Kafka Best Practices

18

☆

Short Description:

Best Practices in Deploying/Using Kafka in Produciton

Article

Kafka Broker:

Java Version

We recommend latest java 1.8 with G1 collector (which is default in new version). If you are using Java 1.7 and G1 collector make sure you are on u51 or higher.

A recommended setting for JVM looks like following

```
    -Xmx8g -Xms8g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -
    XX:G1HeapRegionSize=16M -XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80
```

OS Settings

Once the JVM size is determined leave rest of the RAM to OS for page caching. You need sufficient memory for page cache to buffer for the active writers and readers.

In general disk throughput is a performance bottleneck and more disks are better. Depending on how one configures the flush behavior , a faster disk will be beneficial if they log.flush.interval.messages set to flush for every 100k messages or so.

- File Descriptors limits: Kafka needs open file descriptors for files and network connections . We recommend at least 128000 allowed for file descriptors.
- Max socket buffer size , can be increased to enable high-performance data transfer. More details are here http://www.psc.edu/index.php/networking/641-tcp-tune (http://www.psc.edu/index.php/networking/641-tcp-tune)

Disks And File System

We recommend using multiple drives to get good throughput. Do not share the same drives with any other application or for kafka application logs.

Multiple drives can be configured using log.dirs in server.properties. Kafka assigns partitions in round-robin fashion to log.dirs directories.

Note: If the data is not well balanced among partitions this can lead to load imbalance among the disks. Also kafka currently doesn't good job of distributing data to less occupied disk in terms of space. So users can easily run out of disk space on 1 disk and other drives have free disk space and which itself can bring the Kafka down.

We highly recommend users to create alerts on disk usage for kafka drives to avoid any interruptions to running Kafka service.

RAID can potentially do better load balancing among the disks. But RAID can cause performance bottleneck due to slower writes and reduces available disk space. Although RAID can tolerate disk failures but rebuilding RAID array is I/O intensive that effectively disables the server. So RAID doesn't provide much real availability improvement.

Log Flush Management

Kafka always write data to files immediately and allows users to configure log.flush.interval.messages to enforce flush for every configure number of messages. One needs to set log.flush.scheduler.interval.ms (http://log.flush.scheduler.interval.ms) to a reasonable value for the above config to take into affect.

Also Kafka flushes the log file to disk whenever a log file reaches log.segment.bytes or log.roll.hours.

Note: durability in kafka does not require syncing data to disk, as a failed broker can recover the topic-partitions from its replicas. But pay attention to replica.lag.time.max.ms (http://replica.lag.time.max.ms) *, defaults to 10 secs If a follower didn't issue any fetch request or hasn't consumed from leaders log-end offset for at least this time , leader will remove the follower from ISR. Due to the nature of this there is slight chance of message loss if you do not explicitly set log.flush.interval.messages . If the leader broker goes down and if the follower didn't caught up to the leader it can still be under ISR for those 10 secs and the messages during this leader transition to follower can be lost.*

We recommend using the default flush settings which disables the explicit fsync entirely. This means relying on background flush done by OS and Kafka's own background flush. This provides great throughput and latency and full recovery guarantees provided by replication are stronger than sync to the local disk.

The drawback of enforcing the flushing is that its less efficient in its disk usage pattern and it can introduce latency as fsync in most Linux filesystems blocks writes to the file system compared to background flushing does much more granular page-level locking.

FileSystem Selection

Kafka uses regular files on disk, and such it has no hard dependency on a specific file system.

We recommend EXT4 or XFS. Recent improvements to the XFS file system have shown it to have the better performance characteristics for Kafka's workload without any compromise in stability.

Note: Do not use mounted shared drives and any network file systems. In our experience Kafka is known to have index failures on such file systems. Kafka uses MemoryMapped files to store the offset index which has known issues on a network file systems.

Zookeeper

- Do not co-locate zookeeper on the same boxes as Kafka
- We recommend zookeeper to isolate and only use for Kafka not any other systems should be depend on this zookeeper cluster
- Make sure you allocate sufficient JVM , good starting point is 4Gb
- Monitor: Use JMX metrics to monitor the zookeeper instance

Choosing Topic/Partitions

1. Topic/Partition is unit of parallelism in Kafka
2. Partitions in Kafka drives the parallelism of consumers
3. Higher the number of partitions more parallel consumers can be added , thus resulting in a higher throughput.
4. Based on throughput requirements one can pick a rough number of partitions.
    1. Lets call the throughput from producer to a single partition is P
    2. Throughput from a single partition to a consumer is C
    3. Target throughput is T
    4. Required partitions = Max (T/P, T/C)
5. More partitions can increase the latency
    1. The end-to-end latency in Kafka is defined by the time from when a message is published by the producer to when the message is read by the consumer.
    2. Kafka only exposes a message to a consumer after it has been committed, i.e., when the message is replicated to all the in-sync replicas.
    3. Replication 1000 partitions from one broker to another can take up 20ms. This can be too high for some real-time applications
    4. In new Kafka producer , messages will be accumulated on the producer side. It allows users to set upper bound on the amount of memory used for buffering incoming messages. Internally, producers buffers the message per partition. After enough data has been accumulated or enough time has passed, the accumulated messages will be removed and sent to the broker
    5. If we have more partitions , messages will be accumulated for more partitions on producer side.
    6. Similarly on the consumer side , it fetches batch of messages per partitions . The more partitions that consumer is subscribing to, the more memory it needs.

Factors Affecting Performance

- Main memory. More specifically File system buffer cache.
- Multiple dedicated disks.
- Partitions per topic. More partitions allows increased parallelism.
- Ethernet bandwidth.

Kafka Broker configs

1. Set kafka broker JVM by exporting KAFKA_HEAP_OPTS
2. Log.retention.hours . This setting controls when the old messages in a topic will be deleted. Take into consideration of your disk space and how long you would the messages to be available. An active consumer can read fast and deliver the message to destination.
3. Message.max.bytes . Maximum size of the message the server can receive. Make sure you set replica.fetch.max.bytes to be equal or greater than message.max.bytes
4. Delete.topic.enable - This will allow users to delete a topic from Kafka. This is set to false by default. Delete topic functionality will only work from Kafka 0.9 onwards.
5. unclean.leader.election - This config set to true by default. By turning this on User is making choice of availability over durability. If the a broker which leader for lets topic-part1 goes down and replica gets elected as leader when the original broker comes backup it becomes leader without it being a ISR. This means there is chance of data loss. If durability is more important, we recommend you to set this to false.

Kafka Producer:

org.apache.kafka.producer.KafkaProduer , Upgrade to the New producer .

Critical Configs

- Batch.size (size based batching)
- Linger.ms (http://Linger.ms) (time based batching)
- Compression.type
- Max.in.flight.requests.per.connection (affects ordering)
- Acks (affects durability)

Performance Notes

1. A producer thread going to the same partition is faster than a producer thread that sprays to multiple partitions.

2. The new Producer API provides a flush() call that client can optionally choose to invoke. If using it, the key number of bytes between two flush() calls is key factor for good performance. Microbenchmarking shows that around 4MB we get good perf (we used event of 1KB size).

Thumb rule to set batch size when using flush()

batch.size = total bytes between flush() / partition count.

3. If producer throughput maxes out and there is spare CPU and network capacity on box, add more producer processes.

4. Performance is sensitive to event size. In our measurements, 1KB events streamed faster than 100byte events. Larger events are likely to give better throughput.

5. No simple rule of thumb for linger.ms (http://linger.ms). Needs to be tried out on specific use cases. For small events (100 bytes or less), it did not not seem to have much impact in microbenchmarks.

Lifecycle of a request from Producer to Broker

- Polls batch from the batch queue , 1 batch per partition
- Groups batches based on the leader broker

- Sends the grouped batches to the brokers
- Pipelining if max.in.flight.requests.per.connection > 1

A batch is ready when one of the following is true:

- Batch.size is reached
- Linger.ms (http://Linger.ms) is reached
- Another batch to the same broker is ready
- flush() or close() is called

Big batching means

- Better compression ratio , higher throughput
- **Higher Latency**

Compression.type

- Compression is major part of producer's work
- Speed of different compression types differs a lot
- Compression is in user thread, so adding more threads helps with the throughput if compression is slow

ACKs

Defines durability level for producer.

Acks	Throughput	Latency	Durability
0	High	Low	No Gurantee
1	Medium	Medium	Leader
-1	Low	High	ISR

Max.in.flight.requests.per.connection

Max.in.flight.requests.per.connection > 1 means pipelining.

- Gives better throughput
- May cause out of order delivery when retry occurs
- Excessive pipelining , drops throughput

Kafka Consumer:

Performance Notes

On the consumer side it is generally easy to get good performance without need for tweaking.

Simple rule of thumb for good consumer performance is to keep

Number of consumer threads = Partition count

Microbenchmarking showed that Consumer performance was not as sensitive to event size or batch size as compared to Producer. Both 1kb and 100byte events showed similar throughput.

More details on Kafka microbenchmarking can be found here. https://drive.google.com/drive/u/1/folders/0ByKuMXNl6yEPfjVTRXIwaU45Qmh1Y3ktaExQa3YwZlR6SlZQTVVMckY2RGptb09QRS0zbVE (https://drive.google.com/drive/u/1/folders/0ByKuMXNl6yEPfjVTRXIwaU45Qmh1Y3ktaExQa3YwZlR6SlZQTVVMckY2RGptb09QRS0zbVE)