which is a direct result of Zookeeper being down.
-  Topics were created previously

E02 - Impact of Load and Batch Size on Processing Latency

E02-01: ...

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the batch size in the clickStream-producer configuration: batch.size
 - default is 16384 bytes (16KB)
 - set it to batch.size=65536 bytes (64KB)
- Measure the latency by capturing the time when the record is sent and when the acknowledgment is received. Adding below code snippets to the ClicksProducer and EyeTrackerProducer.

```

List<Long> latencyList = new ArrayList<>();
// ...
long sendTimeNano = System.nanoTime(); // Capture send time
// send the click event
Future<RecordMetadata> future = producer.send(new ProducerRecord<String,
Clicks>(
    topic, // topic
    clickEvent // value
));

RecordMetadata metadata = future.get(); // Blocks until the record is
acknowledged
long ackTimeNano = System.nanoTime(); // Capture ack time
// Calculate and print the latency in milliseconds
long latencyInMillis = (ackTimeNano - sendTimeNano) / 1_000_000;
System.out.println("Latency: " + latencyInMillis + " ms");
latencyList.add(latencyInMillis);
//...
if(counter == 100)
    break;
// ...
double averageLatency =
latencyList.stream().mapToLong(Long::longValue).average().orElse(0);
System.out.println("Average latency: " + averageLatency + " ms");

```

Observations

EyeTrackersProducer

Baseline with default batch size of 16KB

- ...
- gazeEvent sent: eventID: 4997, timestamp: 1350418648235600, xPosition: 892, yPosition: 811, pupilSize: 3, from deviceID: 0
- Latency: 3 ms

- gazeEvent sent: eventID: 4998, timestamp: 1350418658812100, xPosition: 748, yPosition: 884, pupilSize: 3, from deviceID: 1
- Latency: 2 ms
- gazeEvent sent: eventID: 4999, timestamp: 1350418670381200, xPosition: 1482, yPosition: 1038, pupilSize: 3, from deviceID: 0
- **Average latency: 2.3608 ms**

With batch size of 64KB

- **Average latency: 2.282 ms**
- **Average latency: 2.3622 ms**

With batch size of 1KB

- **Average latency: 2.6022 ms**
- **Average latency: 2.3792 ms**

ClickStreamProducer

Baseline with default batch size of 16KB

- ...
- clickEvent sent: eventID: 97, timestamp: 1350611618473000, xPosition: 418, yPosition: 121, clickedElement: EL1,
- Latency: 3 ms
- clickEvent sent: eventID: 98, timestamp: 1350614728184900, xPosition: 1172, yPosition: 24, clickedElement: EL8,
- Latency: 6 ms
- clickEvent sent: eventID: 99, timestamp: 1350616888767700, xPosition: 681, yPosition: 686, clickedElement: EL11,
- **Average latency: 5.49 ms**

With batch size of 64KB

- **Average latency: 5.47 ms**

With batch size of 1KB

- **Average latency: 6.32 ms**

Conclusion

It seems that with smaller batch sizes, the latency increases. This is expected as the producer has to send more requests to the broker, which increases the time it takes to send the data. The default batch size of 16KB seems to be a good balance between latency and throughput.

E03 - Setup of prometheus and grafana

Actions

Given the experiments require to keep track of metrics such as latency and throughput, we decided to investigate dedicated monitoring tools. Research on the topic led to Grafana and Prometheus as popular and for our purposes freely available tools. The setup of both tools was done using Docker containers:

```
prometheus:
  image: prom/prometheus
  volumes:
    - ./prometheus.yml:/etc/prometheus/prometheus.yml
    - prometheus_data:/prometheus # Persistent volume for Prometheus data
  ports:
    - "9090:9090"
  depends_on:
    - kafka

grafana:
  image: grafana/grafana
  volumes:
    - grafana_data:/var/lib/grafana # Persistent volume for Grafana data
  ports:
    - "3000:3000"
  depends_on:
    - prometheus

volumes:
  prometheus_data:
  grafana_data:
```

The Mbeans of Kafka and Zookeeper were then exposed to Prometheus by mounting the JMX exporter jar into the Kafka and Zookeeper containers. This required the following changes to the Kafka and Zookeeper containers:

```
KAFKA_OPTS: "-javaagent:/opt/bitnami/kafka/jmx_prometheus_javaagent-
0.20.0.jar=7203:/opt/bitnami/kafka/config.yml"
volumes:
  - ./jmx_prometheus_javaagent-
0.20.0.jar:/opt/bitnami/kafka/jmx_prometheus_javaagent-0.20.0.jar
  - ./config.yml:/opt/bitnami/kafka/config.yml
```

Observations

A Grafana dashboard can be created to visualize the metrics. This however requires quite some effort to set up. Pre-built dashboards are available but the metrics need to be mapped to the correct labels. Lastly, interpreting the metrics and understanding the labels can be challenging.

E04 The risk of data loss due to offset misconfigurations

E04-01: Read from 0

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer to read from offset 0 and run the lab as provided.

Observations

- The consumer reads all records from the beginning of the topic, including records that were sent before the consumer was started.

Conclusion

- The consumer can read records from the beginning of the topic, which can lead to duplicate processing of records if not handled properly.

E04-02: Read from 200

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer to read from offset 200 and run the lab as provided.

Observations

- The consumer reads all records from offset 200. As the offset starts from 0, the consumer reads the 201st record in the topic.

Conclusion

- The consumer can read records from the specified offset, which can lead to not processing records that were sent before the consumer was started.
- If the consumer is not aware of the offset, it can lead to data loss.

E04-03: Disable auto commit

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer properties to `enable.auto.commit=false` and keep `auto.offset.reset=earliest` and run the lab as provided.

Observations

- The consumer reads the records from the last committed offset and continues from there.

```
[main] INFO org.apache.kafka.clients.consumer.internals.AbstractFetch - [Consumer
clientId=consumer-grp1-1, groupId=grp1] Fetch position FetchPosition{offset=100,
offsetEpoch=Optional.empty, currentLeader=LeaderAndEpoch{leader=Optional[localhost:9092
(id: 1001 rack: null)], epoch=0}} is out of range for partition click-events-0,
resetting offset
```

```
[main] INFO org.apache.kafka.clients.consumer.internals.SubscriptionState - [Consumer
clientId=consumer-grp1-1, groupId=grp1] Resetting offset for partition click-events-0 to
position FetchPosition{offset=0, offsetEpoch=Optional.empty,
currentLeader=LeaderAndEpoch{leader=Optional[localhost:9092 (id: 1001 rack: null)],
epoch=0}}.
```

```
Received click-events - value: {eventID=0, timestamp=7530686035600, xPosition=970,
yPosition=513, clickedElement=EL17}
```

- stopped consumer at eventID: 20
- started consumer again -> Starts at eventID: 0

Conclusion

- Offset always starts from 0 so additional configuration like `enable.auto.commit=false` has no effect.

E04-04: Disable auto commit

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer properties to `enable.auto.commit=false` and **remove** `auto.offset.reset=earliest` and run the lab as provided.

Observations

- The Consumer started at the offset where at the point of running it the producer was approximately at with the last record sent. At around eventID=11
- Stopped the Consumer at eventID=27 (Producer)
- Restarted the Consumer at eventID=48 (Producer)
- Consumer started at eventID=48
- Let both the Consumer and the Producer run until the Producer reached eventID=100 and a few more.
- Stopped the Consumer at eventID=106 (Consumer)
- Restarted the Consumer at eventID=116 (Producer)
- Consumer started at eventID=100
- Stopped and Rerun the Consumer between eventID=258 and eventID=267 -> Consumer started at eventID=100

Conclusion

- It seems that if the Consumer is not committing the offset automatically at the last processed record, it will start at the last sent record (from the Producer) when restarted. No reprocessing of previous records is done.

- After 101 records were sent, the Consumer started at eventID=100 when restarted, even though the last sent record was higher than that.
- This indicates that after 101 records were sent, the offset was committed as eventID=100.

E04-05: Disable auto commit and auto offset store

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer properties to `enable.auto.commit=false` and remove `auto.offset.reset=earliest` and `enable.auto.offset.store=false` and run the lab as provided.

Observations

- Consumer starts at last committed record by the producer
- Consumer stopped at eventID=106 and Restarted at eventID=116 (Producer) -> Consumer started at eventID=100

Conclusion

- Seemingly the same behavior as in previous test.

E04-06: Disable auto commit and auto offset store and add auto commit interval

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Adjust the consumer properties to `enable.auto.commit=false` and remove `auto.offset.reset=earliest` and `enable.auto.offset.store=false` and `auto.commit.interval.ms=1000` and run the lab as provided.

Observations

- Consumer starts at last committed record by the producer
- Consumer stopped at eventID=6 and Restarted at eventID=13 (Producer) -> Consumer started at eventID=13

Conclusion

- Seemingly the same behavior as in previous test.
- The `auto.commit.interval.ms=1000` specifies the frequency in milliseconds at which the consumer attempts to commit offsets to Kafka when auto-commit is enabled. However, since `enable.auto.commit` is set to false, this setting does not have any effect.
- `enable.auto.commit` provides a way to automatically commit offsets at regular intervals without manual intervention. `enable.auto.offset.store` determines whether the consumer should automatically store the offset of the last message it has fetched in preparation for committing it (either automatically or manually).
- So `enable.auto.commit` is used to commit the offset at regular intervals (determined by `auto.commit.interval.ms`) to Kafka.

- And `enable.auto.offset.store` is used to store the offset of the last message fetched in preparation for committing it. This is an in-memory store.

E04-07: Enable auto commit and set auto commit interval

- stop at eventID=16 and restart at eventID=26 -> Consumer started at eventID=26

E05 - Broker and Replication

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Create a topic with a replication factor larger than the number of available brokers.

Observations

When creating a topic with a replication factor larger than the number of available brokers, the following error is thrown:

```
- I have no name!@71ae6ea639ba:/$ KAFKA_OPTS="" /opt/bitnami/kafka/bin/kafka-topics.sh --create --topic test-topic-AAA --bootstrap-server localhost:9092 --partitions 2 --replication-factor 3
- Error while executing topic command : Replication factor: 3 larger than available brokers: 1.
- [2024-03-04 21:46:20,108] ERROR
org.apache.kafka.common.errors.InvalidReplicationFactorException: Replication factor: 3 larger than available brokers: 1.
- (org.apache.kafka.tools.TopicCommand)
- I have no name!@71ae6ea639ba:/$ KAFKA_OPTS="" /opt/bitnami/kafka/bin/kafka-topics.sh --create --topic test-topic-AAA --bootstrap-server localhost:9092 --partitions 2 --replication-factor 1
- Created topic test-topic-AAA.
```

Conclusion

This is because we only have one broker available. The replication factor cannot be larger than the number of available brokers since the data cannot be replicated to a non-existent broker.

E05 - Consumer lag

Parameters and Process

- **Lab:** 02 Part2 Eye-tracking main.
- **Action:** Add `Thread.sleep(1000);` into the polling-loop of the consumer and run the lab as provided.

Observations

- Consumer only processed gaze-events from partition 0 at first
- Encountered following logs:

```
[kafka-coordinator-heartbeat-thread | grp1] INFO org.apache.kafka.clients.Metadata
- [Consumer clientId=consumer-grp1-1, groupId=grp1] Resetting the last seen epoch
of partition click-events-0 to 0 since the associated topicId changed from
XIAyepBTQiW120tzzrejVg to 7tTWV5PlSbi7Sp1-1wKuiw
consumer poll timeout has expired. This means the time between subsequent calls to
poll() was longer than the configured max.poll.interval.ms
Member consumer-grp1-1-8ad2817e-5950-4d40-80a4-e7b5cc3670a7 sending LeaveGroup
request to coordinator localhost:9092 (id: 2147482646 rack: null) due to consumer
poll timeout has expired.
Resetting generation and member id due to: consumer pro-actively leaving the group
Request joining group due to: consumer pro-actively leaving the group
```

- After all records of partition 0 were processed, the consumer started processing records from partition 1
- Encountered following logs:

```
Failing OffsetCommit request since the consumer is not part of an active group
Asynchronous auto-commit of offsets failed
Giving away all assigned partitions as lost since generation/memberID has been
reset
Lost previously assigned partitions click-events-0, gaze-events-0, gaze-events-1
Resetting offset for partition gaze-events-0 to position FetchPosition{offset=0,
offsetEpoch=Optional.empty,
Resetting offset for partition gaze-events-1 to position FetchPosition{offset=0,
offsetEpoch=Optional.empty,
Resetting offset for partition click-events-0 to position FetchPosition{offset=0,
offsetEpoch=Optional.empty
```

- Consumer starts processing records from partition 0 at offset 0 again
- Click-events haven't been consumed yet at all.
- The above situation occurred again and consumer started processing records from partition 0 at offset 0 again.

Conclusion

- The consumer lagged behind the producer significantly.
- Given the property of the consumer `auto.offset.reset= earliest` if a reset of the consumer group occurs, the consumer will start from the beginning of the topic.
- Processing one partition at a time points towards an attempted rebalance where ideally another consumer would take care of the other partition.