Kafka Test

Kafka 基本測驗

Kafka中的Topic和partition的關係為何？

一個 Topic 下會有 1..N 個 Partition，根據不同的 partition assignment strategies，producer 產生的消息會分送到不同 partition 下，在同個 partition 下的消息會有唯一的順序。

Kafka中，partition leader和partition follower是如何運作的呢？請把你了解的寫出來。

每個 partition 都會有唯一一個 Kafka Broker 擔任Partition leader，此 leader 是負責接收 producer 傳送到 Kafka cluster 的消息的， 在接收到消息後，他會把消息複製到 `min-in-sync-replicas` 數量的 follower 中，在 follower 回應複製成功後，leader 才會告訴 producer 「他已經成功發送消息了」。

請從 producer 參數中選出3個最能影響 producer 效能的參數，並且試著描述原因

1. `buffer.memory`: Producer 會把 client 想要傳送的消息存在 buffer, 再依據各種規則 and 參數 (e.g. `batch.size`, `linger.ms`, ... etc) 真正傳送消息出去， 如果 buffer 滿了，就會 block 住至多 `max.block.ms`，如果超過了就會拋出 exception。
2. `batch.size`: 每個 batch 的數量
3. `linger.ms`: producer 會把同個分區的多條消息匯集成 batch，並在 batch 滿後 (`batch.size`) or 每隔一段時間 (`linger.ms`) 就發送消息，所以如果

- `linger.ms` 設定得太大了，在 batch 沒有滿情況下，就有可能會有 consumer 延遲收到消息的情況發生。
- `batch.size` 設定得太小了，在 `linger.ms` 不變的情況下，會需要很多費時的 write system call，這樣就會影響效能。

NOTE: 只要 batch 滿了，Producer 會忽略 `linger.ms` 參數，並把這個 batch 送出去。 Ref:

- confluent doc: buffer.memory
- confluent doc: batch.size
- confluent doc: linger.ms
- kafka生产者Producer参数设置及调优

🚧 請試著介紹各個 partition assignment strategies 的優缺點和適用情境

Ref:

- confluent-partition.assignment.strategy
- From Eager to Smarter in Apache Kafka Consumer Rebalances

🚧 請說明KRaft 與 Zookeeper mode之間的差別。

Zookeeper mode 依賴 zookeeper 做 metadata 的管理，在 cluster 下現實

Ref: Apache Kafka Made Simple: A First Glimpse of a Kafka Without ZooKeeper

Kafka 實作

in-sync-replicas (ISR)

題目：

請試著run 3個broker, 設置min.insync.replicas=2，並且驗證in-sync broker 數量少於min.insync.replicas時會發生什麼事情？請以截圖搭配說明並作成 pdf 上傳 實驗：

ISR 的作用： ISR 指的是需要跟 partition leader 保持同步的 replica 數量，舉例來說，如果我們這裡有 3 個 broker, 那在 partition leader (kafka1) 發送消息，則 kafka1 需要把消息 replicate 到 kafka2, kafka3 才會回覆 client 發送成功。

準備環境：

docker-compose.yml: (設定KAFKA_MIN_INSYNC_REPLICAS: 2)

```
x-common-env: &common-env
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
      KAFKA_MIN_INSYNC_REPLICAS: 2

version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.2.1
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
  kafka1:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka1
    ports:
      - "8097:8097"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8097,INTERNAL://kafka1:9092
      <<: *common-env
  kafka2:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka2
    ports:
      - "8098:8098"
    depends_on:
      - zookeeper
```

```
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8098,INTERNAL://kafka2:9092
      <<: *common-env
  kafka3:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka3
    ports:
      - "8099:8099"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8099,INTERNAL://kafka3:9092
      <<: *common-env
```

創建 topic:  Screenshot 2024-01-05 at 9.21.20 AM

```
$ docker exec -it kafka1 kafka-topics --create --topic test-topic --
partitions 1 --replication-factor 2 --if-not-exists --bootstrap-server
kafka1:9092
```

在 `kafka1` 使用 `console-producer` 發送消息:  Screenshot 2024-01-05 at 9.22.00 AM

```
$ docker exec -it kafka1 kafka-console-producer --broker-list kafka1:9092
--topic test-topic
```

發現可以正常傳送。

在 `kafka3` 使用 `console-consumer` 接收消息，發現可以正常接收  image

在關掉 `kafka2` 後，使用 console-producer 發送消息 `try after isr`，會產生 `NOT_ENOUGH_REPLICAS` 錯誤  Screenshot 2024-01-05 at 9.16.59 AM

Ref: Kafka Topic Configuration: Minimum In-Sync Replicas

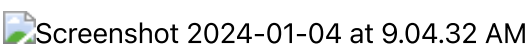
kafka-perf-test

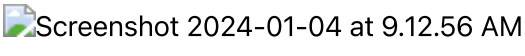
請試著佈署一個kafka cluster，並用kafka-producer-perf-test script送資料給一個topic，試著讓一個topic partition的資料夾內的log segment (xxx.log) 出現至少2個，並用kafka-dump-log script看最前面三筆資料。請以截圖搭配說明並作成 pdf 上傳

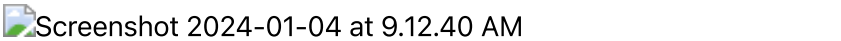
重現步驟：

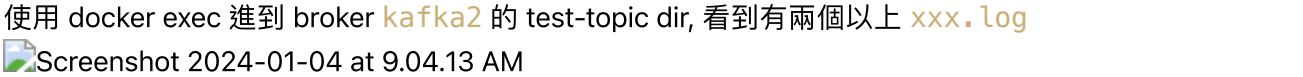
使用以下配置起一個 3 個 broker 的 zookeeper mode kafka cluster

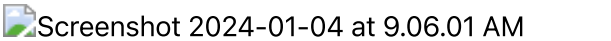
```
version: '3'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.2.1
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
  kafka1:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka1
    ports:
      - "8097:8097"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8097,INTERNAL://kafka1:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka2:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka2
    ports:
      - "8098:8098"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 2
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8098,INTERNAL://kafka2:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
  kafka3:
    image: confluentinc/cp-kafka:7.2.1
    container_name: kafka3
    ports:
      - "8099:8099"
    depends_on:
      - zookeeper
    environment:
      KAFKA_BROKER_ID: 3
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
EXTERNAL:PLAINTEXT,INTERNAL:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS:
EXTERNAL://localhost:8099,INTERNAL://kafka3:9092
      KAFKA_INTER_BROKER_LISTENER_NAME: INTERNAL
```

使用 `kafka-producer-perf-test` 製造 500000 個 record 

使用 kafka-topic 工具找到 partition leader 為 ID = 2 的 broker 

進到 zookeeper 使用 `zookeeper-shell` 工具 執行 `get /brokers/ids/2` 找到這個 ID=2 的 broker 所在的 ip `localhost:8098`, 為 `kafka2` 

使用 docker exec 進到 broker `kafka2` 的 test-topic dir, 看到有兩個以上 `xxx.log`


看最前面那個 log 的前三筆資料 (baseOffset: 0, 16, 32) 

Producer-Consumer-With-Order

請試著佈署一個kafka cluster，並完成下列步驟：

1. 建立一個具有 10 個 partitions 的 topic
2. 使用官方 Java Producer APIs 撰寫一隻程式傳遞 10 筆資料給上述 topic
3. 使用官方 Java Consumer APIs 撰寫一隻程式從上述 topic 收取該10筆資料
4. 請試著讓步驟2 和 步驟 3 的資料順序一致。例如 producer 傳送的資料順序為 a,b,c,d，consumer 收到的資料應該為 a,b,c,d

使用 Java API 建立一個擁有 10 個 partition 的 topic `order`， 然後建立 `ProducerExample` class，建立 `KeyPartitioner` 的 class, 他會實作 `Partitioner` 這個 interface，把 message 都透過 key 來做區分， 這樣在 10 筆資料都使用同個 key 的情況下，就能把他們分到同個 partition, 並維持 prodcer 傳送的資料順序。

`ProducerExample.java`:

```
package producer_consumer;

import java.nio.file.*;
import java.io.*;
import java.util.*;
import org.apache.kafka.clients.admin.*;
import org.apache.kafka.clients.producer.*;

public class ProducerExample {
    public static void main(String[] args) throws IOException {
        if (args.length != 1) {
            System.out.println("Please provide the configuration file path
as a command line argument");
            System.exit(1);
        }

        Properties props = loadConfig(args[0]);
        props.put("partitioner.class",
                "producer_consumer.KeyPartitioner");
        String topic = "order";
        int numPartitions = 10;
        createTopicIfNotExists(props, topic, numPartitions, 2);

        String[] orders = {
```

```
                "order0",
                "order1",
                "order2",
                "order3",
                "order4",
                "order5",
                "order6",
                "order7",
                "order8",
                "order9",
        };

        try (final Producer<String, String> producer = new KafkaProducer<>
(props)) {
            for (String order : orders) {
                String key = "order";
                String value = order + "value";
                producer.send(
                        // send these records to the same partition
                        new ProducerRecord<String, String>(topic, key,
order + "value"),
                        (event, ex) -> {
                            if (ex != null)
                                ex.printStackTrace();
                            else
                                System.out.printf("Produced event to topic
[%s], key=%s, value=%s, partition=%d\n",
                                        topic, key,
                                        value, event.partition());
                        });
                System.out.println(order);
            }
        }
    }

    private static void createTopicIfNotExists(Properties props, String
topic, int numPartitions, int repFactor) {
        try (AdminClient adminClient = AdminClient.create(props)) {
            // Check if the topic already exists
            ListTopicsResult topicsResult = adminClient.listTopics();
            Set<String> existingTopics = topicsResult.names().get();
            if (existingTopics.contains(topic)) {
                System.out.println("Topic already exists: " + topic);
            } else {
                // Create the topic with 10 partitions
                NewTopic newTopic = new NewTopic(topic, numPartitions,
(short) repFactor);
                CreateTopicsResult createTopicsResult =
adminClient.createTopics(Collections.singletonList(newTopic));
                createTopicsResult.all().get();
                System.out.println("topic has been created successfully");
            }
        } catch (Exception e) {
            e.printStackTrace();
```

```
        }
    }

    public static Properties loadConfig(final String configFile) throws
IOException {
        System.out.println("configFile: " + configFile);
        if (!Files.exists(Paths.get(configFile))) {
            throw new IOException(configFile + " not found.");
        }
        final Properties cfg = new Properties();
        try (InputStream inputStream = new FileInputStream(configFile)) {
            cfg.load(inputStream);
        }
        return cfg;
    }
}
```

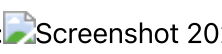
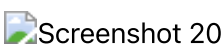
KeyPartitioner.java:

```
// https://redpanda.com/guides/kafka-tutorial/kafka-partition-strategy

package producer_consumer;

import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;
import org.apache.kafka.common.InvalidRecordException;
import java.util.Map;
import java.util.List;

public class KeyPartitioner implements Partitioner {
    public void configure(Map<String, ?> configs) {
    }

    public int partition(String topic, Object key, byte[] keyBytes,
            Object value, byte[] valueBytes, Cluster cluster) {
        List<PartitionInfo> partitions =
cluster.partitionsForTopic(topic);
        int numPartitions = partitions.size();
        if ((keyBytes == null) || (!(key instanceof String)))
            throw new InvalidRecordException("Record must have a valid
string key");
        return Math.abs(key.hashCode() % numPartitions);
    }

    public void close() {
    }

}
```

ConsumerExample.java:

```
package producer_consumer;
import java.time.Duration;
import java.util.Arrays;
import java.util.Properties;

import org.apache.kafka.clients.consumer.*;

public class ConsumerExample {

    public static void main(final String[] args) throws Exception {
        if (args.length != 1) {
            System.out.println("Please provide the configuration file path
as a command line argument");
            System.exit(1);
        }

        final String topic = "order";

        // Load consumer configuration settings from a local file
        // Reusing the loadConfig method from the ProducerExample class
        final Properties props = ProducerExample.loadConfig(args[0]);

        // Add additional properties.
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "order_example");
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        // StickyAssignor sticky = new StickyAssignor();
        // props.put(ConsumerConfig.PARTITION_ASSIGNMENT_STRATEGY_CONFIG,
        // sticky.name());
        try (final Consumer<String, String> consumer = new KafkaConsumer<>
(props)) {
            consumer.subscribe(Arrays.asList(topic));
            while (true) {
                ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));
                for (ConsumerRecord<String, String> record : records) {
                    String key = record.key();
                    String value = record.value();
                    System.out.println(
                            String.format("Consumed event from topic %s:
key = %-10s value = %s, partition: %d", topic,
                                    key, value,
                                    record.partition()));
                }
            }
        }
    }
}
```

Console output:

Producer:  Screenshot 2024-01-07 at 6.48.07 PM Consumer:  Screenshot 2024-01-07 at 6.48.22 PM

開源經驗

你是否有參與過任何開源專案？如果有的話請描述參與的經驗和心得

我寫過 `redis/rueidis` 這個專案， 我的貢獻是實作 `redis/go-redis` 到 `redis/rueidis` 的 adapter, 好讓使用者能無痛轉換。

在這過程中，我了解 API 的更動需要非常小心，盡量避免 breaking change.

並且我也學到了 Type State Pattern 這個技巧， 他可以把狀態綁在 type 上，避免額外的 validation.

舉例來說，以 `ZADD` 這個 command 為例，

```
ZADD key [NX | XX] [GT | LT] [CH] [INCR] score member [score member
  ...]
```

- XX: Only update elements that already exist. Don't add new elements.
- NX: Only add new elements. Don't update already existing elements. `NX` 與 `XX` 只能選一個，在 go-redis 是用這樣的 structure 來裝 argument, 但這就必須要額外的註解與檢查來告訴使用者 `NX`, `XX` 是互斥的。

```
type ZAddArgs struct {
    NX      bool
    XX      bool
    LT      bool
    GT      bool
    Ch      bool
    Members []Z
}
```

Ref: go-redis ZAddArgs

但在 `redis/rueidis` 裡面提供了 command builder, Type System 就會直接禁止你同時設定 `NX`, `XX`

```
client.B().Zadd().Key("1").Nx().Gt().Ch().Incr().ScoreMember().ScoreMember
(1, "1").ScoreMember(1, "1").Build()
```

Ref: rueidis.Builder.Zadd

🚧 你參與開源專案的目的為何？個人價值提升？訓練開發技巧？認識大神？為了身體健康？

🚧 你對於參與 kafka 開源貢獻有什麼想法和期待？