KIP-74: Add Fetch Response Size Limit in Bytes

Status

Current state: Accepted

Discussion thread: *here*

JIRA: *KAFKA-2063*

Released: 0.10.1.0

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

Motivation

Currently the only possible way for client to limit fetch response size is via per-partition response limit **max_bytes** taken from config setting **max.partition.fetch.bytes**.

So the maximum amount of memory the client can consume is **max.partition.fetch.bytes * num_partitions**, where **num_partitions** is the total number of partitions currently being fetched by consumer.

This leads to following problems:

1. Since **num_partitions** can be quite big (several thousands), the memory required for fetch responses can be several GB
2. **max.partition.fetch.bytes** can not be set arbitrarily low since it should be greater than maximum message size for fetch request to work.
3. Memory usage is not easily predictable - it depends on consumer lag

This KIP proposes to introduce new version of fetch request with new top-level parameter **max_bytes** to limit the size of fetch response and solve above problem.

In particular, if consumer issues **N** parallel fetch requests, the memory consumption will not exceed **N * max_bytes**.

Actually, it will be min(**N * max_bytes**, **max.partition.fetch.bytes * num_partitions**) since per-partition limit is still respected.

Public Interfaces

This KIP introduces:

- New fetch request (v.3) with response size limit
- New client-side config parameter **fetch.max.bytes** - client's fetch response size limit
- New replication config parameter **replica.fetch.response.max.bytes** - limit used by replication thread
- New inter-broker protocol version "**0.10.1-IV0**" - starting from this version brokers will use fetch request v.3 for replication

Proposed Changes

Proposed changes are quite straightforward. We introduce FetchRequest v.3 with new top level parameter **max_bytes**:

```
Fetch Request (Version: 3) => replica_id max_wait_time min_bytes max_bytes [topics]
  replica_id => INT32
  max_wait_time => INT32
  min_bytes => INT32
  max_bytes => INT32
  topics => topic [partitions]
    topic => STRING
    partitions => partition fetch_offset max_bytes
      partition => INT32
      fetch_offset => INT64
      max_bytes => INT32
```

Fetch Response v.3 will remain the same as v.2.

Server processes partitions in order they appear in request.

Otherwise, for each partition except the first one server fetches up to corresponding partition limit **max_bytes**, but not bigger than remaining response limit.

For the first partition, server always fetches at least one message**.** Empty response limits will be returned for all partitions that didn't fit into response limit.

This algorithm provides following guarantees:

- FetchRequest always makes progress - if server has message(s), than at least one message is returned irrespective of **max_bytes**
- FetchRequest response size will not be bigger than max(**max_bytes**, size of the first message in first partition)

Since new fetch request processes partitions in order and stops fetching data when response limit is hit, client should use some kind of partition shuffling to ensure fairness.

Consider following example - suppose client want to fetch from 4 partitions: A, B, C, D. Suppose that partitions A and B are growing much faster than C and D.

If client is always fetching partitions in order A,B,C,D than it is possible that response limit is hit before any messages were fetched from C and D.

In this scenario client won't get any messages from C and D until it catches up with A and B.

The solution is to reorder partitions in fetch request in round-robin fashion to continue fetching from first empty partition received or to perform random shuffle of partitions before each request.

Round-robin shuffling seems to be more "fair" and predictable so we decided to deploy it at ReplicaFetcherThread and in Consumer Java API.

Compatibility, Deprecation, and Migration Plan

The new fetch request is designed to work properly even if the top level **max_bytes** is less than the message size. We decided to establish the following defaults:

fetch.max.bytes = 50MB

replica.fetch.response.max.bytes = 10MB

Rejected Alternatives

Some discussed/rejected alternatives:

1. Together with addition of global response limit deprecate per-partitions limit. Rejected since per-partition limit can be useful for Kafka streams (see mail list discussion).
2. Do random partition shuffling on server side. **Pros:** ensure fairness without client-side modifications. **Cons:** non-deterministic behaviour on server side; round-robin can be easily implemented on client side.