<http://www.quora.com/Whats-the-best-way-to-start-learning-Kafka>

<http://www.michael-noll.com/blog/2013/03/13/running-a-multi-broker-apache-kafka-cluster-on-a-single-node/>

1. Kafka is run as a cluster comprised of one or more servers each of which is called a broker.
2. Communication between the clients and the servers is done with a simple, high-performance, language agnostic [**TCP protocol**](https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol).
   1. Kafka Clients

<https://cwiki.apache.org/confluence/display/KAFKA/Clients>

* 1. Kafka protocol

<https://cwiki.apache.org/confluence/display/KAFKA/A+Guide+To+The+Kafka+Protocol>

1. Kafka's performance is effectively constant with respect to data size so retaining lots of data is not a problem.
2. The Kafka cluster retains all published messages—whether or not they have been consumed—for a configurable period of time
3. The offset is controlled by the consumer: normally a consumer will advance its offset linearly as it reads messages, but in fact the position is controlled by the consumer and it can consume messages in any order it likes. For example a consumer can reset to an older offset to reprocess

In my mind, a topic is a grouping of messages of a similar type that will be consumed by the same type of consumer so in the example above, I would just have a single topic and if you´ll decide to push some other kind of data through Kafka, you can add a new topic for that later.

Topics are registered in ZooKeeper which means that you might run into issues if trying to add too many of them, e.g. the case where you have a million users and have decided to create a topic per user.

Partitions on the other hand is a way to parallelize the consumption of the messages and the total number of partitions in a broker cluster need to be at least the same as the number of consumers in a consumer group to make sense of the partitioning feature. Consumers in a consumer group will split the burden of processing the topic between themselves according to the partitioning so that one consumer will only be concerned with messages in the partition itself is "assigned to".

Partitioning can either be explicitly set using a partition key on the producer side or if not provided, a random partition will be selected for every message.

HttpLayer: Network failure handling in kafka

Problem Statement

HttpLayer receives requests from Rover and pushes corresponding pixels to message queue (Kafka) to be consumed by RTS. In following network failure cases, this process slows down and impacts Rover negatively.

Kafka broker is down and producer is not able to connect to broker

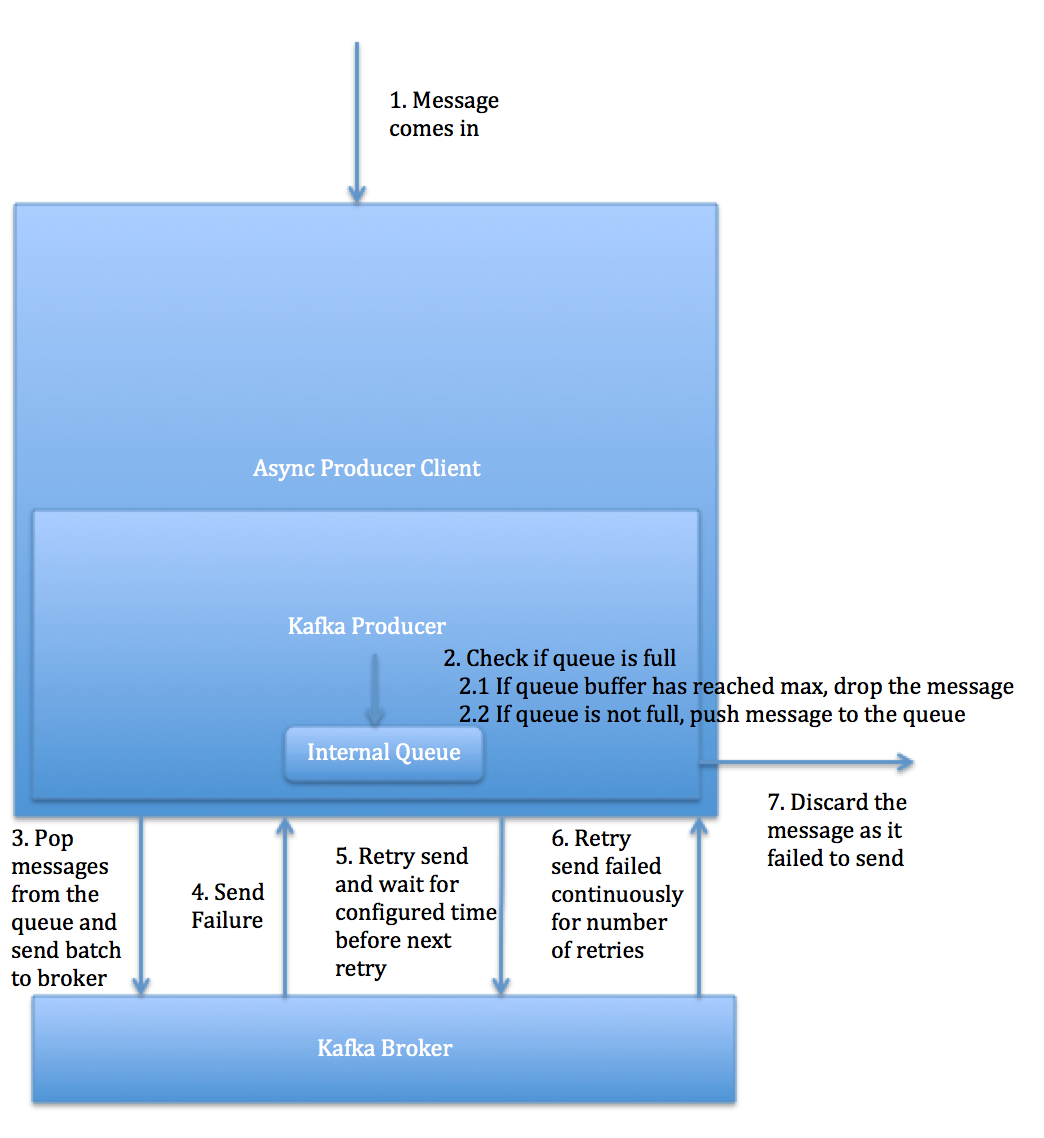
Network or kafka broker is slow, in turn slows down the publish rate

Our goal is to implement a solution to handle above network failure cases and minimize its impact on rover.

Solution

Tune Producer to drop messages

HttpLayer implements a solution to handle above cases by tuning kafka asynchronous producer to drop/discard messages. Following diagram describes the producer flow step by step.



Detailed description of producer config options used to tune async producer are given in the next section.

Async producer will buffer unsent messages until it reaches queue.buffering.max.messages. If queue.enqueue.timeout.ms is set to 0, producer immediately drops messages (will not block)

Once the message has been popped from the queue and sent to broker, it might result in send failure

In case of send failure, producer automatically retries send for given number of times - message.send.max.retries. Before each retry, producer waits for metadata refresh for the amount of time configured by retry.backoff.ms

When number of retries reaches message.send.max.retries with continuous failure, producer throws error and message is discarded.

Producer Configuration Parameters

Property Default Description

message.send.max.retries 3

This property will cause the producer to automatically retry a failed send request. This property specifies the number of retries when such failures occur. Note that setting a non-zero value here can lead to duplicates in the case of network errors that cause a message to be sent but the acknowledgement to be lost.

retry.backoff.ms 100

Before each retry, the producer refreshes the metadata of relevant topics to see if a new leader has been elected. Since leader election takes a bit of time, this property specifies the amount of time that the producer waits before refreshing the metadata.

topic.metadata.refresh.interval.ms 600 \* 1000

The producer generally refreshes the topic metadata from brokers when there is a failure (partition missing, leader not available...). It will also poll regularly (default: every 10min so 600000ms). If you set this to a negative value, metadata will only get refreshed on failure. If you set this to zero, the metadata will get refreshed after each message sent (not recommended). Important note: the refresh happen only AFTER the message is sent, so if the producer never sends a message the metadata is never refreshed

queue.buffering.max.ms 5000 Maximum time to buffer data when using async mode. For example a setting of 100 will try to batch together 100ms of messages to send at once. This will improve throughput but adds message delivery latency due to the buffering.

queue.buffering.max.messages 10000 The maximum number of unsent messages that can be queued up the producer when using async mode before either the producer must be blocked or data must be dropped.

queue.enqueue.timeout.ms -1

The amount of time to block before dropping messages when running in async mode and the buffer has reached queue.buffering.max.messages. If set to 0 events will be enqueued immediately or dropped if the queue is full (the producer send call will never block). If set to -1 the producer will block indefinitely and never willingly drop a send.

batch.num.messages 200 The number of messages to send in one batch when using async mode. The producer will wait until either this number of messages are ready to send or queue.buffer.max.ms is reached.

Monitoring

Kafka provides various producer stats (yammer metrics) that we can listen to and monitor. Following is the list of metrics we have chosen to add to HttpLayer metric registry for monitoring.

Metric

Description

Message Rate (One minute rate / Count) Number of incoming requests

Dropped Message Rate (One minute rate / Count) Number of messages that did not make to the queue

Failed Send Message Rate (One minute rate / Count) Number of messages discarded due to failure in send

Resend Message Rate (One minute rate / Count) Number of send attempts \* Number of messages discarded due to failure in send

Serialization Error Rate (One minute rate / Count) Number of messages discarded due to serialization error

Refer this section from SPO for more details on HttpLayer monitoring tool: HttpLayer SPO#Monitoring

Health-check

For kafka producer healthcheck, we check following meters against the max threshold value.

Dropped message per minute rate

Failed send per minute rate

Serialization error per minute rate

If these per minute rates are higher than max threshold value, it will be reported as unhealthy. These threshold values are configurable.

dropMinuteRateMax

failedSendMinuteRateMax

serializationErrorMinuteRateMax

Troubleshooting

Problem

Cause(s)

Resolution

**Process not running** Varies Find what is causing the issue: /var/log/kafka is a place to start.

**Out of disk space**

* Lost a node, so effectively lost replication capacity resulting in too much data in too few nodes [easy diagnosis, a host in cluster is DOWN or has lost disk(s)]
* Message production has suddenly spiked beyond normal bounds. Offset metrics can be viewed under any node in "EAP Kafka" in Ganglia.
* Message consumers may have issue consuming queue (this is the most unlikely, as a TTL has been enacted to automatically drop messages after a certain age)

Most of these causes are outside the scope of breakfix actions.

As such, the only immediate remediation is to run the following command to have kafka purge all the messages older than 16 hours. It will take about 10 minutes for kafka to purge the messages. There is no need to restart kafka process:

find /data1/kafka/\*queue\* -type f -cmin +960 | sudo xargs touch -c -t 1501010000

The other way is to manually delete files as follow. But please do not do this if you are not sure what you are doing.

Stop Kafka server and remove the oldest .log and .index files contained in the topic partition subfolders. (e.g. files inside "/data1/kafka/prod.rts.queue-0"). It is not recommended to blow away the entire parent topic folder.

Restart Kafka afterwards.

Run these commands in the exact order to manual cleaning up the space.

ps -ef | grep kafka. # make sure there is no kafka process running, very important !!

find /data1/kafka/\*queue\* -type f -cmin +960 | sudo xargs touch -c -t 1501010000

find /data1/kafka/\*queue\* -type f -mtime +30 | sudo xargs rm

df -h # make sure there are more than 10Gb of free space

sudo /etc/init.d/kafka start

ps -ef | grep kafka # make sure kafka stay up

If process comes up, but disk fills again, try deleting a batch of oldest logs according to disk space and folder size needs. Do not delete the latest log and index files.

If problem persists, there is a critical issue that likely requires engineering and pillar lead intervention.

**Process unable to start** Various failure conditions can result in Zookeeper and Kafka to be completely out of sync. Generally, this is after a catastrophic cluster failure, or a major disruptive maintenance.

Recovering from a failure such as this will require a complete purge and rebuild of Kafka and Zookeeper logs (these logs are NOT output error logs, but data and partition management "logs")

Stop Zookeeper, Kafka

Remove all partition data /data1/kafka/\*

Remove all zookeeper log data /var/lib/zookeeper/version-2/\* (do not delete "myid" file or zK will fail to start)

Restart Zookeeper, Kafka

Monitor logs for errors

Restart any producers/consumers that are stuck on stale sessions (See Inter-dependencies and find hosts matching IPs in logs)

Create any topics that are not created properly

Notify EAP pillar lead of status to validate broker and application health.

**Stale session errors in zookeeper log** Consumers can get stuck trying to establish connections using stale sessions after troubleshooting steps or after any situation requiring Kafka/Zookeeper restarts. Restart the offending dependent application. The requestor IP and Inter-dependencies section can be used to determine the source host. The application specific SPO can be then checked for the various stop|start|restart commands

The source code for Apache Kafka System Tools are located <https://github.com/apache/kafka/tree/0.8/core/src/main/scala/kafka/tools>

If you are looking for the replication related tools then please check out the wiki page on Replication tools

Consumer Offset Checker

Dump Log Segment

Export Zookeeper Offsets

Get Offset Shell

Import Zookeeper Offsets

JMX Tool

Kafka Migration Tool

Mirror Maker

Replay Log Producer

Simple Consumer Shell

State Change Log Merger

Update Offsets In Zookeeper

Verify Consumer Rebalance

System tools can be run from the command line using the run class script (i.e. bin/kafka-run-class.sh package.class --options)

Consumer Offset Checker

Displays the: Consumer Group, Topic, Partitions, Offset, logSize, Lag, Owner for the specified set of Topics and Consumer Group

bin/kafka-run-class.sh kafka.tools.ConsumerOffsetChecker

required argument: [group]

Option Description

------ -----------

--broker-info Print broker info

--group Consumer group.

--help Print this message.

--topic Comma-separated list of consumer

topics (all topics if absent).

--zkconnect ZooKeeper connect string. (default: localhost:2181)

Dump Log Segment

This can print the messages directly from the log files or just verify the indexes correct for the logs

bin/kafka-run-class.sh kafka.tools.DumpLogSegments

required argument "[files]"

Option Description

------ -----------

--deep-iteration if set, uses deep instead of shallow iteration

--files <file1, file2, ...> REQUIRED: The comma separated list of data and index log files to be dumped

--max-message-size <Integer: size> Size of largest message. (default: 5242880)

--print-data-log if set, printing the messages content when dumping data logs

--verify-index-only if set, just verify the index log without printing its content

Export Zookeeper Offsets

A utility that retrieves the offsets of broker partitions in ZK and prints to an output file in the following format:

/consumers/group1/offsets/topic1/1-0:286894308

/consumers/group1/offsets/topic1/2-0:284803985

bin/kafka-run-class.sh kafka.tools.ExportZkOffsets

required argument: [zkconnect]

Option Description

------ -----------

--group Consumer group.

--help Print this message.

--output-file Output file

--zkconnect ZooKeeper connect string. (default: localhost:2181)

Get Offset Shell

get offsets for a topic

bin/kafka-run-class.sh kafka.tools.GetOffsetShell

required argument [broker-list], [topic]

Option Description

------ -----------

--broker-list <hostname:port,..., REQUIRED: The list of hostname and hostname:port> port of the server to connect to.

--max-wait-ms <Integer: ms> The max amount of time each fetch request waits. (default: 1000)

--offsets <Integer: count> number of offsets returned (default: 1)

--partitions <partition ids> comma separated list of partition ids. If not specified, will find offsets for all partitions (default)

--time <Long: timestamp in milliseconds / -1(latest) / -2 (earliest) timestamp; offsets will come before this timestamp, as in getOffsetsBefore >

--topic <topic> REQUIRED: The topic to get offsets from.

Import Zookeeper Offsets

can import offsets for a topic partitions

file format is the same as for the export

/consumers/group1/offsets/topic1/1-0:286894308

/consumers/group1/offsets/topic1/2-0:284803985

bin/kafka-run-class.sh kafka.tools.ImportZkOffsets

required argument: [input-file]

Option Description

------ -----------

--help Print this message.

--input-file Input file

--zkconnect ZooKeeper connect string. (default: localhost:2181)

JMX Tool

prints metrics via JMX

bin/kafka-run-class.sh kafka.tools.JmxTool

Option Description

------ -----------

--attributes <name> The whitelist of attributes to query. This is a comma-separated list. If no attributes are specified all objects will be queried.

--date-format <format> The date format to use for formatting the time field. See java.text. SimpleDateFormat for options.

--help Print usage information.

--jmx-url <service-url> The url to connect to to poll JMX data. See Oracle javadoc for JMXServiceURL for details. (default: service:jmx:rmi:///jndi/rmi://: 9999/jmxrmi)

--object-name <name> A JMX object name to use as a query. This can contain wild cards, and this option can be given multiple times to specify more than one query. If no objects are specified all objects will be queried.

--reporting-interval <Integer: ms> Interval in MS with which to poll jmx stats. (default: 2000)

Kafka Migration Tool

Migrates a 0.7 broker to 0.8

bin/kafka-run-class.sh kafka.tools.KafkaMigrationTool

Missing required argument "[consumer.config]"

Option Description

------ -----------

--blacklist <Java regex (String)> Blacklist of topics to migrate from the 0.7 cluster

--consumer.config <config file> Kafka 0.7 consumer config to consume from the source 0.7 cluster. You man specify multiple of these.

--help Print this message.

--kafka.07.jar <kafka 0.7 jar> Kafka 0.7 jar file

--num.producers <Integer: Number of Number of producer instances (default: producers> 1)

--num.streams <Integer: Number of Number of consumer streams (default: 1)consumer threads>

--producer.config <config file> Producer config.

--queue.size <Integer: Queue size in Number of messages that are buffered terms of number of messages> between the 0.7 consumer and 0.8 producer (default: 10000)

--whitelist <Java regex (String)> Whitelist of topics to migrate from the 0.7 cluster

--zkclient.01.jar <zkClient 0.1 jar zkClient 0.1 jar file file required by Kafka 0.7>

Mirror Maker

Provides mirroring of one Kafka cluster to another, for more info check out the wiki page on Kafka mirroring (MirrorMaker)

bin/kafka-run-class.sh kafka.tools.MirrorMaker

required argument [consumer.config]

Option Description

------ -----------

--blacklist <Java regex (String)> Blacklist of topics to mirror.

--consumer.config <config file> Consumer config to consume from a source cluster. You may specify multiple of these.

--help Print this message.

--num.producers <Integer: Number of Number of producer instances (default: producers> 1)

--num.streams <Integer: Number of Number of consumption streams. threads> (default: 1)

--producer.config <config file> Embedded producer config.

--queue.size <Integer: Queue size in Number of messages that are buffered terms of number of messages> between the consumer and producer (default: 10000)

--whitelist <Java regex (String)> Whitelist of topics to mirror.

Replay Log Producer

Consume from one topic and replay those messages and produce to another topic

bin/kafka-run-class.sh kafka.tools.ReplayLogProducer

required argument [broker-list], [input-topic], [output-topic], [zookeeper]

Option Description

------ -----------

--async If set, messages are sent asynchronously.

--batch-size <Integer: batch size> Number of messages to send in a single batch. (default: 200)

--broker-list <hostname:port> REQUIRED: the broker list must be specified.

--compression-codec <Integer: If set, messages are sent compressed compression codec > (default: 0)

--delay-btw-batch-ms <Long: ms> Delay in ms between 2 batch sends. (default: 0)

--inputtopic <input-topic> REQUIRED: The topic to consume from.

--messages <Integer: count> The number of messages to send. (default: -1)

--outputtopic <output-topic> REQUIRED: The topic to produce to

--reporting-interval <Integer: size> Interval at which to print progress info. (default: 5000)

--threads <Integer: threads> Number of sending threads. (default: 1)

--zookeeper <zookeeper url> REQUIRED: The connection string for the zookeeper connection in the form host:port. Multiple URLS can be given to allow fail-over. (default: 127.0.0.1:2181)

Simple Consumer Shell

Dumps out consumed messages to the console using the Simple Consumer

bin/kafka-run-class.sh kafka.tools.SimpleConsumerShell

required argument [broker-list], [topic]

Option Description

------ -----------

--broker-list <hostname:port,..., REQUIRED: The list of hostname and hostname:port> port of the server to connect to.

--clientId <clientId> The ID of this client. (default: SimpleConsumerShell)

--fetchsize <Integer: fetchsize> The fetch size of each request. (default: 1048576)

--formatter <class> The name of a class to use for formatting kafka messages for display. (default: kafka.consumer. DefaultMessageFormatter)

--max-messages <Integer: max-messages> The number of messages to consume (default: 2147483647)

--max-wait-ms <Integer: ms> The max amount of time each fetch request waits. (default: 1000)

--no-wait-at-logend If set, when the simple consumer reaches the end of the Log, it will stop, not waiting for new produced messages

--offset <Long: consume offset> The offset id to consume from, default to -2 which means from beginning; while value -1 means from end (default: -2)

--partition <Integer: partition> The partition to consume from. (default: 0)

--print-offsets Print the offsets returned by the iterator

--property <prop>

--replica <Integer: replica id> The replica id to consume from, default -1 means leader broker. (default: -1)

--skip-message-on-error If there is an error when processing a message, skip it instead of halt.

--topic <topic> REQUIRED: The topic to consume from.

State Change Log Merger

A utility that merges the state change logs (possibly obtained from different brokers and over multiple days).

bin/kafka-run-class.sh kafka.tools.StateChangeLogMerger

Provide arguments to exactly one of the two options "[logs]" or "[logs-regex]"

Option Description

------ -----------

--end-time <end timestamp in the The latest timestamp of state change format java.text. log entries to be merged (default: SimpleDateFormat@f17a63e7> 9999-12-31 23:59:59,999)

--logs <file1,file2,...> Comma separated list of state change logs or a regex for the log file names

--logs-regex <for example: /tmp/state- Regex to match the state change log change.log\*> files to be merged

--partitions <0,1,2,...> Comma separated list of partition ids whose state change logs should be merged

--start-time <start timestamp in the The earliest timestamp of state change format java.text. log entries to be merged (default: SimpleDateFormat@f17a63e7> 0000-00-00 00:00:00,000)

--topic <topic> The topic whose state change logs should be merged

Update Offsets In Zookeeper

A utility that updates the offset of every broker partition to the offset of earliest or latest log segment file, in ZK.

bin/kafka-run-class.sh kafka.tools.UpdateOffsetsInZK

USAGE: kafka.tools.UpdateOffsetsInZK$ [earliest | latest] consumer.properties topic

Verify Consumer Rebalance

Make sure there is an owner for every partition. A successful rebalancing operation would select an owner for each available partition.

This means that for each partition registered under /brokers/topics/[topic]/[broker-id], an owner exists under /consumers/[consumer\_group]/owners/[topic]/[broker\_id-partition\_id]

bin/kafka-run-class.sh kafka.tools.VerifyConsumerRebalance

required argument: [group]

Option Description

------ -----------

--group Consumer group.

--help Print this message.

--zookeeper.connect ZooKeeper connect string. (default: localhost:2181)

Replication tools

Skip to end of metadata

Added by Sriram Subramanian, last edited by Neha Narkhede on Mar 14, 2014 (view change)

Go to start of metadata

1. Controlled Shutdown

What is controlled shutdown?

How to use the tool?

FAQ

How does the tool work when ack = 0?

What happens when there are no other replicas in the "in-sync" set for a partition?

2. Preferred Replica Leader Election Tool

What does the tool do?

How to use the tool?

FAQ

What happens if the preferred replica is not in the ISR?

How to find if all the partitions have been moved to the "preferred replica" after running the tool?

3. List Topic Tool

What does the tool do?

How to use the tool?

4. Create Topic Tool

What does the tool do?

How to use the tool?

5. Add Partition Tool

What does the tool do?

How to use the tool?

6. Reassign Partitions Tool

What does the tool do?

How to use the tool?

Cluster Expansion

Selectively moving some partitions to a broker

7. StateChangeLogMerger Tool

What does the tool do ?

How to use the tool ?

1. Controlled Shutdown

What is controlled shutdown?

In 0.8, each partition can have multiple replicas. These replicas are distributed across different brokers for better availability. At any given instance of time, only one of these replicas will serve reads and writes. In other words, one replica acts as the leader. When a broker needs to be shutdown (A broker can be shutdown for doing a new release, changing config etc), it would be useful to still serve the requests for the partitions on this broker using the other in-sync replicas. This is what the Controlled shutdown tool helps you to achieve. It transfers the leadership of the partitions from the broker (to be shutdown) to the other available replicas(in the in-sync set) on the remaining brokers.

Basically, it reduces the unavailable window. If we simply bring down a broker without controlled shutdown, partitions with leader on the broker are not available until the new leaders are elected. Since we currently elect leaders sequentially one partition at a time and each leader election involves reads/writes to ZK, it may take some time for all leaders to be elected, especially when there are many partitions. Doing the controlled shutdown allows us to move the leader proactively one at a time and thus reduce the unavailable window.

There are two ways to do controlled shutdown:

The first approach is to set "controlled.shutdown.enable" to true in the broker. Then, the broker will try to move all leaders on it to other brokers one at time before shutting itself down. One can tweak controlled.shutdown.max.retries and controlled.shutdown.retry.backoff.ms to control the max amount time the broker spends during controlled shutdown.

The second approach is to run a command line tool.

A summary of the steps that the tool does is shown below -

1. Finds the jmx port of the broker where the controller resides.

2. Issues a shutdown command to the controller using the port specifying the broker to be shutdown.

3. The controller finds the list of partitions the broker is currently leading.

4. For each of the partitions, the controller finds another broker that is in the "in-sync" replica set and makes it the leader for the partition.

5. For a given partition, if the controller cannot find any other replica that is in the 'in-sync" set, it fails the shutdown.

6. The shutdown tool waits for the response of the controller. If it finds that the shutdown was not successful, it retries (this is configurable) and eventually succeeds or fails based on the controller response.

Note that the shutdown tool only moves the leadership of the partitions to the other available brokers. It does not terminate the broker. This needs to be done manually after running the tool.

How to use the tool?

bin/kafka-run-class.sh kafka.admin.ShutdownBroker --zookeeper localhost:12913/kafka --broker #brokerId# --num.retries 3 --retry.interval.ms 60

The tool takes in a list of zookeeper hosts (including the namespace if any) and the broker id that needs to be shutdown. Both these arguments are mandatory. The tool also allows to specify an optional number of retries and the interval between those retries. If the shutdown attempt by the controller fails, the tool retries after the specified interval.

FAQ

How does the tool work when ack = 0?

When the leadership for a partition is changed, the clients (producer and consumer) gets an error when they try to produce or consume from the old leader when they wait for a response. The client then refreshes the partition metadata from zookeeper and gets the new leader for the partition and retries. This does not work for the producer client when ack = 0. This is because the producer does not wait for a response and hence does not know about the leadership change. The client would end up loosing messages till the shutdown broker is brought back up.

What happens when there are no other replicas in the "in-sync" set for a partition?

When a partition does not have any other replicas that are in the "in-sync" set, the tool fails to shutdown the broker. This is to ensure that we do not have an unclean leader election (choosing a replica that is outside the "in-sync" set) and thereby causing data loss.

2. Preferred Replica Leader Election Tool

What does the tool do?

With replication, each partition can have multiple replicas. The list of replicas for a partition is called the "assigned replicas". The first replica in this list is the "preferred replica". When topic/partitions are created, Kafka ensures that the "preferred replica" for the partitions across topics are equally distributed amongst the brokers in a cluster. In an ideal scenario, the leader for a given partition should be the "preferred replica". This guarantees that the leadership load across the brokers in a cluster are evenly balanced. However, over time the leadership load could get imbalanced due to broker shutdowns (caused by controlled shutdown, crashes, machine failures etc). This tool helps to restore the leadership balance between the brokers in the cluster. A summary of the steps that the tool does is shown below -

1. The tool updates the zookeeper path "/admin/preferred\_replica\_election" with the list of topic partitions whose leader needs to be moved to the preferred replica.

2. The controller listens to the path above. When a data change update is triggered, the controller reads the list of topic partitions from zookeeper.

3. For each topic partition, the controller gets the preferred replica (the first replica in the assigned replicas list). If the preferred replica is not already the leader and it is present in the isr, the controller issues a request to the broker that owns the preferred replica to become the leader for the partition.

Note that the tool only updates the zookeeper path and exits. The controller moves the leader for a partition to the preferred replica asynchronously.

How to use the tool?

bin/kafka-preferred-replica-election.sh --zookeeper localhost:12913/kafka --path-to-json-file topicPartitionList.json

The tool takes a mandatory list of zookeeper hosts and an optional list of topic partitions provided as a json file. If the list is not provided, the tool queries zookeeper and gets all the topic partitions for the cluster. The tool exits after updating the zookeeper path "/admin/preferred\_replica\_election" with the topic partition list.

Example json file (This is optional. This can be specified to move the leader to the preferred replica for specific topic partitions)

{

"partitions":

[

{"topic": "topic1", "partition": "0"},

{"topic": "topic1", "partition": "1"},

{"topic": "topic1", "partition": "2"},

{"topic": "topic2", "partition": "0"},

{"topic": "topic2", "partition": "1"},

]

}

FAQ

What happens if the preferred replica is not in the ISR?

The controller will fail to move the leadership to the preferred replica if it is not in the ISR. This is to ensure that there is no dataloss. When the replica becomes "in-sync" with the leader, the tool can be run again to move the leader.

How to find if all the partitions have been moved to the "preferred replica" after running the tool?

ListTopicCommand is an excellent tool that provides an overview of all the topic partitions in the cluster. For each topic partition, it displays the leader, assigned replicas and current "in-sync" replica set. If the leader and the first replica in the assigned replica set are the same then the Preferred replica leader election" tool succeeded. If not, the tool failed and may have to be run again.

3. List Topic Tool

What does the tool do?

This tool lists the information for a given list of topics. If no topics are provided in the command line, the tool queries zookeeper to get all the topics and lists the information for them. The fields that the tool displays are - topic name, partition, leader, replicas, isr. Two optional arguments can be provided to the tool. If "under-replicated-partitions" is specified, the tool only provides information for those topic / partitions which have replicas that are under replicated. If "unavailable-partitions" is specified, the tool only provides information for those topic/partitions whose leader is not available.

How to use the tool?

List info for topic1

bin/kafka-list-topic.sh --zookeeper localhost:2121 --topic topic1

List info for all topics

bin/kafka-list-topic.sh --zookeeper localhost:2121

List info for topics which have under replicated count

bin/kafka-list-topic.sh --zookeeper localhost:2121 --under-replicated-partitions

List info for topics whose leader for a partition is not available

bin/kafka-list-topic.sh --zookeeper localhost:2121 --unavailable-partitions

4. Create Topic Tool

What does the tool do?

By default, Kafka auto creates topic if "auto.create.topics.enable" is set to true on the server. This creates a topic with a default number of partitions, replication factor and uses Kafka's default scheme to do replica assignment. Sometimes, it may be required that we would like to customize a topic while creating it. This tool helps to create a topic and also specify the number of partitions, replication factor and replica assignment list for the topic.

How to use the tool?

bin/kafka-create-topic.sh

Option Description

------ -----------

--partition <Integer: # of partitions> number of partitions in the topic

(default: 1)

--replica <Integer: replication factor> replication factor for each partitions

in the topic (default: 1)

--replica-assignment-list for manually assigning replicas to brokers

(default: )

<broker\_id\_for\_part1\_replica1 :

broker\_id\_for\_part1\_replica2,

broker\_id\_for\_part2\_replica1 :

broker\_id\_for\_part2\_replica2, ...>

--topic <topic> REQUIRED: The topic to be created.

--zookeeper <urls> REQUIRED: The connection string for

the zookeeper connection in the form

host:port. Multiple URLS can be

given to allow fail-over.

5. Add Partition Tool

What does the tool do?

In Kafka partitions act as the unit of parallelism: messages of a single topic are distributed to multiple partitions that can be stored and served on different servers. Upon creation of a topic, the number of partitions for this topic has to be specified. Later on more partitions may be needed for this topic when the volume of this topic increases. This tool helps to add more partitions for a specific topic and also allow manual replica assignment of the added partitions. This tool is only available in the 0.8 branch for now.

How to use the tool?

bin/kafka-add-partitions.sh

Option Description

------ -----------

--partition <Integer: # of partitions> REQUIRED: Number of partitions to add

to the topic

--replica-assignment-list For manually assigning replicas to

<broker\_id\_for\_part1\_replica1 : brokers for the new partitions

broker\_id\_for\_part1\_replica2, (default: )

broker\_id\_for\_part2\_replica1 :

broker\_id\_for\_part2\_replica2, ...>

--topic <topic> REQUIRED: The topic for which

partitions need to be added.

--zookeeper <urls> REQUIRED: The connection string for

the zookeeper connection in the form

host:port. Multiple URLS can be

given to allow fail-over.

6. Reassign Partitions Tool

What does the tool do?

The goal of this tool is similar to the Referred Replica Leader Election Tool as to achieve load balance across brokers. But instead of only electing a new leader from the assigned replicas of a partition, this tool allows to change the assigned replicas of partitions – remember that followers also need to fetch from leaders in order to keep in sync, hence sometime only balance the leadership load is not enough.

A summary of the steps that the tool does is shown below -

1. The tool updates the zookeeper path "/admin/reassign\_partitions" with the list of topic partitions and (if specified in the Json file) the list of their new assigned replicas.

2. The controller listens to the path above. When a data change update is triggered, the controller reads the list of topic partitions and their assigned replicas from zookeeper.

3. For each topic partition, the controller does the following:

3.1. Start new replicas in RAR - AR (RAR = Reassigned Replicas, AR = original list of Assigned Replicas)

3.2. Wait until new replicas are in sync with the leader

3.3. If the leader is not in RAR, elect a new leader from RAR

3.4 4. Stop old replicas AR - RAR

3.5. Write new AR

3.6. Remove partition from the /admin/reassign\_partitions path

Note that the tool only updates the zookeeper path and exits. The controller reassign the replicas for the partitions asynchronously.

This tool is only available in the 0.8 branch for now.

How to use the tool?

WARNING: The tool was released in beta in 0.8 and has some bugs that can render the topic unusable. The tool is known to be stable in 0.8.1.

bin/kafka-reassign-partitions.sh

Option Description

------ -----------

--broker-list <brokerlist> The list of brokers to which the

partitions need to be reassigned in

the form "0,1,2". This is required

for automatic topic reassignment.

--execute [execute] This option does the actual

reassignment. By default, the tool

does a dry run

--manual-assignment-json-file <manual The JSON file with the list of manual

assignment json file path> reassignmentsThis option or topics-

to-move-json-file needs to be

specified. The format to use is -

{"partitions":

[{"topic": "foo",

"partition": 1,

"replicas": [1,2,3] }],

"version":1

}

--topics-to-move-json-file <topics to The JSON file with the list of topics

reassign json file path> to reassign.This option or manual-

assignment-json-file needs to be

specified. The format to use is -

{"topics":

[{"topic": "foo"},{"topic": "foo1"}],

"version":1

}

--zookeeper <urls> REQUIRED: The connection string for

the zookeeper connection in the form

host:port. Multiple URLS can be

given to allow fail-over.

Please note that by default the script runs in a dry-run mode and does not initiate the partition movement. Only when the --execute option is specified, the tool proceeds to start the partition movement

Cluster Expansion

The partition reassignment tool can be used to expand an existing 0.8 cluster. Cluster expansion involves including brokers with new broker ids in a Kafka 08 cluster. Typically, when you add new brokers to a cluster, they will not receive any data from existing topics until this tool is run to assign existing topics/partitions to the new brokers. The tool allows 2 options to make it easier to move some topics in bulk to the new brokers. These 2 options are a) topics to move b) list of newly added brokers. Using these 2 options, the tool automatically figures out the placements of partitions for the topics on the new brokers.

The following example moves 2 topics (foo1, foo2) to newly added brokers in a cluster (5,6,7)

nnarkhed$ ./bin/kafka-reassign-partitions.sh --topics-to-move-json-file topics-to-move.json --broker-list "5,6,7" --execute

nnarkhed$ cat topics-to-move.json

{"topics":

[{"topic": "foo1"},{"topic": "foo2"}],

"version":1

}

Selectively moving some partitions to a broker

The partition movement tool can also be moved to selectively move some replicas for certain partitions over to a particular broker. Typically, if you end up with an unbalanced cluster, you can use the tool in this mode to selectively move partitions around. In this mode, the tool takes a single file which has a list of partitions to move and the replicas that each of those partitions should be assigned to.

The following example moves 1 partition (foo-1) from replicas 1,2,3 to 1,2,4

nnarkhed$ ./bin/kafka-reassign-partitions.sh --manual-assignment-json-file partitions-to-move.json --execute

nnarkhed$ cat partitions-to-move.json

{"partitions":

[{"topic": "foo",

"partition": 1,

"replicas": [1,2,4] }],

}],

"version":1

}

7. StateChangeLogMerger Tool

What does the tool do ?

The goal of this tool is to collect data from the brokers in a cluster and format it in a central log to help troubleshoot issues with state changes. Every broker in a Kafka cluster emits a state-change.log that logs the lifecycle of every state change received by the broker. Often times, there is some problem with leader election for a subset of topics/partitions and the question is what caused the problem. In order to answer this question, we need a global view of state changes in the kafka cluster, possibly filtered on a time range and/or specific topics/partitions. This is exactly what the StateChangeLogMerger tool does. It takes in a list of state-change.log files, merges them in time order, filters on a certain time range if specified by the user, filters on topics/partitions if specified by the user, and outputs a merged and formatted state-change.log that is easy to query and understand the root cause.

How to use the tool ?

nnarkhed-mn:kafka-git-idea nnarkhed$ ./bin/kafka-run-class.sh kafka.tools.StateChangeLogMerger

Provide arguments to exactly one of the two options "[logs]" or "[logs-regex]"

Option Description

------ -----------

--end-time <end timestamp in the The latest timestamp of state change

format java.text. log entries to be merged (default:

SimpleDateFormat@f17a63e7> 9999-12-31 23:59:59,999)

--logs <file1,file2,...> Comma separated list of state change

logs or a regex for the log file

names

--logs-regex <for example: /tmp/state- Regex to match the state change log

change.log\*> files to be merged

--partitions <0,1,2,...> Comma separated list of partition ids

whose state change logs should be

merged

--start-time <start timestamp in the The earliest timestamp of state change

format java.text. log entries to be merged (default:

SimpleDateFormat@f17a63e7> 0000-00-00 00:00:00,000)

--topic <topic> The topic whose state change logs

should be merged

RabbitMQ vs Kafka

About Test

In this test we want to compare rabbitmq and kafka consumer message throughout.

Test Environment

RabbitMQ Version 3.3.0

Kafka Version 0.8.1

Message producer was running on Vqctd-hadoopas1.phx01.ebayadvertising.com

Message consumer was running on vdeap1.phx01.ebayadvertising.com

Kafka, RabbitMQ, Zookeeper were installed on phx-ctd-qa1.shopping.com

Producer Process was running with 10 Threads.

Consumer Process was a single thread.

For RabbitMQ we used direct exchange.

For Kafka we used 5 partitions and async message transfer for producer.

Tested with User Profile JSON object(approx. size 1KB).

Test Summary

In our tests we observed Kafka Consumer is approximately 2 times faster than RabbitMQ.

RabbitMQ producer is faster than Kafka producer.

RabbitMQ consumer throughput degrades under load.

Single Kafka Consumer was able to process 140K messages per sec. In same environment RabbitMQ consumer was able to reach 61K.

Attached test XLS (Kafka vs RabbitMQ.xls)

**QA: kafka for HttpLayer**

**QA steps**

Basic Functionality

Deploy anthill build for HttpLayer

Restart tomcat

Observe catalina.out log to verify application started without errors

Submit rover requests using testEvent page

Observe httplayer.log and verify there are no errors during publish

Using kafka system tool, verify that messages are being published to kafka

kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic qa.pixel.queue --time -1

Use monitoring tool and verify counts HttpLayer SPO#Monitoring

Run PixelListener

Verify that pixel listener consumes messages from kafka

Refer this for testing pixel listener: QA steps - kafka for RTS

Network Failure Scenario

Stop kafka QA. Alternatively, modify metadata.broker.list parameter in application\_global.properties to point to non-existing hostname.

Verify that following parameters have been set correctly in kafka producer properties list

queue.buffering.max.messages

queue.enqueue.timeout.ms (should be set to 0)

message.send.max.retries

retry.backoff.ms (should be high enough ~10000 in order to simulate network slowness)

Submit multiple rover requests. Number of requests should reach greater than queue.buffering.max.messages value

Use monitoring tool to observe following counts

dropped message count

failed send count

resend count

Verify that messages are dropped/discarded and rover is unaffected

Health-check

Set following parameters in application\_global.properties to desired values

dropMinuteRateMax

failedSendMinuteRateMax

Simulate above network failure scenario such that droppedMessagePerMinute reaches higher than set threshold

Verify that health-check returns unhealthy

Simulate above network failure scenario such that failedSendMinuteRateMax reaches higher than set threshold

Verify that health-check returns unhealthy