FLIP-10: Unify Checkpoints and Savepoints

Status

Discussion thread	
Vote thread	
JIRA	~~FLINK-4484~~ - FLIP-10: Unify Savepoints and Checkpoints CLOSED
Release	1.2

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

Motivation

Currently checkpoints and savepoints are handled in slightly different ways with respect to storing and restoring them. The main differences are that savepoints 1) are manually triggered, 2) persist checkpoint meta data, and 3) are not automatically discarded.

With this FLIP, I propose to allow to unify checkpoints and savepoints by allowing savepoints to be triggered automatically.

Proposed Changes

Persistent Checkpoints

The checkpoint coordinator has a fixed-size FIFO queue of completed checkpoints that are retained (current default size is 1). Checkpoints are discarded when they are removed from this queue. I propose to allow persisting these checkpoints like savepoints. This means that if a job fails permanently the user will have a checkpoint available to restore from. These can be subsumed like regular checkpoints..

As an example think of the following scenario: a job runs smoothly until it hits a bad record that it cannot handle. The current behaviour will be that the job will try to recover, but it will hit the bad record again and keep on failing. With the proposed change, some recent checkpoint is stored as a savepoint and the user can update the program to handle bad records and restore from the savepoint.

Periodic Savepoints

Allow the user to configure periodic triggering of savepoints. The behaviour should be similar to manually triggering savepoints periodically. Furthermore, we bound the number of available periodic savepoints. On shut down, they are never removed.

What's the Difference?

Although persistent checkpoints and periodic savepoints look similar, persistent checkpoints have a major difference: they can only be recovered with the same job and Flink version whereas savepoints allow to modify both the job and Flink version.

Miscellaneous

Always Create CheckpointCoordinator

Currently, the checkpoint coordinator is only created if checkpointing is enabled. This means that it is not possible to trigger a savepoint for stateful jobs that don't have periodic checkpointing enabled.

I propose to always create the CheckpointCoordinator, but only start periodic checkpoint and/or savepoint triggering if it is enabled. This way, users can trigger savepoints for jobs that don't have checkpointing enabled.

Deprecate Savepoint Backends

Savepoints should always go against files. Currently we allow `jobmanager` and `filesystem` backends and it's not possible to restore a `filesystem` savepoint if the `jobmanager` backend is configured. There is no good reason for this and we should get rid of this distinction (see deprecation plan below).

Public Interfaces

Configuration

Add `state.savepoints.dir` configuration key to specify the **default savepoint directory** and deprecate current configuration keys (see below).

Furthermore, there is the option to specify the directory **ad-hoc per savepoint** (via `CheckpointConfig` or the CLI, see below). This ad-hoc value has precedence over the default value.

Extend CheckpointConfig

```
// Stores completed checkpoints as savepoints
@PublicEvolving
void enablePersistentCheckpoints(String path);

@PublicEvolving
void disablePersistentCheckpoints();

// Enables periodic savepoints
@PublicEvolving
void enablePeriodicSavepoints(long interval, TimeUnit unit, String path):

@PublicEvolving
void disablePeriodicSavepoints();

// Sets the maximum number of retained periodic savepoints
@PublicEvolving
void setMaximumNumberOfRetainedPeriodicSavepoints(int max);
```

Furthermore we can extend the `StreamExecutionEnvironment` with shortcuts:

```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment();
env.enablePersistentCheckpoints(String path);
env.disablePersistentCheckpoints();
```

Add Optional Savepoint Path Argument to CLI

Currently, we require that a default path is configured and only use that one.

```
bin/flink savepoint <JobID> [<Path>]
```

Add REST API and Logging

Automatically created savepoints shall be logged at the JobManager and furthermore exposed via the REST APIs.

Add Savepoints REST API

Add `/jobs/:jobid/savepoints` to the REST API. This should list all currently available savepoints to restore from. This needs to be accessible after the job has entered a terminal state.

Furthermore, extend the `/jobs/:jobid/checkpoints` call to include a flag for each savepoint whether it is a checkpoint or savepoint.

Add Log Statements in CheckpointCoordinator

The `CheckpointCoordinator` shall log where savepoints have been created.

Compatibility, Deprecation, and Migration Plan

Compatibility

Compatability is not affected by these changes.

Deprecation

- Deprecate `savepoints.state.backend` and ignore any configured value. All savepoints go to files and `HeapSavepointStore` is only used for testing. With Flink 1.1, users can trigger savepoints out of the box without any configuration. In this case, savepoints are stored on the JobManager heap and the returned address has the format `jobmanager://savepoint-0`. By deprecating this, users will not be able to do this any more.
- Deprecate `savepoints.state.backend.fs.dir` and replace it by `state.savepoints.dir` (in line with the other default state backend configurations). If a value for `savepoints.state.backend.fs.dir` is configured, use that one. If both are configured, use `state.savepoints.dir`. This is used as the default directory for savepoints.

Migration Plan

The deprecated jobmanager "savepoint backend" does not affect migration from older Flink versions.

Test Plan

Describe in few sentences how the FLIP will be tested. We are mostly interested in system tests (since unit-tests are specific to implementation details). How will we know that the implementation works as expected? How will we know nothing broke?

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.