KIP-987: Connect Static Assignments

Status

Current state: Under Discussion

Discussion thread: *here*

JIRA: *here*

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

Motivation

Connect runs Connectors and Tasks, each is a "job", and all together form the "workload". In the standalone model, one worker handles the complete workload. In the distributed model, the workload is shared among several workers in the cluster, using a scheduling algorithm to assign each job to a host worker. Currently there are two scheduling algorithms:

- The `eager` protocol, uses unweighted, round-robin scheduling. This means that each job is given equal weight, and at each rebalance, the work is distributed to the workers by sending the `k-th` piece of work to the `k%n-th` worker. For a specific workload and cluster size, this assignment is deterministic.
- The `compatible` and `sessioned` protocols use unweighted, minimum-disruption scheduling. This means that each job is given equal weight, and at each rebalance, the work is distributed to workers by performing a minimum number of revocations and assignments relative to the existing assignment. For a specific workload and cluster size, this assignment is non-deterministic.

These algorithms perform well when both the workload and cluster are homogeneous, where each job consumes similar resources, and each worker provides equivalent resources. However, workloads in Connect are often heterogeneous, which cause these algorithms perform poorly. For example:

- Connectors typically consume less resources than even a single task of the same connector.
- Tasks with different plugins/implementations can have different resource usage profiles (high-memory vs high-cpu vs high-io)
- Tasks with different configurations can have different resource usages (such as buffer/linger/etc configurations)
- Tasks for the same configuration may distribute their workload unequally by design or by necessity

In all of these situations, the different jobs do not consume similar resources, and so the unweighted algorithms currently available can cause hot-spots to appear in the cluster when multiple high-consumption jobs are assigned to the same worker. Additionally in the `compatible` protocol, these hot-spots appear non-deterministically and can be difficult to remediate.

In order to manage these hot-spots, it is necessary to change the scheduling algorithm to take these constraints into account. But Connect is not situated to be able to manage resources directly, as the shared-JVM model does not permit strong isolation between threads.

If instead each job can be assigned to it's own worker, resource constraints can be specified at the process-boundary. Existing tooling for managing process resources can monitor and enforce resource utilization for that job by enforcing them on the worker containing that job. In order to make this possible, Connect should provide a mechanism for the user or management layer to assign jobs to specific workers, and a scheduling algorithm that respects these assignments.

This feature is written to be generic, such that it is usable manually and by any management system that users already use. To aid adoption and make this feature more functional for users on release, A sister proposal to the Strimzi project describes how this feature could be used to provide a resource-isolated cluster with a convenient user experience and upgrade story.

Public Interfaces

Workers will accept a new `connect.protocol` , `static` which is backwards-compatible with `sessioned` and will be the new default.

Workers will accept two new optional configurations, collectively known as the "static assignments":

`static.connectors` A list of connector names (e.g. connector-name) which are allowed to be executed by this node

`static.tasks` A list of task ids (e.g. connector-name-0) which are allowed to be executed by this node

If static assignments are not specified, or at least one worker in the cluster is not using the `static` protocol, they are ignored and the worker may receive an arbitrary assignment.

If a job is specified in the static assignments does not exist, it will be ignored. A `static.tasks` for a connector may be specified without a corresponding `static.connectors` .

Choosing static assignments

Users and management frameworks are expected to choose the values for these configurations with knowledge of their workload (i.e. set of connectors and tasks).

The workload can be read from the `GET /connectors` and `GET /connectors/{connector}` endpoints, which reveals both the set of connector IDs and task IDs. For closed-loop control, these endpoints should be polled regularly to discover changes in the number of tasks requested by each connector.

Alternatively if a fixed set of workers is desirable, the set of connectors and tasks can be inferred from the connector configurations alone. Each connector will provision at most `tasks.max` tasks, so workers for all of the tasks can be provisioned ahead-of-time. This has the downside that connectors which spawn fewer than the maximum number of tasks may receive static assignments for jobs which don't exist, possibly leaving some static workers idle.

For a new cluster without any connectors, a worker may be started with an empty assignment, assignments for the first connector to be created if they are known in advance, or an unsatisfiable assignment if they are not known. For subsequent connectors, the connector can be created before, during, or after static workers for that connector are added.

Proposed Changes

If the connect.protocol is set to static, each worker will send it's `static.connectors` and `static.tasks` to the coordinator during rebalances.

The leader will perform the following assignment algorithm if all members of the cluster support the `static` protocol.

1. Categorize workers with static assignments as "static workers" and those without as "wildcard workers".
2. Categorize connectors and tasks that are included in at least one static assignment are "static jobs", and those not included in any as "wildcard jobs".
3. Revoke any static jobs running on wildcard workers
4. Revoke any static jobs running on static workers which do not specify that job.
5. Revoke any wildcard jobs running on static workers.
6. Assign each unassigned static job to a static worker which specifies that job, choosing arbitrarily if there are multiple valid workers.
7. Revoke wildcard jobs from wildcard workers following the least-disruption algorithm assuming that all wildcard jobs (assigned or not) must eventually be evenly distributed among the wildcard workers.
8. Assign each unassigned wildcard job to a wildcard worker which is least loaded.

Note that the final two steps are the algorithm followed by the existing Incremental Cooperative Assignor, but limited in scope to the wildcard jobs & wildcard workers.

This has the properties that:

- A heterogeneous-protocol cluster, (with both `sessioned` and `static` protocol in-use), will still require workers with static assignments to run arbitrary assignments, as the leader may not support the `static` protocol. This preserves backwards-compatibility & rolling upgrades.
- A `static` protocol cluster with no static workers behaves identically to the `sessioned` protocol. This makes the static assignment feature opt-in even if the protocol is automatically upgraded by default.
- A cluster with both static and wildcard workers can be used in an ad-hoc manner to isolate specific jobs in a shared cluster.
    - Disruptive jobs may be given a separate worker to lessen interruptions to other jobs
    - Jobs may be temporarily given a static worker for additional instrumentation (debugging/metrics/etc)
    - Connector instances may be assigned to a static worker to improve round-robin balance of tasks among the wildcard workers
- A cluster with both static and wildcard workers can use wildcard workers as backups for disaster recovery.
- A cluster with both static and wildcard workers can be an intermediate state during a rolling upgrade to a cluster with only static workers.
- A cluster with only static workers can completely replace the internal unweighted scheduler with custom scheduling which includes resource usage estimates and heterogeneous workers.
- A cluster with only static workers can specify single tasks/connectors per worker to provide process and resource isolation between jobs.

Compatibility, Deprecation, and Migration Plan

After finishing an upgrade to a version which supports the Static Assignments feature, workers can individually be given assignments.

- If used in an ad-hoc manner, workers can be added to the cluster with a static assignment. If that additional worker then goes offline, the job will migrate back to the wildcard workers.
- If migrating to a cluster with only static workers, static workers can be added until the wildcard workers are drained, and then the wildcard workers can be removed.

Downgrading to a version which does not support the `static` protocol will cause any static assignments to be ignored, so they can be safely left in-place during a temporary downgrade if necessary.

Test Plan

System tests will test the rolling update flow:

- Begin with an `eager,compatible,` or `sessioned` cluster
- Create multiple connectors with multiple tasks
- Roll the cluster to `static` and confirm that data flow continues
- Add a static worker with a static connector assignment
- Add a static worker with a static task assignment
- Add a static worker with duplicate connector and task assignments
- Remove a wildcard worker
- Add remaining necessary static workers to cover all connectors and tasks
- Remove remaining wildcard workers

Rejected Alternatives

Model per-job resource constraints and measure resource utilization within a shared JVM

Measuring and enforcing resource constraints is not practical when multiple jobs share the JVM, as it is difficult to track which threads, memory, and I/O are used by each job. Without the ability to measure or enforce these constraints as hard limits, misconfigured resource limits would be difficult to track down as resource exhaustion on a single worker could not be easily attributed to a single job.

Spawn additional JVM processes attached to a single worker for individual connectors and tasks

This has the benefits of preserving the ability to assign multiple pieces of work to a single worker, while getting JVM isolation between connectors and tasks.

However, this separates the lifetime of the connectors and tasks, and leaves the possibility of a connector or task running while being completely out of sight of the rest of the cluster. Similar bugs within the current single-JVM model have been very harmful, and so the opportunity for such bugs should be avoided.

Replace the existing scheduling algorithm with a "weighted" algorithm using weights as a proxy for resource utilization.

This doesn't allow for an external system to correlate the resource utilization of a worker process to a job, without installing a contrived series of weights that force the algorithm to perform a static assignment. Abstract weights are also difficult to reason about, and may diverge from the resource constraints they are meant to model if there is no way to compare an abstract weight to a real utilization.

Implement a more complex Worker Selector/taint/affinity system like Kubernetes

It is possible to apply "labels" to a worker, and then have the connector/task "select" the labels which it requires it's hosting worker to have. Such a system would be extensible to arbitrary heterogeneity in clusters (plugin versions, resource availability, secrets, etc) without the need to use Kubernetes directly.

However, because tasks within a single connector can be heterogeneous, it is necessary to attach a different selector/affinity declaration to each task. But because tasks are dynamically created by the connector, either every task's selector must be added manually after defaulting to a wildcard affinity, or the Connect REST API would need a way to template multiple task affinities (parameterized by task-id). It felt more natural for a management layer to read the number of tasks that were dynamically created, and then start workers with corresponding static assignments.

Static Assignments and Worker Selectors have equal expressiveness when using Kubernetes, but Worker Selectors would be more expressive when running bare clusters. Worker selectors are more complex and bug-prone, and already implemented well in Kubernetes, so it would be more appropriate to encourage users to compose the tools rather than import a feature.

Accept static assignments via REST API instead of or in addition to specifying static assignments in the worker config

If only the REST API could set static assignments, this would require an additional configuration to distinguish static workers with empty static assignments from wildcard workers. This would allow clusters with only static workers to disable wildcard workers entirely. More importantly, persisting the assignments so that subsequent restarts get the same assignment requires persisting the worker identifier (id, hostname, etc) in the config topic, which will not be effective if the identifier changes between restarts. If the identifier regularly changes, a static worker will need to wait for a REST API call to install an assignment before being able to start work.

If both the worker config and REST API could set static assignments, this would cause the runtime configuration of the worker to diverge from the worker config. This divergence would then go away after a restart, (similar to the /admin/loggers API) which is not appropriate for a configuration that significantly changes the behavior of the worker.

Implement this as an alternative to Distributed mode which disables mutability of the worker via REST API

This represents a much larger investment, and has a much more difficult upgrade path for users of the Distributed deployment model, as it would require migrating connectors between Connect clusters. It also would require re-examining many of the abstractions used in Distributed mode, such as the config topic, connector config forwarding, zombie worker fencing, etc. Implementing an opt-in extension to Distributed mode which can force jobs to exclusively reside on certain nodes is much smaller incremental change, but still empowers users and external management layers to solve the resource isolation problems which are most painful.

Isolate ephemeral jobs in addition to persistent jobs

The Connect framework performs a number of ephemeral jobs using plugins, such as configuration validation and offsets resetting, version gathering, etc. These happen on whatever worker needs the information, most commonly the worker servicing a REST request or the leader of the cluster.

Because any worker can be expected to service any REST request, and any worker can become the leader, isolating these ephemeral jobs to specific nodes is not practical, and would require significant changes to the framework, such as the additional-process model or a change in request forwarding or leadership.