Engineering

The Job Executor: What Is Going on in My Process Engine?

By Thorben Lindhauer October 4, 2019

Sometimes we start a process, and after some initial processing, it does not continue for no apparent reason. Usually, this happens at **asynchronous continuations** or **BPMN timer events**, so points in the process where the *job executor*, the process engine’s component for deferred work, takes over.

While the job executor is a well-tested and battle-proven piece of software, it is not a closed system. It integrates with a relational database, an application server and the processes and applications it runs. Problems with job execution typically arise in this interaction of components. In this article, we look at common reasons why job execution can be delayed or stuck, how to diagnose, and how to resolve them.

Who Takes Care of Executing Processes?

Two types of threads execute processes in Camunda BPM:

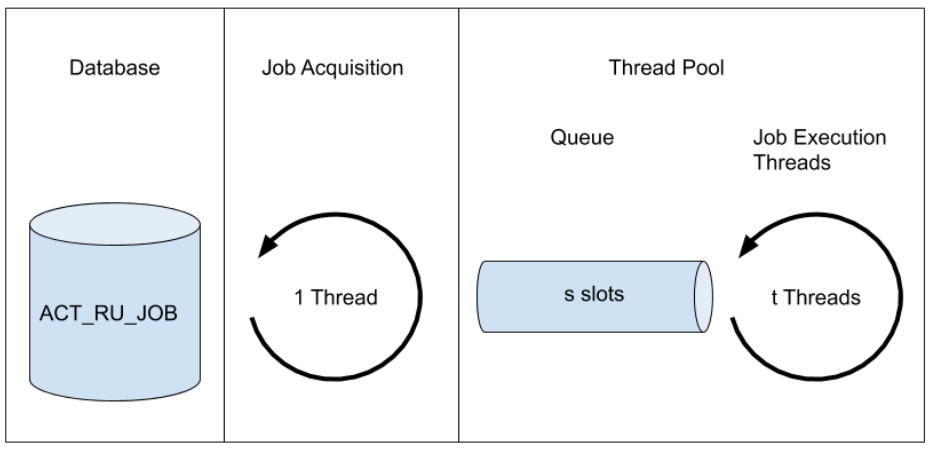
1. Application threads
2. Job Executor threads

Application threads are threads that call the Camunda Java API from an application context. For example, if we have a REST endpoint that starts a process instance, the HTTP worker thread calls `RuntimeService#startProcessInstanceByKey` and runs the process instance until **wait states** are reached. The calling thread receives immediate feedback: If the call fails, we can catch and handle an exception. If the call does not return, we can debug directly into our process’s delegate code and see where execution gets stuck.

The second type of threads is job execution threads. A job in the process engine is a piece of work that needs to be done at *some* time in the future. In the case of BPMN timer events, the BPMN model defines this point in time. In the case of asynchronous continuations, the time is as soon as possible. Job execution threads take care of this work and are managed by the job executor. Being rather detached from an application, it is sometimes hard to understand when job execution happens or, more importantly, when it does not happen or when it is slow.

The Job Executor

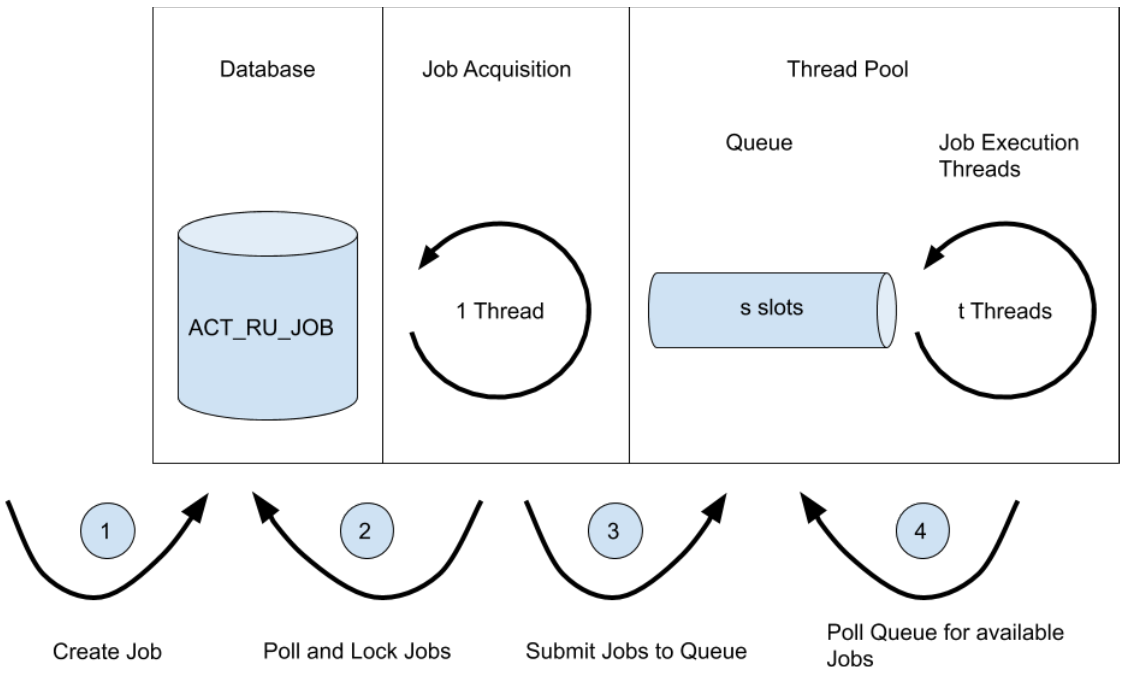
On the highest level, we can think of the job executor as a thread pool that picks and executes jobs whenever they become available. One level deeper, we have the following components:



1. The table `ACT_RU_JOB` in the Camunda database schema
2. A thread called *job acquisition*
3. A thread pool consisting of a work queue and collection of threads

From creation to completion, a job makes its way through each of these components:

easy!”



1. The job is inserted into `ACT_RU_JOB`, e.g., when a process instance reaches an asynchronous continuation.
2. Job acquisition picks up the job by polling `ACT_RU_JOB`. To ensure a job is only acquired once, acquisition sets a time-based lock in the database.
3. Job acquisition submits the job into the thread pool’s queue.
4. As soon as a job execution thread is free, it picks up the job from the queue and executes its business logic. Once done, it deletes the job from `ACT_RU_JOB`

The description of steps 3 and 4 is not precise in how thread pools work in Java, but it is fine for a conceptual understanding.

What Can Go Wrong and How Do I Find the Cause?

Jobs can get stuck or delayed in any stage of the pipeline. In the following, we look at common causes and the tools to diagnose each component.

Diagnosing `ACT_RU_JOB`

Problem:

- Jobs are acquired only with high delay or never

Identifying the problem:

- Inspect the table `ACT_RU_JOB`. If a job’s `LOCK_EXP_TIME_` and `LOCK_OWNER_` fields are empty, it has not been picked up by job acquisition.

Possible causes and solutions:

- The job is not yet available for execution.
    - Check the `DUE_DATE_` column to see if the job is scheduled for a future date (e.g., in case of a BPMN timer).
    - Check the `RETRIES_` column to see if the job has no retries left due to previous execution failures. Increase the retries via API or **Camunda Cockpit**.
    - Check the `SUSPENDED_` column to see if a job is suspended, in which case job acquisition ignores it (1 = job is active; 2 = job is suspended). Use the `ManagementService` API, REST API, or Cockpit to activate the job or process instance.
- Job execution is overloaded. If you produce jobs at a higher rate than the job executor can consume them, then jobs pile up. Consider adding more processing resources (e.g., by increasing the job execution thread pool or setting up a second job executor).

Diagnosing Job Acquisition

Problem:

- A job is available, but not acquired for execution or only with great delay.

Identifying the problem:

- Set the loggers `org.camunda.bpm.engine.impl.persistence.entity.JobEntity`, `org.camunda.bpm.engine.jobexecutor`, `org.camunda.bpm.engine.cmd` to `DEBUG`. This will produce output as follows:

```
...
14:44:15.011 [main] ENGINE-14014 Starting up the
JobExecutor[org.camunda.bpm.engine.impl.jobexecutor.RuntimeContainerJobExecuto
14:44:15.015 [Thread-6] ENGINE-14018
JobExecutor[org.camunda.bpm.engine.impl.jobexecutor.RuntimeContainerJobExecuto
starting to acquire jobs
14:44:15.016 [Thread-6] ENGINE-13005 Starting command --------------------
AcquireJobsCmd ----------------------
14:44:15.052 [Thread-6] ==>  Preparing: select RES.* from ACT_RU_JOB RES where
(RES.RETRIES_ > 0) and ( RES.DUEDATE_ is null or RES.DUEDATE_ <= ? ) and
(RES.LOCK_OWNER_ is null or RES.LOCK_EXP_TIME_ < ?) and RES.SUSPENSION_STATE_
and (RES.DEPLOYMENT_ID_ is null ) and ( ( RES.EXCLUSIVE_ = 1 and not exists(
select J2.* from ACT_RU_JOB J2 where J2.PROCESS_INSTANCE_ID_ =
RES.PROCESS_INSTANCE_ID_ -- from the same proc. inst. and (J2.EXCLUSIVE_ = 1)
also exclusive and (J2.LOCK_OWNER_ is not null and J2.LOCK_EXP_TIME_ >= ?) --
progress ) ) or RES.EXCLUSIVE_ = 0 ) LIMIT ? OFFSET ?
14:44:15.054 [Thread-6] ==> Parameters: 2019-10-02 14:44:15.017(Timestamp),
2019-10-02 14:44:15.017(Timestamp), 2019-10-02 14:44:15.017(Timestamp),
3(Integer), 0(Integer)
14:44:15.055 [Thread-6] <==      Total: 1
14:44:15.056 [Thread-6] ==>  Preparing: update ACT_RU_JOB SET REV_ = ?,
EXECUTION_ID_ = ?, LOCK_EXP_TIME_ = ?, LOCK_OWNER_ = ?, RETRIES_ = ?,
EXCEPTION_STACK_ID_ = ?, EXCEPTION_MSG_ = ?, DUEDATE_ = ?, SUSPENSION_STATE_ =
PROCESS_DEF_ID_ = ?, PROCESS_DEF_KEY_ = ?, JOB_DEF_ID_ = ?, DEPLOYMENT_ID_ = ?
HANDLER_CFG_ = ?, PRIORITY_ = ?, SEQUENCE_COUNTER_ = ? where ID_= ? and REV_ =
14:44:15.060 [Thread-6] ==> Parameters: 18(Integer), null, 2019-10-02
14:49:15.055(Timestamp), 7f0982bf-f720-430e-8bb5-ac95e01c542a(String),
3(Integer), null, null, 2019-10-02 00:01:00.0(Timestamp), 1(Integer), null, nu
null, null,
{"countEmptyRuns":0,"immediatelyDue":false,"minuteFrom":0,"minuteTo":59}(Strin
0(Long), 1(Long), 30f974ae-e511-11e9-a7b5-7e7a91cc35df(String), 17(Integer)
14:44:15.062 [Thread-6] ENGINE-13006 Finishing command --------------------
AcquireJobsCmd ----------------------
14:44:15.062 [Thread-6] ENGINE-14022 Acquired 1 jobs for process engine
'default': [[30f974ae-e511-11e9-a7b5-7e7a91cc35df]]
14:44:15.062 [Thread-6] ENGINE-14023 Execute jobs for process engine 'default'
[30f974ae-e511-11e9-a7b5-7e7a91cc35df]
14:44:15.063 [Thread-6] ENGINE-14011 Job acquisition thread sleeping for 4952
millis
14:44:15.122 [Thread-6] ENGINE-14012 Job acquisition thread woke up
14:44:15.122 [Thread-6] ENGINE-13005 Starting command --------------------
AcquireJobsCmd ----------------------
14:44:15.122 [Thread-6] ENGINE-13009 opening new command context
14:44:15.123 [Thread-6] ==>  Preparing: select RES.* from ACT_RU_JOB RES where
(RES.RETRIES_ > 0) and ( RES.DUEDATE_ is null or RES.DUEDATE_ <= ? ) and
(RES.LOCK_OWNER_ is null or RES.LOCK_EXP_TIME_ < ?) and RES.SUSPENSION_STATE_
and (RES.DEPLOYMENT_ID_ is null ) and ( ( RES.EXCLUSIVE_ = 1 and not exists(
select J2.* from ACT_RU_JOB J2 where J2.PROCESS_INSTANCE_ID_ =
RES.PROCESS_INSTANCE_ID_ -- from the same proc. inst. and (J2.EXCLUSIVE_ = 1)
also exclusive and (J2.LOCK_OWNER_ is not null and J2.LOCK_EXP_TIME_ >= ?) --
progress ) ) or RES.EXCLUSIVE_ = 0 ) LIMIT ? OFFSET ?
14:44:15.125 [Thread-6] ==> Parameters: 2019-10-02 14:44:15.122(Timestamp),
2019-10-02 14:44:15.122(Timestamp), 2019-10-02 14:44:15.122(Timestamp),
3(Integer), 0(Integer)
14:44:15.128 [Thread-6] <==      Total: 0
14:44:15.136 [Thread-6] ENGINE-13011 closing existing command context
14:44:15.136 [Thread-6] ENGINE-13006 Finishing command --------------------
AcquireJobsCmd ----------------------
14:44:15.137 [Thread-6] ENGINE-14022 Acquired 0 jobs for process engine
'default': []
...
```

Job acquisition is no longer a black box, but we can see how it works: When is `ACT_RU_JOB` polled, which SQL query is used, and which jobs are found and then submitted for execution.

Possible causes and solutions:

- The job acquisition query performs not well enough. For example, if we can query for one job in one second, our throughput is limited to 60 jobs per minute.
    - Improve query performance. Start with generating query plans and then analyze where time is spent.
    - Increase the number of jobs acquired with every query. Often the performance of the query does not depend on the number of jobs selected. Set the job executor configuration property **maxJobsPerAcquisition** to a higher value.
- The acquisition query's predicate `DEPLOYMENT_ID_ IN (...)` does not include the job's deployment id
    - If you do not use a shared process engine with process applications (e.g., you deploy processes only via REST API): Deactivate the process engine configuration property **jobExecutorDeploymentAware**.
    - Else: Provide the process as part of a process application. During deployment, the deployment will be registered with the job executor. Alternatively, register the deployment manually with the job executor via **ManagementService#registerDeploymentForJobExecutor**.
- Acquisition polls `ACT_RU_JOB` not fast enough
    - If the job table is empty, job acquisition applies exponential backoff to its polling. Use the job executor configuration properties **waitTimeInMillis** and **maxWait** to control the backing off.
    - Set up a second job executor to have two acquisition threads.
- Jobs are no longer picked up because the thread pool's queue is full and remains in that sate. This indicates that job execution threads are blocked (e.g., waiting for input they never receive, such as an HTTP response).
    - Take a JVM thread dump to verify this cause and find the blocking line of code. If the threads are blocked in your delegation code, try to rewrite the code such that unlimited blocking can never occur.

Diagnosing the Thread Pool

Problem:

- Too much time elapses between acquisition and execution of a job.

Identifying the problem:

- As above, set the loggers `org.camunda.bpm.engine.impl.persistence.entity.JobEntity`, `org.camunda.bpm.engine.jobexecutor`, `org.camunda.bpm.engine.cmd` to `DEBUG`. This will produce output as follows:

```
...
14:44:15.062 [Thread-6] ENGINE-13006 Finishing command --------------------
AcquireJobsCmd ----------------------
14:44:15.062 [Thread-6] ENGINE-14022 Acquired 1 jobs for process engine
'default': [[30f974ae-e511-11e9-a7b5-7e7a91cc35df]]
14:44:15.062 [Thread-6] ENGINE-14023 Execute jobs for process engine
'default': [30f974ae-e511-11e9-a7b5-7e7a91cc35df]
14:44:15.063 [Thread-6] ENGINE-14011 Job acquisition thread sleeping for
4952 millis
14:44:15.065 [pool-2-thread-1] ENGINE-13005 Starting command
-------------------- ExecuteJobsCmd ----------------------
14:44:15.066 [pool-2-thread-1] ENGINE-13009 opening new command context
14:44:15.066 [pool-2-thread-1] ==>  Preparing: select * from ACT_RU_JOB
where ID_ = ?
14:44:15.066 [pool-2-thread-1] ==> Parameters: 30f974ae-e511-11e9-
a7b5-7e7a91cc35df(String)
14:44:15.067 [pool-2-thread-1] <==      Total: 1
...
```

Compare the timestamps of the log entries to see how much time passes between acquisition and execution.

Possible causes and solutions:

- Execution threads are busy with other jobs
    - Increase the number of threads in the thread pool. Note: More threads can only help if the CPU (and other shared resources) is not already running at maximum capacity.

        - Decrease the time a job takes to run. For example, if a job makes an expensive computation, try to optimize your code. Use **external service tasks** instead, so that job execution thread pool does not perform the computation.
- The queue has grown too large
    - Decrease the size of the queue. Job acquisition does not acquire as many jobs anymore but waits for the queue to drain once it is full.

Diagnosing a Job During Execution

Problem:

- Job execution begins, but it never completes or takes a long time to complete.

Identifying the problem:

- If the job is stuck, take a JVM thread dump to understand where it is stuck.
- If the job is slow, activate debug logging for modules it uses to understand where time is spent. Set the logger `org.camunda.bpm.engine.impl.persistence` to `DEBUG` to see all SQL queries the process makes and how much time they take.

Working with Log Files

Some general hints when working with log files:

- Always include thread IDs in the log output. When reading the files, make sure not to mix up log output from different threads.
- When using any process engine logging, it always makes sense to set `org.camunda.bpm.engine.cmd` to `DEBUG`, so you can see where exactly a process engine command begins and ends.
- Learn how to configure logging in the environment in which you use Camunda. Some resources for common setups:
    - **Camunda documentation on logging**
    - **Configuring logging in Spring Boot projects**
    - **Configuring logging on Tomcat 9**. Be aware that Tomcat uses java.util.logging, where log levels have different names. `DEBUG` is `FINE`.
    - **Configuring logging on WildFly 17**

Happy Troubleshooting

If the job executor was a black box to you, we hope this blog post has raised the curtain and enables you to help yourself when you encounter problems. In any case, we are happy to help you on the **Camunda user forum**. Please share your log files with us and what you have tried.