KIP-389: Introduce a configurable consumer group size limit

Status

Current state: *Accepted*

Discussion thread: Here

JIRA: ⚠ Unable to render Jira issues macro, execution error.

This KIP is part of a series of related proposals which aim to solidify Kafka's consumer group protocol

KIP-345: Introduce static membership protocol to reduce consumer rebalances

⚠ *Unable to render Jira issues macro, execution error.*

KIP-394: Require member.id for initial join group request

Motivation

Consumer groups are an essential mechanism of Kafka. They allow consumers to share load and elastically scale by dynamically assigning the partitions of topics to consumers. In our current model of consumer groups, whenever a rebalance happens every consumer from that group experiences downtime - their `poll()` calls block until every other consumer in the group calls poll(). That is due to the fact that every consumer needs to call `JoinGroup` in a rebalance scenario in order to confirm it is still in the group.

Today, if the client has configured `max.poll.interval.ms` to a large value, the group coordinator broker will take in an unlimited number of join group requests and the rebalance could therefore continue for an unbounded amount of time.
Further, large consumer groups are not very practical with our current model due to two reasons:
1. The more consumers there are, the likelier it is that one will fail/timeout its session, causing a rebalance
2. Rebalances are upper-bounded in time by the slowest-reacting consumer. The more consumers, the higher the chance one is slow (e.g called `poll()` right before the rebalance and is busy processing the records offline). This means that rebalances are more likely to be long-lived and disruptive to consumer applications.

The bigger problem is the potential risk described in ⚠ Unable to render Jira issues macro, execution error. where N faulty (or even malicious) clients could result in the broker thinking more than N consumers are joining during the rebalance. This has the potential to burst broker memory before the session timeout occurs and puts additional CPU strain on the Coordinator Broker - causing problems for other consumer groups using the same coordinator.
The root of the problem isn't necessarily the client's behavior (clients can behave any way they want), it is the fact that the broker has no way to shield itself from such a scenario.

Consumer Group Size Considerations

Large consumer groups can be seen as an anti-pattern. To summarize, we have the following concerns:

- Memory usage of stable groups is not very high, but the runaway consumer group scenario described in KAFKA-7610 can reach large consumer numbers very quickly and affect memory *(rough memory usage documented)*
- CPU spikes - there are a number of O(N) operations done on the consumers collection for a group
- Rebalance times do not grow linearly with the consumer group size - unfortunately we do not have any concrete results, just anecdotes. It is recommended to be wary of rebalance frequencies and duration when consumer counts reach hundreds

Proposition

We propose to address the critical stability issue via the addition of a configurable upper-bound for the number of consumers in a consumer group. Adding such a config will enable server-side protection against buggy/malicious applications.
It is also useful in the sense that this configuration gives Admin/Ops teams better control over the cluster, limiting the ways in which novice developers can shoot themselves in the foot (via large consumer groups).

Public Interfaces

Add a new cluster-level group.max.size config with a default value of `*Int.MAX_VALUE*`.

Add a new response error:

Errors.java

```
GROUP_MAX_SIZE_REACHED(77, "Consumer group is already at its full capacity.",
 GroupMaxSizeReachedException::new);
```

Proposed Changes

We shall block registration of new member once a group reaches its configured capacity. Any subsequent `JoinGroup` requests will receive a response with the `GROUP_MAX_SIZE_REACHED` error.

Since the cap should never be reached in practice, the consumer will fatally exit upon receiving this error message.

When the Coordinator loads the consumer groups state from the log, it will force a rebalance for any groups that cross the max.size threshold so that the newly-formed generation will abide by the size constraint.

Compatibility, Deprecation, and Migration Plan

This is a backward compatible change. Old clients will still fail by converting the new error to the non-retriable UnknownServerException

Migration Plan

When upgrading to the new version with a defined `group.max.size` config, we need a way to handle existing groups which cross that threshold.
Since the default value is to disable the config, users who define it should do their due diligence to shrink the consumer groups that cross it or expect them to be shrunk by Kafka.

Rejected Alternatives

- Topic-level config
  - It is harder to enforce since a consumer group may touch multiple topics. One approach would be to take the min/max of every topic's group size configuration.
  - This fine-grained configurability does not seem needed for the time being and may best be left for the future if the need arises
- There are other ways of limiting how long a rebalance can take, discussed here
  - In the form of time - have a max rebalance timeout (decoupled from `max.poll.interval.ms`)
    - Lack strictness, a sufficiently buggy/malicious client could still overload the broker in a small time period
  - In the form of memory - have a maximum memory bound that can be taken up by a single group
    - Lacks intuitiveness, users shouldn't think about how much memory a consumer group is taking
- Default value of 250
  - Large consumer groups are currently considered an anti-pattern and a sensible default value would hint at that well
  - It is better to be considerate of possible deployments that already pass that threshold. A Kafka update shouldn't cause disruption
- High default value (5000)
  - This might mislead users into thinking big consumer groups aren't frowned upon
- Do not force rebalance on already-existing groups that cross the configured threshold
  - Groups will therefore eventually get shrunk when each consumer inevitably gets restarted and is unable to join the already-over-capacity group

- Users might perceive this as unintuitive behavior
- Since we settled on a default value that disables the functionality, it is reasonable to be more strict when the config is defined