19 Parallel running of ensemble loops and activities on one forecasting shell instance

Function:	runInLoopParallelProcessorCount, set the amount of cores available to Delft-FEWS when running workflow portions in parallel in one forecasting shell instance

Module Name:	runInLoopParallelProcessorCount
Where to Use?	global properties file
Why to Use?	to speed-up ensemble runs on multi core machines
Description:	The runInLoopParallelProcessorCount entry in the global properties files indicated the number of cores Delft-FEWS may use when running ensemble members in a loop
Preconditions:	2009-02 release, multi core cpu or multi cpu computer
Outcome(s):	speed-up of the computations
Scheendump (s):	*link to attached screendump(s) for displays only*
Remark(s):	The speedup that may be obtained is highly dependent on the type of module you are running
Available since:	DelftFEWS200902

Contents

- Contents
- Overview
- Configuration
- Tested modules
- Sample input and output (with explanation of all options)
- Combination of ensembles and normal time series
- Error and warning messages
- Known issues
- Related modules and documentation
- Technical reference

Overview

Delft-FEWS can, within one Forecasting Shell instance, split ensemble workflows (that have the runInLoop element set to true) over multiple cores. Based on the available amount of cores a number of queues is made, one for each core. When running the activity the different ensemble members are added to the different queues. Since FEWS 2017.01 it is optionally possible to run parallel activities on multiple forecasting shells. This can also apply for ensemble workflows as is explained in section 'parallel' on page 06 Configuring WorkFlows.

To split an ensemble activity over multiple cores within one Forecasting Shell instance, use runInLoop=true in the workflow file (see example below) in combination with the global property definition as explained below the xmlblock:

```
	<activity>
		<runIndependent>true</runIndependent>
		<moduleInstanceId>MOGREPS_Spatial_Interpolation</moduleInstanceId>
		<ensemble>
			<ensembleId>MOGREPS</ensembleId>
			<runInLoop>true</runInLoop>
		</ensemble>
	</activity>
```

By default the General Adapter runs are performed in the %TEMP_DIR% directory. The %TEMP_DIR% variable is an internal variable which points to a unique temporary directory which is created in the $REGION_HOME$/Temp and which will be removed afterwards.

Configuration

By default Delft-FEWS will only use one core and all tasks are run one after another. To enable the parallel running of ensemble members the runInLoopParallelProcessorCount entry must be set in the global properties file. Here you either specify the number of cores to use or specify 100 to use all available cores.

In the global properties

Config Example	`# to use all available cores/cpu's:` `runInLoopParallelProcessorCount=100`

For all internal Delft-FEWS modules that have been tested no changes are needed to the configuration. For external modules that are run using the General adapter some changes may be needed to the configuration.

Tested modules

Module	Remarks
Transformation (old)	Test ok
Interpolation	Test ok. Interpolation via DLL not tested
TransformationModule (new)	Test ok. Interpolation via DLL not tested
pcrTransformation	Test ok
General Adapter	test ok

Sample input and output (with explanation of all options)

An example for parallel running an import and then processing of the import modules is:

```
<parallel>
  <activity>
    <runIndependent>true</runIndependent>
    <moduleInstanceId>Import1</moduleInstanceId>
  </activity>
  <activity>
    <runIndependent>true</runIndependent>
    <moduleInstanceId>Import2</moduleInstanceId>
  </activity>
  <activity>
    <runIndependent>true</runIndependent>
    <moduleInstanceId>Import3</moduleInstanceId>
  </activity>
</parallel>
<activity>
  <runIndependent>true</runIndependent>
  <moduleInstanceId>CombineImports</moduleInstanceId>
</activity>
<activity>
  <runIndependent>true</runIndependent>
  <moduleInstanceId>SpatialInterpolation</moduleInstanceId>
</activity>
```

In this example the imports are run in parallel (as many as the runInLoopParallelProcessorCount is defined). Once they are all ready, the postprocessing modules are performed.

Another example for running a SobekRE model requires more advanced configuration. It is possible to run in parallel the sobek models, but the preprocessor of the adapter is not thread safe. So we need to ensure that the preprocessor (that converts the PI-timeseries into model) never runs in parallel.

The workflow shows:

```
<activity>
  <runIndependent>false</runIndependent>
  <moduleInstanceId>Sobek_Prep</moduleInstanceId>
  <ensemble>
    <ensembleId>EPS</ensembleId>
    <runInLoop>true</runInLoop>
  </ensemble>
</activity>
<activity>
  <runIndependent>false</runIndependent>
  <moduleInstanceId>Sobek_GA</moduleInstanceId>
  <ensemble>
    <ensembleId>EPS</ensembleId>
    <runInLoop>true</runInLoop>
  </ensemble>
</activity>
<activity>
  <runIndependent>true</runIndependent>
  <moduleInstanceId>Sobek_Post</moduleInstanceId>
</activity>
```

The workflow first runs the Sobek_Prep module for each ensemble member individually. You can also run the Sobek_prep in one go (like the Sobek_Post). In that case the whole ensemble is looped in the transformation module itselves but that requires more RAM and does not use multiple threading.
The next module Sobek_GA runs the sobek model for each individual ensemble member. See the details for sobek below. The last module Sobek_Post is run in one go. Here defined as an example, but probably it is more efficient to run this activity also in parallel like the Sobek_Prep for performance reasons.

The General Adapter configuration of the Sobek_GA should look like:

```
<general>
  <rootDir>%TEMP_DIR%</rootDir>
  <workDir>%ROOT_DIR%/work</workDir>
  ...
</general>
<activities>
  <exportActivities>

    <exportDataSetActivity>
      <moduleInstanceId>Sobek_GA</moduleInstanceId>
    </exportDataSetActivity>
  </exportActivities>

  <executeActivities>
    <executeActivity>
      <command>
        <className>nl.wldelft.fews.adapter.sobek.PreSobekModelAdapter</className>
      </command>
      <arguments>
        <argument>%ROOT_DIR%</argument>
        <argument>Config/sobekConfig.xml</argument>
      </arguments>
      <timeOut>60000</timeOut>
      <waitForOtherRun>true</waitForOtherRun>
      <overrulingDiagnosticFile>%ROOT_DIR%/diagnostics/presobekmodeladapter.xml</overrulingDiagnosticFile>
    </executeActivity>
    <executeActivity>
      <command>
        <executable>%ROOT_DIR%/bin/sobeksim.exe</executable>
      </command>
      <arguments>
        <argument>%ROOT_DIR%/bin/sobeksim.fnm</argument>
      </arguments>
      <timeOut>600000</timeOut>
      <ignoreDiagnostics>true</ignoreDiagnostics>
    </executeActivity>
    <executeActivity>
      <command>
        <className>nl.wldelft.fews.adapter.sobek.PostSobekModelAdapter</className>
      </command>
      <arguments>
        <argument>%ROOT_DIR%</argument>
        <argument>Config/sobekConfig.xml</argument>
      </arguments>
      <timeOut>60000</timeOut>
      <overrulingDiagnosticFile>%ROOT_DIR%/diagnostics/postsobekmodeladapter.xml</overrulingDiagnosticFile>
    </executeActivity>
  </executeActivities>
  <importActivities>
    ...

  </importActivities>
</activities>
```

Notice that it is ensured that the preprocessor does **not** run in parallel with another ensemblemember by defining the element waitForOtherRun=true.

In this example the next global.properties should be defined:

```
runInLoopParallelProcessorCount=2
```

This makes that maximum 2 threads are used and the Sobek runs are performed in temporary directories that are created in regionhome/temp/session. These directories are removed after running. The directories are filled with data that you should export from a moduledataset. Notice that purgeActivities will not work therefore. When running in debug mode the temp is deleted when closing FEWS instead when the run is finished

In a test using 2 CPU the total run of the whole ensemble reduced from exactly 900 seconds to 546 seconds.

Internal variables

You can use in any filename or directory the properties from the global.properties file or the next internal variables:

- TEMP_DIR
- ROOT_DIR
- WORK_DIR
- ENSEMBLE_MEMBER_ID
- ENSEMBLE_MEMBER_INDEX
- TIME0
- TASK_ID
- TASK_RUN_ID
- TASK_RUN_ID_FOR_PATH
- TASK_DESCRIPTION
- TASK_USER_ID
- TIME_ZONE_OFFSET_SECONDS
- MC_ID (from 2018.02 onwards)
- FSS_ID (from 2018.02 onwards)

The colon characters ":" will be replaced by an underscore "_". Use the internal variables with % characters (like %TEMP_DIR%) and the global.properties variables with $ characters (like e.g. $DUMP_DIR$).

Combination of ensembles and normal time series

You may have a combination of for example a rainfall ensemble with other non-ensemble timeseries (like structure operation or boundary levels). In case you run an ensemble with the runInLoop option at workflow level by default all timeseries are used from the defined ensembleId and member, also for a non-ensemble timeseries. That means that you should configure transformations to create also an ensemble of your non-ensemble series, which is of course not convenient. To enable non-ensemble timeseries FEWS has an option that overrules the ensemble member forcing at workflow level. Therefore you define for the non-ensemble series the fixed ensembleId "main". This is a "virtual" ensemble.

A configuration example:

```
<errorModelSet>
  <inputVariable variableId="observation">
    <timeSeriesSet>
      <moduleInstanceId>Import</moduleInstanceId>
      <valueType>scalar</valueType>
      <parameterId>Q.obs</parameterId>
      <locationId>NA_Mastenbroek</locationId>
      <timeSeriesType>external historical</timeSeriesType>
      <timeStep unit="hour"/>
      <relativeViewPeriod unit="hour" start="-96" end="0" startOverrulable="true" endOverrulable="false"/>
      <readWriteMode>read only</readWriteMode>
      <ensembleId>main</ensembleId>
    </timeSeriesSet>
  </inputVariable>
  <inputVariable variableId="update_run">
    <timeSeriesSet>
      <moduleInstanceId>Sobek_Update</moduleInstanceId>
      <valueType>scalar</valueType>
      <parameterId>Q.sim.hist</parameterId>
      <locationId>NA_Mastenbroek</locationId>
      <timeSeriesType>simulated historical</timeSeriesType>
      <timeStep unit="hour"/>
      <relativeViewPeriod unit="hour" start="-96" end="0" startOverrulable="true" endOverrulable="false"/>
      <readWriteMode>read only</readWriteMode>
      <ensembleId>main</ensembleId>
    </timeSeriesSet>
  </inputVariable>
  <inputVariable variableId="forecast_run">
    <timeSeriesSet>
      <moduleInstanceId>Sobek_Forecast</moduleInstanceId>
      <valueType>scalar</valueType>
      <parameterId>Q.sim.for</parameterId>
      <locationId>NA_Mastenbroek</locationId>
      <timeSeriesType>simulated forecasting</timeSeriesType>
      <timeStep unit="hour"/>
      <relativeViewPeriod unit="hour" start="-96" end="120" startOverrulable="true" endOverrulable="true"/>
      <readWriteMode>read only</readWriteMode>
      <ensembleId>EPS</ensembleId>
    </timeSeriesSet>
  </inputVariable>
  <autoOrderMethod>
    <orderSelection>true</orderSelection>
```

```
    <order_ar>3</order_ar>
    <order_ma>1</order_ma>
    <subtractMean>true</subtractMean>
    <boxcoxTransformation>false</boxcoxTransformation>
    <lambda>0</lambda>
    <observedTimeSeriesId>meting</observedTimeSeriesId>
    <simulatedTimeSeriesId>forecast_run</simulatedTimeSeriesId>
    <simulatedTimeSeriesId>update_run</simulatedTimeSeriesId>
    <outputTimeSeriesId>corrected</outputTimeSeriesId>
  </autoOrderMethod>
  <interpolationOptions>
    <interpolationType>linear</interpolationType>
    <gapLength>6</gapLength>
  </interpolationOptions>
  <minResult>0</minResult>
  <ignoreDoubtful>true</ignoreDoubtful>
  <outputVariable variableId="corrected">
    <timeSeriesSet>
      <moduleInstanceId>Sobek_Forecast</moduleInstanceId>
      <valueType>scalar</valueType>
      <parameterId>Q.updated.for</parameterId>
      <locationId>NA_Mastenbroek</locationId>
      <timeSeriesType>simulated forecasting</timeSeriesType>
      <timeStep unit="hour"/>
      <relativeViewPeriod unit="hour" start="-96" end="120" startOverrulable="true" endOverrulable="true"/>
      <readWriteMode>add originals</readWriteMode>
      <ensembleId>EPS</ensembleId>
    </timeSeriesSet>
  </outputVariable>
</errorModelSet>
```

Error and warning messages

Description of errors and warnings that may be generated

Error:	Error message
Action:	Action to fix

Known issues

- Running modules in parallel means you will use more memory

In some cases, the increase in speed may be very limited. Although it depends on a case by case basis the following simple rules may be used to determine the experted increase in execution speed:

- execution time of an individual module <= 1 sec: expected increase < 20%
- execution time of an individual module > 1 sec < 10: expected increase between 20 and 50%
- execution time of an individual module > 10 sec: expected increase > 50 % and < 100 %

The percentage given in the list above should be scaled using the amount of cores used. The 100% in the example above is a two-fold increase using two cores.

Other factors that influence this are the amount of data being retrieved and store in the FEWS database in relation to the total execution time and (in the case of an external module) the amount of data written to and read from the file system.

Related modules and documentation

Links to related parts of the system

Technical reference

Entry in moduleDescriptors:	none
Link to schema:	none