KIP-731: Record Rate Limiting for Kafka Connect

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: ⚠ 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

Kafka Connect suffers from a lack of Quality of Service (QoS) features that are typical of multi-tenant systems. We identify two related problems in this space:

First, "the noisy neighbor problem": one bad-behaving Task can cause issues for other Tasks on the same node or in the same cluster, by exhausting shared resources such as CPU and network bandwidth. An easy way to combat this problem in multi-tenant systems is to enforce quotas on individual tenants, but Connect does not support this.

Second, the "firehose" problem: Connect tends to run full-throttle and can overwhelm downstream systems. This is especially problematic with MirrorMaker, which will try to replicate entire Kafka clusters as quickly as possible.

These problems are unfortunate given Connect's common use-case: a system for connecting disparate systems, including across regions and cloud providers, often using plugins from third-parties. It is impractical to assume that each such external system has some quota mechanism of its own. Moreover, it is impractical to assume that each Connector implementation handles such external limits gracefully.

We have found that Kafka's own quota mechanism is insufficient to solve these problems for us. This is because the limits we are trying to honor are often in external systems -- not Kafka. For example, if we want to limit total throughput to some cloud region, a specific on-prem Kafka broker's per-topic and per-client quotas are largely irrelevant.

Public Interfaces

We propose the following new public interface:

```
/**
 * Throttles ConnectRecords based on configurable rate limits.
 *
 */
public interface RateLimiter<R extends ConnectRecord> extends Closeable, Configurable {

    /**
     * Initialize and start rate-limiting based on the provided Time source.
     *
     */
    void start(Time time);

    /**
     * Account for the given recent record batch when subsequently throttling.
     */
    void accumulate(Collection<R> records);

    /**
     * How long to pause (throttle) in order to achieve the desired throughput.
     *
     */
    long throttleTime();

    /** Configuration specification for this RateLimiter */
    ConfigDef config();

    /** Signal that this instance will no longer will be used. */
    @Override
    void close();
}
```

Additionally, we propose to add the following configuration properties and metrics to Kafka Connect.

Connector Configuration

These apply to all Connectors (same as `name`, `tasks.max`, etc).

- **rate.limiters**: list of enabled RateLimiters (class names). Defaults to all baked-in RateLimiters: RecordRateLimiter, RecordBatchRateLimiter.
- **record.rate.limit**: max records-per-second throughput for each of this Connector's Tasks.
- **record.batch.rate.limit**: max record-batches-per-second throughput for each of this Connector's Tasks.

Task Metrics

- **record-rate-avg**
- **record-rate-max**
- **throttled-ratio**: Fraction of time spent throttling for any reason

Worker Metrics

- **throttled-ratio**: Fraction of time spent throttling for any reason across all Tasks on this Worker

Proposed Changes

We propose a new pluggable interface (RateLimiter) which enables operators to control throughput on a per-Task basis. Two built-in RateLimiters (RecordRateLimiter, RecordBatchRateLimiter) will provide basic functionality suitable for most use-cases.

By default, both built-in RateLimiters will be enabled but effectively no-ops unless `record.rate.limit` or `record.batch.rate.limit` are configured.

Additional custom RateLimiters can be enabled by implementing the RateLimiter interface and including related jars in the Connect plugin path (same as custom Connectors, SMTs, etc).

Compatibility, Deprecation, and Migration Plan

Defaults for these limits will be MAX_DOUBLE. Existing configuration will continue to work as it does today, unless these limits are changed to smaller values.

The proposed rate limiting will not step on the existing backoff/retry mechanism for failures, though these may interact to some degree. For example (an extreme case), if a SinkTask is limited to one batch per hour, this will supersede a backoff/retry max delay of 1 minute.

Rejected Alternatives

- We could attempt to rate-limit throughput based on bytes instead of record count. This would be more immediately useful for operating Connect, but there are some thorny issues with tracking byterate, since Connect doesn't have any view into the actual bytes sent to external systems. We anticipate that future work will address this. In the meantime, record rate is a good proxy for overall throughput.
- We could introduce an SMT to provide rate-limiting without modifying the Worker. This is slightly less user-friendly and limits what we can expose via metrics. For example, we want to ensure that `throttled-ratio` is emitted even if a rate-limiting SMT is not enabled for some Connector. Additionally, any pauses introduced via SMTs would be additive, but a collection of RateLimiters should only sleep by the max throttleTime returned from each RateLimiter.
- We could enforce limits across an entire Connector, instead of on a per-Task basis. This would require more coordination between Tasks than exists today. Given that tasks.max is already required, we think it is reasonable to expect an operator to multiply per-task limits by tasks.max to arrive at an approximate Connector-wide limit. However, we acknowledge that future work could make this friendlier.
- We could divide these limits by the number of actual tasks in a Connector in order to approximate Connector-wide limits. For example, a limit of 100 records/second with 20 tasks could yield 5 records/second per task. However, this assumes that workloads are perfectly balanced across tasks, which is often not the case in practice. So instead, we don't address Connector-wide limits for now, with the expectation that task-level limits are good enough. Again, we acknowledge that future work could improve this.
- We could leverage the existing Connector and Task state machine to automatically pause Connectors or Tasks that overstep a limit. However, we feel this is way too heavy-handed for throttling at high-frequency.
- We could throttle at very low frequency, e.g. by pausing a task for a long period of time if it oversteps a limit. This helps mitigate bad-behaving Tasks, but does not help us control the throughput of well-behaving Tasks operating near some external quota.
- We could just use broker quotas to limit throughput. However, broker quotas are meant to protect throughput to/from a specific broker, not an entire external system. It's much easier to reason about limiting bytes between Sources and Sinks.
- We could build backpressure and rate limiting into every Connector implementation we operate. This would work, but we feel all Connectors should benefit from these features out-of-the-box.
- We could drop the pluggable interface. This was my plan at first, but we foresee the need to implement custom RateLimiters, e.g. to better integrate with external control planes, or to provide more dynamic or adaptive rate limiting.