Broker Configuration

JVM and Garbage Collection

Garbage collection has a huge impact on performance of JVM based applications. It is recommended to use the Garbage-First (G1) garbage collector for Kafka broker.

```
-server -XX:+UseG1GC -XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35 -XX:+DisableExplicitGC
-Djava.awt.headless=true -Djava.net.preferIPv4Stack=true
```

Set 4-8 GB of JVM heap size memory for the brokers depending on your use case. As Kafka's performance depends heavily on the operating systems page cache, it is not recommended to collocate with other memory-hungry applications.

- Large messages can cause longer garbage collection (GC) pauses as brokers allocate large chunks. Monitor the GC log and the server log. Add this to Broker Java Options:

```
-XX:+PrintGC -XX:+PrintGCDetails
-XX:+PrintGCTimeStamps
-Xloggc:</path/to/file.txt>
```

- If long GC pauses cause Kafka to abandon the ZooKeeper session, you may need to configure longer timeout values for ZooKeeper.

ZooKeeper Performance Tuning

Kafka uses Zookeeper to store metadata information about topics, partitions, brokers and system coordination (such as membership statuses). Unavailability or slowness of Zookeeper makes the Kafka cluster unstable, and Kafka brokers do not automatically recover from it. Use a 3-5 machines Zookeeper ensemble solely dedicated to Kafka (co-location of applications can cause unwanted service disturbances).

- `zookeeper.session.timeout.ms` is a setting for Kafka that specifies how long Zookeeper shall wait for heartbeat messages before it considers the client (the Kafka broker) unavailable. If this happens, metadata information about partition leadership owned by the broker will be reassigned. If this setting is too high, then it might take a long time for the system to detect a broker failure. On the other hand, if it is set to too small, it might result in frequent leadership reassignments.
- `jute.maxbuffer` is a crucial Java system property for both Kafka and Zookeeper. It controls the maximum size of the data a znode can contain. The default value, one megabyte, might be increased for certain production use cases.
- There are cases where Zookeeper can require more connections. In those cases, it is recommended to increase the `maxClientCnxns` parameter in Zookeeper.
- Note that old Kafka consumers store consumer offset commits in Zookeeper (deprecated). It is recommended to use new consumers that store offsets in internal Kafka topics (reduces load on

Zookeeper).

Network and I/O Threads

Kafka brokers use network threads to handle client requests. Incoming requests (such as produce and fetch requests) are placed into a requests queue from where I/O threads are taking them up and process them. After a request is processed, the response is placed into an internal response queue from where a network thread picks it up and sends response back to the client.

- `num.network.threads` is an important cluster-wide setting that determines the number of threads used for handling network requests (that is, receiving requests and sending responses). Set this value mainly based on number of producers, consumers and replica fetchers.
- `queued.max.requests` controls how many requests are allowed in the request queue before blocking network threads.
- `num.io.threads` specifies the number of threads that a broker uses for processing requests from the request queue (might include disk I/O).

ISR Management

An in-sync replica (ISR) set for a topic partition contains all follower replicas that are caught-up with the leader partition, and are situated on a broker that is alive.

- If a replica lags “too far” behind from the partition leader, it is removed from the ISR set. The definition of what is too far is controlled by the configuration setting `replica.lag.time.max.ms`. If a follower hasn't sent any fetch requests or hasn't consumed up to the leaders log end offset for at least this time, the leader removes the follower from the ISR set.

- `num.replica.fetchers` is a cluster-wide configuration setting that controls how many fetcher threads are in a broker. These threads are responsible for replicating messages from a source broker (that is, where partition leader resides). Increasing this value results in higher I/O parallelism and fetcher throughput but brokers use more CPU and network.

- `replica.fetch.min.bytes` controls the minimum number of bytes to fetch from a follower replica. If there is not enough bytes, wait up to `replica.fetch.wait.max.ms`.

- `replica.fetch.wait.max.ms` controls how long to sleep before checking for new messages from a fetcher replica. This value should be less than `replica.lag.time.max.ms`, otherwise the replica is kicked out of the ISR set.

- To check the ISR set for topic partitions, run the following command:

```
kafka-topics --zookeeper ${ZOOKEEPER_HOSTNAME}:2181/kafka --describe --
topic ${TOPIC}
```

- If a partition leader dies, a new leader is selected from the ISR set. There will be no data loss. If there is no ISR, unclean leader election can be used with the risk of data-loss.
- Unclean leader election occurs if `unclean.leader.election.enable` is set to true. By default, this is set to false.

Log Cleaner

The log cleaner implements log compaction. The following cluster-wide configuration settings can be used to fine tune log compaction:

- `log.cleaner.threads` controls how many background threads are responsible for log compaction. Increasing this value improves performance of log compaction at the cost of increased I/O activity.
- `log.cleaner.io.max.bytes.per.second` throttles log cleaner's I/O activity so that the sum of its read and write is less than this value on average.
- `log.cleaner.dedupe.buffer.size` specifies memory used for log compaction across all cleaner threads.
- `log.cleaner.io.buffer.size` controls total memory used for log cleaner I/O buffers across all cleaner threads.
- `log.cleaner.min.compaction.lag.ms` controls how long messages are left uncompacted.
- `log.cleaner.io.buffer.load.factor` controls log cleaner's load factor for the dedupe buffer. Increasing this value allows the system to clean more logs at once but increases hash collisions.
- `log.cleaner.backoff.ms` controls how long to wait until the next check if there is no log to compact.