# Coherence Scheduler

Table of Contents

[Coherence Scheduler 1](#_Toc404965423)

[1. Introduction 2](#_Toc404965424)

[2. Alternate Technologies/Approach and Limitations 2](#_Toc404965425)

[2.1 Quartz Framework 2](#_Toc404965426)

[2.2 Coherence processing pattern 2](#_Toc404965427)

[3. Architecture 3](#_Toc404965428)

[3.1 Description 4](#_Toc404965429)

[3.2 Pending tasks 6](#_Toc404965430)

## Introduction

Coherence Scheduler is a Job Scheduling timer service which exclusively runs on Oracle Coherence (12C) cluster. It supports job scheduling features like Quartz Framework, but on a distributed environment improving the fault tolerance and performance.

GitHub Link: <https://github.com/prabal77/Coherence-Scheduler.git/trunk/CoherenceScheduler>

## Alternate Technologies/Approach and Limitations

The architecture of Coherence Scheduler is highly inspired from that of Quartz Scheduler and Coherence Processing Pattern (Oracle Coherence Incubator Project). Few observations and limitation of existing technologies which led to the current design of Coherence Scheduler are as follows.

### Quartz Framework

It is probably the best open source scheduling framework available in market, but mostly suitable for standalone applications. And distributed version either requires a Database or Teracotta In-Memory Data Grid Cluster (Coherence’s Rival product - No Native support for Oracle Coherence).

Quartz stores all the Jobs, Calendars and Trigger related details in Local heap memory, and then a single thread reads all the data from the memory and passes it to Threadpool for execution. All jobs (Simple or Cron) are stored in the local JVM’s memory in different data structures (for default standalone installation) or in the case of distributed Quartz instance it is stored inside SQL tables of a dedicated DB or in terracotta enabled nodes. Direct integration of Oracle Coherence with Quartz scheduler has the following limitations.

#### 2.1.1Limitations

1. As all the triggers are stored local to the scheduler instance, jobs cannot be shared across cluster and this also creates Single point of failure.
2. Overriding JobStore Class to use Oracle Coherence as trigger storage rather than local JVM requires lots of custom serialization (Considering Cluster is using POF serialization by default and not the Native Java Serialization).
3. Jobs need to be directly submitted to Quartz scheduler instance for execution, which is not an optimum usage of cluster.

### Coherence processing pattern

The Oracle Coherence incubator project – Processing pattern, provides feature to execute jobs on a Coherence Cluster and return result once done. It provides optimized usage of cluster and job distribution can be configured i.e. it can be either done in round robin fashion or attribute based.

#### 2.2.1 Limitations

Doesn’t support scheduling Cron jobs, can execute only those jobs which are submitted real time by clients.

## Architecture

***Coherence - Cluster***

**Member Ship Store Cache**

Trigger Store Cache

**Submission Cache**

Resource Registry

Coherence Scheduler Service

Timer Service Thread

*Member Node1*

**Job Store Cache**

Resource Registry

Coherence Scheduler Service

Timer Service Thread

*Member Node - 2*

### Description

A new namespace handler is introduced named **SchedulerNamespaceHandler**, on node start this handler reads the **coherence-scheduler-cache-config.xml** and process all the elements with XML namespace “scheduler”. This namespace defines the following elements:

1. **scheduler:schedulerservice**

It defines the name, max thread count and other parameters which will be used to create the Coherence Scheduler instance on that particular node.

1. **scheduler:jobs**

This defines all the jobs which will be scheduled on cluster startup.

1. **scheduler:triggers**

This element defines all the triggers associated with the startup jobs. Triggers must be defined as a sub-child of <scheduler:job>.

Once a particular node is started SchedulerNamespaceHandler creates a custom service thread known as **TimerServiceThread** (which is an instance of a Daemon thread) and adds it to the Resource registry object (more on this later).

The SchedulerNameSpaceHandler also creates a new object **LifecycleInterceptor** which is an instance of Event Listener and is listening to the **Lifecycle Events** on that particular node. (Check Coherence Live Events).

**coherence-scheduler-cache-config.xml** also defines the following cache services which are used by Coherence Scheduler to perform the job scheduling tasks.

##### Job-Submission-Cache:

This is the only cache which is accessible to a Client application and is used to submit the Jobs and triggers to Coherence Scheduler. To submit jobs to this class, Client can create an instance on *JobDetail* and *Trigger* (must use builder class *SchedulerJobBuilder* and *SchedulerTriggerBuilder*) and invoke the submit() method of *CoherenceSchedulerFacade.*

**org.prabal.scheduler.listener.SubmissionsListener** of type Event Listener is listening to this cache and it asynchronously performs the initial processing of the submissions and puts the data to their respective caches i.e. Triggers to Trigger-Store-Cache and Jobs and Calendars to Job-Store-Cache.

**Note:** Startup jobs will also be submitted to the submission cache by the current node. So that the same job doesn’t get submitted/executed by multiple nodes (as cache config is same).

##### Job-Store-Cache:

It contains JobDetails objects, Calendar Objects and also hold list of all the active members for round robin job submission.

##### Trigger-Store-Cache:

This cache contains Trigger objects and will be polled constantly by TimerServiceThread to fetch the latest trigger to be executed.

##### Membership-store-cache

It is used to hold **TaskProcessorInstance** objects which are representation of a single storage enable Member Node in this cluster. These objects are used by Timer Thread to submit jobs to a target Coherence Scheduler service which is running on the node which is represented by the Object.

##### LifecycleInterceptor

This class is responsible for starting and stopping the CoherenceScheduler infrastructure on a particular node. During node start up, once all the services defined in the Cache Config file are started, it generates LifeCycle Event of type “Activated” which in turn is captured by the LifeCycleInterceptor object and it invokes **SchedulerInfraHelper.ensureInfrastructureStarted()**. Similarly during node shutdown it capture the “Disposing” event and calls SchedulerInfraHelper.cleanupResources().

##### SchedulerInfraHelper

This helper class performs the following functions to create and start the CoherenceScheduler Infrastructure.

* Creates all the required caches (mentioned above).
* Create the payload queue and provides it to the JobExecutionShell.
* Creates an instance the JobExecutionShell (the actual scheduler class) and starts the same.
* Registers the queue and execution shell to resource registry.
* Creates an object of type TaskProcessorInstance and adds it to the MembershipStore Cache.
* Also creates and MemberShip Listener to listen for member leaving event and assigns to the cache service.
* Starts the TimerService Thread.

Similarly cleanUpResources will shut down the Executor services running on that node and un-register all the resources added to the resource registry.

##### Job Execution Shell

This is the actual class which performs the execution of the jobs. It creates Threadpool of fixed size (configurable value) and also a payload queue which is an instance of a BlockingQueue of the same size as of threadpool.

Once started the execution shell keeps on listening to the Blockingqueue for JobExecutionPayload Objects and executing them, if no object is present it gets blocked.

##### Timer Service Thread

This a custom service which runs on every node and is responsible for fetching triggers from the trigger store cache, fetching the next eligible TaskProcessorInstance object (target node where job needs to be scheduled) and submitting the job to its payload queue.

This thread will keep on poling the trigger store cache at regular interval (configurable) and should only stop with the node shutdown.

*TimeServiceThread.run()* calls *CoherenceJobStore.acquireNextTrigger()* and pass it a data range to fetch the trigger list satisfying the same. CoherenceJobStore returns a list of triggers sorted by the NextFireTime value. These triggers are pushed to the payload queue of the next eligible TaskProcessorInstance (with *TaskProcessorKey.nextAvailableProcessor* as true). After pushing the data to queue, it also updates the trigger status to “TRIGGERED”.

Each Time Thread is responsible for the triggers which are present in the Trigger-Store-Cache partition owned by the node in which it is running.

### 3.2 Pending tasks

1. Have to complete the CoherenceJobStore class. Fetching triggers and applying the calendar object is not proper.
2. Need improve efficiency of the code and have to reduce the number of calls to the cache services.
3. Have to provide delay start feature to the Timer Thread and the schedulers.