KIP-1010: Topic Partition Quota

Status

Current state: "Under Discussion"

Discussion thread: here

JIRA:

Unable to render Jira issues macro, execution error.

Unable to render Jira issues macro, execution error.

Unable to render Jira issues macro, execution error.

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

In KIP-13, which introduced Kafka's network bandwidth quotas, the byte-rate threshold configurations are based on individual users and/or clients. This strategy, while effective in mitigating the impact of misbehaving clients on others within the same topic, introduces a trade-off in terms of quota distribution.

The current per-client quota method ensures that a single misbehaving client doesn't unintentionally affect multiple clients functioning as producers or consumers within the same topic. However, this approach has limitations, particularly when **multiple partitions of the same topic are located on the same broker**. For example, consider a topic (T) with 6 partitions, and an **expected** total write throughput of 12MBps. Ideally, each partition should receive a 2Mbps quota, but the current implementation faces challenges. If a broker hosts 2 leader partitions for topic T, the collective throughput for that topic on the broker is limited to 2MBps, rather than anticipated 4MBps.

blocked URL

Given that clients are often unaware of the partition distribution of topics, which can dynamically change with tools like Cruise Control, there is room for a more **consistent experience for producers and consumers**. This KIP proposes an alternative implementation wherein network bandwidth quotas can also be defined on a per-topic-partition basis. This would offer greater flexibility in aligning quotas with the specific characteristics of individual topic partitions, potentially enhancing the overall control and predictability of Kafka's network bandwidth management.

Public Interfaces

Metrics

For the introduction of topic-partition quotas, we must now record the byte rate and throttle time for a particular topic-partition. To accommodate this, we suggest extending the existing public `ClientQuotaCallback` interface by adding a new method for `quotaMetricTags`. This new method overload includes an extra parameter for TopicPartition. In order to maintain backward compatibility with custom implementations of this interface, we propose that the default implementation of this method should ignore the new parameter and simply call the original `quotaMetricTags`.

Quota Configuration

To support produce and consume quota values per topic, we propose adding two configurations to the Kafka client's `TopicConfig` and `QuotaConfig`. The first configuration enables the introduction of a new setting in the topic, while the second allows the utilization of that setting in quota management. To align with the established topic configuration naming convention, we recommend using dot-separated names such as `producer.byte.rate` and `consumer.byte.rate`

Default Quota Configuration

In Kafka, default configurations are currently defined for user/client but not for topics. To align with the familiar pattern of falling back to default quota values, we suggest introducing this feature to topic configuration. Implementing this involves making adjustments in `ConfigCommand`, accommodating `<default>` in topic name validation, and making necessary updates in places like `AdminManager` implementations.

Proposed Changes

Following the same pattern as KIP-13, we categorize the changes around Quota Policy, Distribution, Metrics, Quota Actions and Configuration Management to highlight the differences.

Quota Policy

Similar to the original implementation, we propose to throttle based on the client-id. The primary distinction lies in our extended approach, where we assess whether a request should be throttled based on the byte-rate associated with each topic-partition produced or consumed within that request. In other words, alongside capturing the byte rate based on the client and user-id metrics in the client sensor, we now also register the byte rate for each specified topic-partition in the request. Regardless of whether the topic-partition quota is enabled or not, this metric also aids in monitoring traffic per partition and may have additional use cases.

Additionally, we suggest setting default quota values for topic-partitions and providing the option to override them for specific topics, similar to the existing user/client quota system.

Quota Distribution

Following the current method, Kafka sets quotas per broker, avoiding the complexities of sharing quota usage across all brokers. Our extended approach maintains this principle. The only adjustment is that, by specifying quotas with more detail for topic-partitions, we aim to prevent early throttling issues when multiple partitions share the same broker.

Quota Actions

We're not suggesting any alterations to the current approach. The only small adjustment is the necessity to offer more detailed information about the size of produced/requested topic partitions for quota management.

Metrics

Following the original implementation, we are using the standard Kafka metrics mechanism with sensors. The only change involves introducing new metric tags to record the produce/consume byte rate, including topic-name and partition-id. A positive outcome of this modification is the ability to monitor traffic per topic per partition in clusters with the new mechanism enabled.

Configuration Management

Even though we're implementing throttling based on individual partitions in a topic, we assume that all partitions in that topic share the same quotas for both production and consumption. To manage this, we suggest adding new configurations—like `producer.byte.rate` and `consumer.byte.rate`—specifically for quotas.

Just like other topic settings, you can easily adjust these configurations using `kafka-config.sh`, and it works seamlessly with both Zookeeper and Kraft implementations.

For default topic-partition quotas, we propose using `<default>` as a special topic name. This makes it easy to set up quotas for `producer.byte.rate`, `consumer.byte.rate`, and potentially other default topic settings in the future.

Dynamic Configuration

Similar to the dynamic user/client quota configuration, we aim to make topic quota configuration dynamic as well. To achieve this, the `QuotaManager` needs to be notified of any changes. This necessitates adjustments in the `ConfigHandler` class.

Partition Size per Request

To accurately measure the size of the topic-partition per request, we propose making the `partitionSizes` in `ProduceRequest` public and implementing a corresponding method in the `FetchResponse` class, utilizing `MessageSizeAccumulator`. These methods can then be employed in `KafkaApis` to calculate the individual partition size for both produce and consume requests, subsequently providing this information to quota management methods like `maybeRecordAndGetThrottleTimeMs` and `getMaxValueInQuotaWindow`.

Quota Management

To dynamically support both user/client quotas and topic/partition quotas, we suggest extending the `ClientQuotaManager` to track overall request size and record it per user/client (as in the current quota implementation), as well as individual topic-partition size recorded per topic per partition. This approach allows the quota manager to make decisions on request throttling based on whether a user/client quota, a topic-partition quota, or both have been reached.

Compatibility, Deprecation, and Migration Plan

The introduced feature maintains backward compatibility by extending the existing implementation without removing user/client quota behavior or altering any current configurations. The only major change is the collected metrics for the topic-partition byte-rate in produce and consume requests. If there are concerns regarding heap size, we can potentially introduce a feature flag to halt the collection of these events.
For migration, users can simply configure the topic with the new quota settings and utilize the feature independently or in conjunction with user/client quotas.

Test Plan

Integration and unit tests would cover most of the functionalities.

For performance test, we should be able to use the current numbers as the baseline and compare with the topic-partition quota enabled.

Rejected Alternatives

1. *No Default Config for Topic-Partition Quota*: While this option would simplify implementation, it would require defining quotas for every new topic or providing unlimited quotas (assuming user/client quotas are not used).
2. *Static per-broker limit*: Although this will address the broker limit, it does not fit the multi-tenant clusters and the noisy-neighbor problem.