KIP-73 Replication Quotas

Contents:

Status

Current state: Adopted

Jira: KAFKA-1464. here is no specific config for the number of throttled replica fetcher threads as the config for the number of replica

Relates to: KIP-13: Quotas, KIP-74: Add Fetch Response Size Limit in Bytes

Mail Thread: here

Released: 0.10.1.0

Revision History

- 10th Aug 2016: Switched from a delay-based approach, which uses dedicated throttled fetcher threads, to an inclusion-based approach, which puts throttled and unthrottled replicas in the same request/response
- 25th Sept 2016: Split throttled replica list into two properties. One for leader side. One for follower side.
- 30th Sept 2016: Split the quota property into two values, one for the leader and one for the follower. This adds consistency with the replicas property changed previously.

Motivation

Currently data intensive admin operations like rebalancing partitions, adding a broker, removing a broker or bootstrapping a new machine create an unbounded load on inter-cluster traffic. This affects clients interacting with the cluster when a data movement occurs.

The intention of this proposal is to provide an upper bound on this traffic, so that clients can be guaranteed a predictable level of degradation, regardless of which partitions are moved, whilst ensuring it’s easy for admins to reason about progress of a replica movement operation.

Throttles are proposed on both leader- and follower-sides of the replication process so that replica movements, which are asymmetric, can be guaranteed on any single machine. That's to say, admin tasks, such as rebalancing, can result in the movement of a replicas where the number of bytes read from some brokers and the number of bytes written to others is not uniform, necessitating throttles on both sides.

Goals

- The administrator should be able to configure an upper bound on the "effect" replication traffic (moving partitions, adding brokers, shrinking brokers) has on client traffic (writes & reads) so clients get a guaranteed throughput, regardless of moving partitions or bootstrapping brokers.
- The administrator should be able to modify throttle bounds with no restart, so they can speed up or slow down replica movement tasks. For example a move might be progressing too slowly, might not be making progress, or a client be seeing too much service degradation.
- The throttle bound should be strictly upheld on all brokers (both transmitted and received), regardless of the arrangement of leaders and followers involved, so that the behaviour is intuitive and ubiquitous from the admin's perspective.

- A Kafka Admin, who wishes to strictly bound the effect replica rebalancing will have on clients to X% of the cluster's utilisation, but also reliably predict that specific rebalancing processes will complete in Y seconds, should be able to easily determine the appropriate throttle setting to use based on X & Y.

Background Concepts

The replication process has some important properties. The follower has a single thread per leader. This sends fetch requests, and waits for responses. The fetch request contains a list of all partitions the follower is interested in, and they are returned in a single fetch response. The leader processes fetch requests, from each follower, serially, to ensure strict ordering.

The follower puts a cap on the size of the request by passing a max.bytes field (aka fetchSize) in the request (set to replica.fetch.max.bytes). This doesn't apply to the request as a whole. It applies to each partition returned. So the size of the response will be dependent on this setting multiplied by the number of partitions being replicated.

Kafka already includes the concept of throttling (via Quotas), applied to Client traffic. This mechanism, when enabled, ensures a single client will not exceed the certain bandwidth figure, specified for produce and/or fetch traffic. The quota value, can be changed at runtime for each client.

The current Quotas implementation works by delaying requests for an appropriate period of time, if the quota is breached. The mechanism is somewhat similar to Purgatory. If the request should be throttled, an appropriate delay is calculated. The request is then dequeued until that time period elapses, after which the response is sent back to the client. An equivalent approach is used for produce requests.

Proposed Changes

The proposed solution can be split, conceptually, into two parts: The throttle mechanism itself, and the invocation process used for defining which replicas should be throttled. Brokers are provided a list of replicas to throttle and a quota value, in B/s, which must not be exceeded by total throttled traffic on that broker. These settings are used to throttle replication traffic for those partitions.

Throttles are applied on both leaders and followers separately.

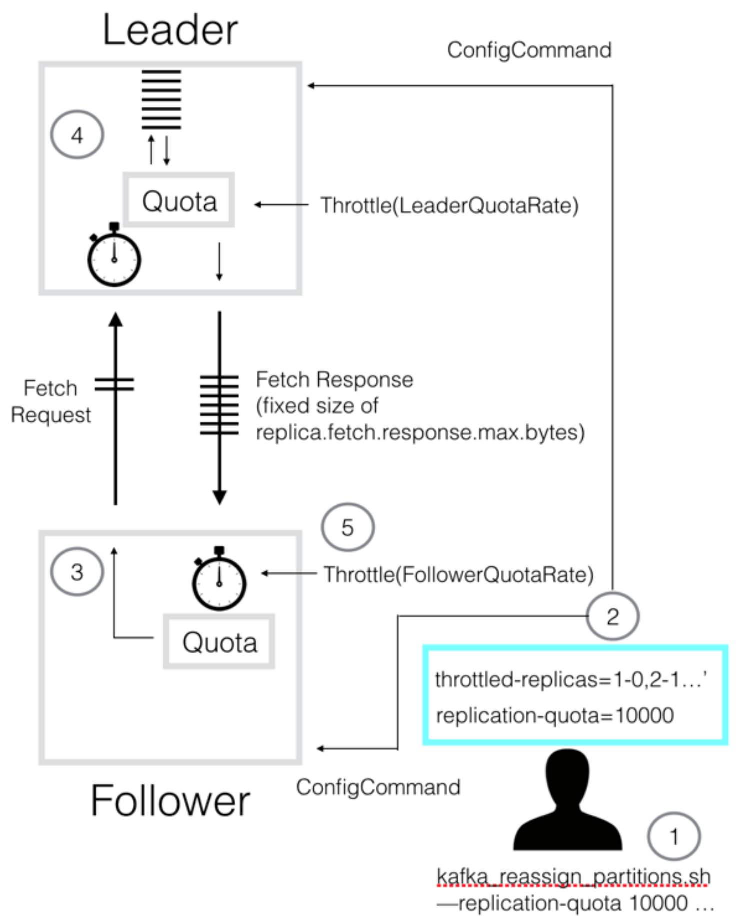
The rebalancing script, kafka-reassign-partitions.sh, will be altered to determine (internally) which replicas will be moved, as well as allowing the admin to define an appropriate quota, using a new command line argument.

The Throttle Mechanism

The throttle is applied to both leaders and followers. This allows the admin to exert strong guarantees on the throttle limit, applied to both transmitted and received bytes, regardless of the distribution of partitions in a move.

This is best understood diagrammatically:

1. The admin initiates a rebalance or other admin task, specifying a maximum bandwidth for replication on any single broker.
2. The throttle value, and the set of replicas affected, are transmitted to brokers.
3. The follower includes throttled partitions in the fetch request only if the FollowerQuotaRate has not been exceeded. Requests are now fixed size and the Fetcher randomises the order that partitions passed in the request to ensure fairness.
4. The leader processes partitions in the order defined in the request. Throttled partitions are excluded from the response if the LeaderQuotaRate is exceeded.
5. The follower reads the response and increases the quota by the received bytes. The process repeats (goto 3.).


Leader
ConfigCommand
4
Quota
Throttle(LeaderQuotaRate)
Fetch Request
Fetch Response (fixed size of replica.fetch.response.max.bytes)
5
3
Throttle(FollowerQuotaRate)
Quota
2
throttled-replicas=1-0,2-1…'
replication-quota=10000
ConfigCommand
Follower
1
kafka_reassign_partitions.sh —replication-quota 10000 …

So there are two quota mechanisms, backed by separate metrics. One on the follower, one on the leader. The leader tracks the rate of requested bytes (LeaderQuotaRate). The follower tracks the throttled bytes allocated to fetch responses for throttled replicas (FollowerQuotaRate).

The follower makes a requests, using the fixed size of replica.fetch.response.max.bytes as per KIP-74. The order of the partitions in the fetch request are randomised to ensure fairness.

When the leader receives the fetch request it processes the partitions in the defined order, up to the response's size limit. If the inclusion of a partition, listed in the leader's throttled-replicas list, causes the LeaderQuotaRate to be exceeded, that partition is omitted from the response (aka returns 0 bytes). Logically, this is of the form:

```
if (throttled(partition))

   var includedInFetchResponse: Boolean = quota.recordAndEvaluate(bytesRequestedForPartition)
```

When the follower receives the fetch response, if it includes partitions in its throttled-partitions list, it increments the FollowerQuotaRate:

```
var includeThrottledPartitionsInNextRequest: Boolean = quota.recordAndEvaluate(previousResponseThrottledBytes)
```

If the quota is exceeded, no throttled partitions will be included in the next fetch request emitted by this replica fetcher thread.

This mechanism is optimistic. That’s to say a large cluster could exceed the quota with the first request, only then will a throttled partitions be omitted from future requests/responses, to bring the rate down to the desired throttle value. To bound this issue we limit the size of requests to a fixed and configurable bound. This uses a new config, covered in KIP-74: replica.fetch.response.max.bytes. This config must be tuned by the admin to ensure that the initial set of requests (a) does not cause the quota to be violated on the first request and (b) will return a response within the configured window. However, the default value, of 10MB, is small enough to support leader quotas over 1MB/s (see Q&A 5 below).

The Invocation Process

The standard dynamic config mechanism is used to define which replicas will be throttled and to what throughput. This is covered by two separate configs:

1. A list of replicas that should be throttled. This takes the form [partitionId]-[replicaId],[partitionId]-[replica-id]...

2. The quota for a broker. For example 10MB/s.

The admin sets the throttle value when they initiate a rebalance:

```
kafka-reassign-partitions.sh --execute … --replication-quota 10000
```

The tool, kafka-reassign-partitions.sh, calculates a mapping of topic -> partition-replica for each replica that is either (a) a move origin or (b) a move destination. A leader throttle is applied to all existing replicas that are moving. A follower throttle is applied to replicas that are being created as part of the reassignment process (i.e. move destinations). There are independent configs for the leader and follower, but this is wrapped in kafka-reassign-partitions.sh so the admin only need be aware of these if they alter them directly via kafka-configs.sh.

```
leader.replication.throttled.replicas = [partId]:[replica], [partId]:[replica]...
```

```
follower.replication.throttled.replicas = [partId]:[replica], [partId]:[replica]...
```

```
leader.replication.throttled.rate = 1000000
```

```
follower.replication.throttled.rate = 1000000
```

The admin removes the the throttle from Zookeeper by running the --verify command after completion of the rebalance.

Alternatively each property can be set independently using kafka-configs.sh (see below)

Public Interfaces

FetchRequest

A new field is required in the fetch request to bound the total number of bytes within the fetch response. This is covered by KIP-74

Metrics

The metrics reuse the Kafka Metrics (rather than Yammer) to be inline (and reuse the windowing functionality of) the existing Client Quotas implementation. This is the list of metrics we add as part of this change:

- LeaderReplicationThrottledRate: The rate of throttled replication for transmitted bytes from a broker.
- FollowerReplicationThrottledRate: The rate of throttled replication for transmitted bytes into a broker.
- PartitionBytesInRate: Equivalent to BytesInPerSec, but at a partition level (i.e. total traffic - throttled and not throttled). This is required for estimating how long a rebalance will take to complete. B/s. See usability section below.
- SumReplicaLag: This is the sum of all replica lag values on the broker. This metric is used to monitor progress of a rebalance and is particularly useful for determining if the rebalance has become stuck due to an overly harsh throttle value (as the metric will stop decreasing).

Config & Zookeeper

Topic-level dynamic config (these properties cannot be set through the Kafka config file, or on topic creation)

```
bin/kafka-configs … --alter
--add-config 'leader.replication.throttled.replicas=[partId]-[replica], [partId]-[replica]...'

--entity-type topic
--entity-name topic-name
```

```
bin/kafka-configs … --alter
--add-config 'follower.replication.throttled.replicas=[partId]-[replica], [partId]-[replica]...'

--entity-type topic
--entity-name topic-name
```

Broker-level dynamic config (these properties cannot be set through the Kafka config file), unit=B/s

```
bin/kafka-configs … --alter
--add-config 'leader.replication.throttled.replicas=10000'

--entity-type broker
--entity-name brokerId
```

```
bin/kafka-configs … --alter
--add-config 'follower.replication.throttled.
replicas=10000'
--entity-type broker
--entity-name brokerId
```

Wildcard support is also provided for setting a throttle to all replicas:

```
bin/kafka-configs … --alter
--add-config 'leader.replication.throttled.replicas=*'
--entity-type topic
```

And to set a ubiquitous throttle value to all brokers:

```
bin/kafka-configs … --alter
--add-config 'leader.replication.throttled.
rate=10000'
--entity-type broker
```

The ‘replication-quota’ is only applied to ‘throttled-replicas’.

Here we add the concept of a dynamic config, applied at a broker level. This is equivalent, in implementation, to the existing entity-type = client configuration, but applied at a broker level and available for dynamic change.

NB - whilst it is possible to change throttle configs in this way, there should not be any requirement for admins to use kafka-configs directly when rebalancing partitions, as this will be handled internally within kafka-reassign-partitions.sh. The admin would be required to use this interface to throttle a bootstrapping broker. The mechanism for doing this is described in the Q&A below.

These are reflected in zookeeper via a new Zookeeper path: /config/broker/[broker-id]

```
//Sample configuration for throttled replicas
{
 "version":1,
 "config": {
  "leader.replication.throttled.replicas":"0:0,0:1,0:2,1:0,1:1,1:2"

 }
}
```

```
//Sample configuration for throttled replicas
{
 "version":1,
 "config": {
  "follower.replication.throttled.replicas":"0:0,0:1,0:2,1:0,1:1,1:2"

 }
}
```

```
//Sample configuration for replication-quota
{
 "version":1,
 "config": {
  "replication-quota":"1000000"
 }
}
```

```
// Change notification for replication-quota
{
 "version":1,
 "entity_path": "/config/broker/"
}
```

Inline with client-quota, two configs are provided to control the window used for ThrottledRateIn/Out.

```
replication.quota.window.num = The number of samples to retain in memory (default 11)

replication.quota.window.size.seconds = The time span of each sample (default 1)
```

Script: kafka-reassign-partitions.sh

This will be altered to add an additional, optional parameter:

```
kafka-reassign-partitions.sh --execute … --replication-quota 1000
```

Where the replication-quota is: the maximum bandwidth, in B/s, allocated to moving replicas. This parameter is only valid when used in conjunction with --execute. If omitted, no quota will be set.

In addition the kafka-reassign-partitions script will include in its output (from --generate option only) a MoveRatio:

```
MoveRatio = #partitions-to-be-moved / #total-partition-count
```

This is simply a number which the admin can use to estimate how long a rebalance will take. See the Q&A section below.

Test Plan

System tests to include:

Given a static two node cluster (p=100, r=2)

When data is moved from one node to the other

Then replicas should move at the quota dictated rate.

Given a two node cluster with incoming produce requests (p=100, r=2)

When data is moved from one node to the other

Then replicas should move at the quota dictated rate - the inbound rate.

Given a three node cluster (p=100, r=2)

When data is moved from one node to the other two nodes

Then replicas should move at the quota dictated rate.

[repeat with produce requests]

Given a three node cluster (p=100, r=2)

When data is moved from two nodes to a single node

Then replicas should move at the quota dictated rate.

[repeat with produce requests]

Amendments

While testing KIP-73, we found an issue described in https://issues.apache.org/jira/browse/KAFKA-4313. Basically, when there are mixed high-volume and low-volume partitions, when replication throttling is specified, ISRs for those low volume partitions could thrash. KAFKA-4313 fixes this issue by avoiding throttling those replicas in the throttled replica list that are already in sync. Those in-sync replicas traffic will still be accounted for the throttled traffic though.

Q&A

1. How should an admin set the throttle value when rebalancing partitions?

First consider how much network bandwidth you are prepared to give up for rebalancing. Looking at your broker's network utilisation is a good way to do this. If your network is saturated, whatever value you pick here will translate into a proportional impact on your clients, so if you set --replication-quota to [30% of your network utilisation] your clients would see a maximum degradation of 30%.

On the flip side you need to ensure enough progress so that your rebalance completes in some known amount of time. You can estimate this is by observing two metrics. max(BytesInPerSec) & log.topic.partition.Size. kafka-reassign-partitions.sh includes, in its output, the proportion of partitions that will be moved as part of the rebalance. The MoveRatio. Using this you can calculate how long the move will take by calculating:

```
MoveTime = MoveRatio x TotalLogSizePerBroker x #Brokers / (replication-quota - max(BytesInPerSec))
```

If the quota is set aggressively, compared to the inbound rate, it is still possible for you to get into a situation where you will never make progress on one or more brokers. Specifically as (replication-quota - max(BytesInPerSec)) -> 0. This could happen if replicas moved in such a way that max (BytesInPerSec) increased or followers became concentrated on a single broker such that their total “keep up” traffic exceeds the throttle value or simply because you have had an unexpected increase in load. This should be relatively rare and is easy to deal with. The administrator monitors a new metric SumReplicaLag. If this stops reducing, before rebalancing completes, then the admin must simply increase the throttle value a little to compensate.

2. How do I throttle a bootstrapping broker?

Bootstrapping a broker isn’t specifically covered by any Kafka tooling but you can still configure a throttle manually for this admin operation. Pick a throttle value using the same method described for reassigning partitions (i.e. above). Then set the replication-quota using kafka_configs.sh.

Next work out which partitions are assigned to the broker(s) being bootstrapped. Assign these replicas to the throttled-replicas config using kafka_configs.sh.

Use the SumReplicaLag metric to monitor progress, looking out for partitions not making progress. Make adjustments if necessary.

Finally, remove the configs when the bootstrapping broker is fully caught up.

3. Could an ISR replica ever be throttled? Could a leader ever be throttled?

Yes. An in-sync-replica could be throttled when it catches up, before the config value is removed. This should be OK. The replica had enough bandwidth allocated to catch up in the first place. But producers writing to throttled leaders could incur, what should be a short, delay.

It is also possible to throttle a leader. Either a ‘caught up’ replica could become a leader via automated leader balancing, in which case it would be throttled. We discuss the sizing of the throttle, for this case, in the next question.

4. Do we need to disable auto leader rebalancing when moving partitions or bootstrapping a broker?

So long as the throttle is set to the same value on all nodes, it should not matter if leaders move. There could potentially be an issue if an asymmetric throttle was applied (for example setting a higher throttled throughput on a bootstrapping broker, and lower throttle values on other brokers). This would result in a lack of progress and the throttle value would have to be changed dynamically to allow progress to resume.

The admin can prevent this issue by checking the throttle value is set correctly. If the incoming bytes per broker is IN and the throttle value is T on a network of throughput N with replication factor R, we know for any broker:

```
IN < T < N - IN/R
```

Intuitively this means the throttle must be greater than the write rate, so there is bandwidth for both Catch-Up and Keep-Up traffic. Conversely, the throttle must be low enough to allow inbound user traffic (which will not be throttled) assuming all leaders have rebalanced (think bootstrapping broker as the most extreme case. Here IN/R is the proportion of inbound traffic for these leaders)).

5. Do I need to alter the default value for replica.fetch.response.max.bytes?

Generally you should not need to alter the default fetch request size. The critical factor is that the initial set of requests, for throttled replicas, return in the configured window duration. By default the total window is 11 x 1s and replica.fetch.response.max.bytes = 10MB. Intuitively the constraint is:

```
#brokers x replica.fetch.response.max.bytes / min(QuotaLeader x #brokers, NetworkSpeed) < total-window-size
```

So, for the majority of use cases, where the sum of leader throttles < NetworkSpeed we can reduce this to:

```
replica.fetch.response.max.bytes < QuotaLeader x total-window-size
```

So consider a 5 node cluster, with a leader quota of 1MB/s, and a window of 10s, on a GbE network. The leader's throttle dominates, so the largest permissible replica.fetch.response.max.bytes would be 1MB/s * 10s = 10MB. Note that this calculation is independent of the number of brokers. However if we had a larger cluster, of 500 nodes, the network on the follower could become the bottleneck. Thus we would need to keep replica.fetch.response.max.bytes less than total-window-size * NetworkSpeed / #brokers = 10s * 100MB/s / 500 = 2MB.

Rejected Alternatives

There appear to be two sensible approaches to this problem: (1) omit partitions from fetch requests (follower) / fetch responses (leader) when they exceed their quota (2) delay them, as the existing quota mechanism does, using separate fetchers. Both appear to be valid approaches with slightly different design tradeoffs. The former was chosen as the underlying code changes are simpler (based on explorations of each). The details of the later are discussed here.

We also considered a more pessimistic approach which quota's the follower's fetch request, then applies an adjustment when the response returns. This mechanism has some advantages, most notably it is conservative, meaning the throttle value will never be exceeded. However, whilst this approach should work, the adjustment process adds some complexity when compared to the optimistic approaches. Thus this proposal was rejected (This is discussed in full here).