Performance optimization for Apache Kafka - Brokers

Written by Lovisa Johansson 2018-09-12

The Kafka Broker is the central part of Kafka. It receives and stores log messages until the given retention period has exceeded. By refining your broker setup, you can avoid common errors and ensure your configuration meets your expectations. This guide is divided into three parts, and this is part two. The previous and following blog post focuses on the Producers and Consumers in the Apache Kafka symbiosis.

Topics and Partitions

Kafka topics are divided into a number of partitions, which contains messages in an unchangeable sequence. Each message in a partition is assigned and identified by its unique offset. A topic can have multiple partition logs.

More Partitions Lead to Higher Throughput

The number of consumers is equal to the number of partitions. One partition will only be able to handle one consumer. Multiple partitions allow for multiple consumers to read from a topic in parallel, meaning that you will have a more scalable system. With more partitions, you can handle a larger throughput since all consumers can work in parallel.

Do not set up too many partitions

Partitions are the key to Kafka scalability, but that does not mean that you should have too many partitions. We have seen a few customers with way too many partitions which in turn consumes all resources from the server. Each partition in a topic uses a lot of RAM (file descriptors). The load on the CPU will also get higher with more partitions since Kafka needs to keep track of all of the partitions. More than 50 partitions for a topic are rarely recommended good practice.

Keep a good balance between cores and consumers

Each partition in Kafka is single threaded, too many partitions will not reach its full potential if you have a low number of cores on the server. Therefore you need to try to keep a good balance between cores and consumers. You don't want to have consumers idling, due to fewer cores than consumers.

Kafka Broker

A Kafka cluster consists of one or more servers (Kafka brokers), which are running Kafka.

How many brokers should we have?

In Kafka, replication is implemented at the partition level. Kafka automatically failover to these replicas when a server in the cluster fails so that messages remain available in the presence of failures. Each partition of a topic can be replicated on one or many nodes, depending on the number of nodes you have in your cluster. This redundant unit of a partition is called a replica. By having replicas, your data can be found on multiple places. The number of replicas you have for your topic is specified by you when you create the topic. The number of replicas can be changed in the future. The minimum number of in-sync replicas specify how many replicas that need to be available for the producer to successfully send messages to a partition.

A higher number of minimum number of in-sync replicas gives you higher persistency, but on the other hand, it might reduce availability, since the minimum number of replicas given must be available before a publish. If you have a 3 node cluster and minimum in-sync replicas is set to 3, and one node goes down, the other two are not able to receive any data.

You only care about the minimum number of in-sync replicas when it comes to the availability of your cluster and reliability guarantees. **The minimum number of in-sync replicas has nothing to do with the throughput. Setting the minimum number of in-sync replicas to larger than 1 may ensure less or no data loss, but throughput varies depending on the acks configuration.**

Partition load between brokers

The more brokers you have in your cluster, the higher performance you get since the load is spread between all of your nodes. A common error is that load is not distributed equally between brokers. **You should always keep an eye on partition distribution and do re-assignments to new brokers if needed, to ensure no broker is overloaded while another is idling.**

The CloudKarafka MGMT interface will show a warning if/when partition distribution is needed. The partition distribution is also simplified in the MGMT interface, you can simply press a button to distribute the data. The MGMT interface will check for existing partitions and spread the data between them. If you are not using the CloudKarafka MGMT, we recommend you to use the command line tool to spread your data.

Do not hardcode partitions

Keys are used to determine the partition within a log to which a message is appended to. A common error is that the same key is used when sending messages, making every message ending up on the same partition. **Make sure that you never hardcode the message key value.**

How many partitions should we have?

How many partitions you should have depends on your need. Most customers of CloudKarafka has 3 nodes in the setup, and the number of replicas set to 3. You can have as many replicas as you have nodes in your system.

A higher number of partitions is preferable for high throughput in Kafka, although a high number of partitions will put more load on the machines and might affect the latency of messages. Consider your desired result and don't exaggerate.

The configuration of the Apache Kafka Broker

One thing that we have changed a lot for all CloudKarafka instances is the number of file descriptors given to Apache Kafka. All CloudKarafka brokers have a very large number of file descriptors.

The topic is created by default

When sending a message to a non-existing topic, the topic is created by default since *auto.create.topics.enable* is set to *true* by default in Apache Kafka.

This config can be changed so that topics are not created if they do not exist. This configuration can be helpful in the matter of minimizing mistakes caused by misspelling or miscommunication between developers. Send us an email if you would like to change the default value of *auto.create.topics.enable* in your CloudKarafka cluster.

Change default Minimum In-sync Replicas

Default minimum In-sync Replicas is set to 1 by default in CloudKarafka, meaning that the minimum number of in-sync replicas that must be available for the producer to successfully send messages to a partition must be 1. This setting can be changed to a higher number if higher persistency is required. Send us an email if you would like to change the default minimum in-sync replicas in your cluster.

Default Retention Period

A message sent to a Kafka cluster is appended to the end of one of the logs. The message remains in the topic for a configurable period of time or until a configurable size is reached or until the specified retention for the topic exceeds. The message stays in the log, even if the message has been consumed. The default retention period can be changed in CloudKarafka MGMT interface.

Message Order in Kafka

One partition will guarantee an unchangeable sequence of your logstream. Two or more partitions will break the order since the order is not guaranteed between partitions.

Messages sent within Apache Kafka can be strictly ordered, even though your setup contains more than one partition. You will achieve a strict order of messages by setting up a consistent message key that sorts messages in the order specified, for example, user-ID. This will guarantee that all messages from a specific user always ends up in the same partition.

Please note that if the purpose of using Apache Kafka requires that all messages must be ordered within one topic, then you have to use only one partition.

More on this topic can be found in this blog post.

Number of Zookeepers

Zookeeper requires a majority of servers to be functioning. If you, for example, have 5 servers in your cluster, you would need 3 servers to be up and running for Zookeeper to be working.

I.e., you can afford to lose one Zookeeper in a 3 node cluster, and you can afford to lose 2 Zookeeper in a 5 node cluster.

What type of server do I need for Apache Kafka?

What you need when setting up a Kafka cluster is lots of memory. The data sent to the broker is always written to disk, but it also stays in the memory for as long as there is space to keep it in there. More memory will give you a higher throughput since Kafka Consumers, first of all, try to read memory data.

Kafka does not require high CPU, as long as you are not running too many partitions.

A larger plan in CloudKarafka gives you a larger disk. You will be able to have a longer retention period for your log messages since you will not run out of disk space.

Let's continue...

NEXT UP:

Part 3 - Performance optimization for Apache Kafka Consumers ⇢