Prototype of Esper-based DAQ expert system

Tomasz Bawej

September 30, 2014

Contents

1 Overview **2**
1.1 Goal . . . 2
1.2 Data sources . . . 2
1.3 Data input and processing . . . 2
1.4 Output . . . 2

2 Details **2**
2.1 Settings . . . 2
2.1.1 Proxy parameters . . . 2
2.1.2 Flashlist database settings . . . 3
2.1.3 Online flashllists . . . 3
2.1.4 Dumped flashlists . . . 3
2.1.5 EPL directories . . . 3
2.1.6 Output modules . . . 3
2.1.7 Events playback range . . . 3

3 Use cases **4**
3.1 Offline analysis . . . 4
3.2 Online analysis . . . 4
3.3 Dumping data for offline analyses . . . 4
3.3.1 Dumping online data . . . 4
3.3.2 Populating database . . . 5
3.4 Dumping online data . . . 5

4 Configuration requirements **6**
4.1 Hardware Configuration Database . . . 6
4.2 proxy . . . 6
4.3 Events Database . . . 6
4.4 Overview . . . 6
4.5 Details . . . 6

5 Overview of implemented checks **6**

1 Overview

1.1 Goal

Level **0** **A**nomaly **D**etective, hereinafter commonly abbreviated as L0AD, has been implemented in an effort to evaluate Esper as a potential backbone of the CMS DAQ expert system. The project consists of a collection of EPL scripts for processing events as well as components responsible for definition and retrieval of events, registration of EPL statements and displaying the outcome of analyses. The project has never reached a mature state, hence parts of it are not well structured.

1.2 Data sources

The program receives input data in the form of events. While Esper allows several possible event representations, logically a single event is a collection of labelled data, a map. For an event to be fed into the Esper engine it has to be delivered in a predefined stream. Streams define the structure of events they can carry so that all events entering a particular stream would contain fields of the same names and types.

In L0AD, the definition and continuous population of event streams is handled by `EventsTap` objects. The project contains two implementations of taps: one for retrieving the events from a database and another for fetching them as they are published on the web. More implementations could be added by subclassing the `EventsTap` or any of its descendants.

1.3 Data input and processing

Once the event streams are defined, the engine can be instructed to detect occurrences of specific circumstances. Statement objects represent instructions for the engine and Esper provides a bidirectional conversion mechanism between a statement object and its textual EPL representation. Thus, all the queries have been saved in epl files under the `epl` directory. At runtime these files are deployed as modules by a `FileBasedEplProvider` object that relies on Esper's capabilities in that matter.

1.4 Output

The project introduces an `EventsSink` abstract class to handle outputting the information from the Esper event processing engine. Provided implementations include `FileSink`, `SwingGui` and `ThroughputMonitor` classes, where `ThroughputMonitor` only outputs the performance-related information. It is up to the `EventsSink` implementations to provide instances of `com.espertech.esper.client.StatementAwareUpdateListener` object for statements annotated with @Verbose and @Watched annotations. The object provided will be registered as a listener to the statement, enabling it to receive relevant updates.

2 Details

The following section is meant to fill in the big picture drawn above.

2.1 Settings

`load.properties` file explained:

2.1.1 Proxy parameters

```
socksProxyHost=127.0.0.1 proxySet=true socksProxyPort=1080
```

2.1.2 Flashlist database settings

flashlistDbMode=read flashlistDbType=mysql flashlistDbHost=localhost flashlistDbUser=load
flashlistDbName=flashlists_rest flashlistDbEngine=myisam retrievalTimestampName=fetchstamp
flashlistDbIndexTimestamps=true

2.1.3 Online flashllists

onlineFlashlistsRoot[0]=http://srv-c2d04-19.cms:9941/urn:xdaq-application:lid=400/
onlineFlashlistsRoot[1]=http://srv-c2d04-19.cms:9942/urn:xdaq-application:lid=400/

2.1.4 Dumped flashlists

flashlistForDbDir[0]=/depot/flashlists13.11/ flashlistForDbDir[1]=/depot/flashlists13.11_2/
flashlistForDbDir[2]=/depot/flashlists13.11_3/ flashlistForDbDir[0]=/depot/flashlists13.11_4/
flashlistForDbDir[0]=/home/bawey/Desktop/flmini/flashlistsExport flashlistForDbDir[2]=/home/

2.1.5 EPL directories

eplDir[0]=epl

2.1.6 Output modules

view[0]=ThroughputMonitor view[1]=FileSink view[2]=SwingGui

2.1.7 Events playback range

timerStart=1383763860000 timerEnd=1383763860800
dateFormat=yyyy-MM-dd'T'HH:mm:ss.SSS
outputDir=/home/bawey/load/
flashlists=levelZeroFM_subsys;EVM;frlcontrollerLink;frlcontrollerCard;levelZeroFM_static;gt_cell_lumiseg;
blacklist_jobcontrol=jobTable blacklist_frlcontrollerCard=myrinetProcFile,myrinetBadEventNumber,myrin
blacklist_levelZeroFM_subsys=FEDS blacklist_StorageManagerPerformance=activeEPs;averagingTime;bandwi
logSinks="ch.cern.cms.load.sinks.SwingGui";
Connecting to databas
Dumping data The very main method of the program allows switching into dumping mode:

```
public static final void main(String[] args) {
   Thread.currentThread().setName("Level 0 Anomaly Detective");
   instance = getInstance();
   if (instance.settings.getProperty(DataBaseFlashlistEventsTap.KEY\
      _DB\_MODE, "read").equalsIgnoreCase("write")) {
     MysqlDumper.main(args);
   } else {
     instance.defaultSetup();
   }
  }
```

Which in fact was an ugly way to merge the original dumper into the project. Anyway, it might still work.

HwInfo dumping During development it was possible to dump the HwInfo database into file and load it afterwards to speed the program launch up. It however required several modifications to the framework (making all the classes involved in dumping implement Serializable) that have not been introduced into it. Hence the HwInfo class still examines the settings for options specifying the HwInfo dump file to write to / read from, yet without providing these it will simply use the remote DB. EPL annotations As mentioned above, L0AD introduces two annotations

that can be used in EPL statements: @Verbose @Watched The options they take is described in the source code.

EPL statements (Please note: for the statements to be effective, the events have to be defined)

Event definitions Event definitions are dynamically fed into the program every time it starts up, based on the

Data types resolving The FieldTypeResolver class stores information about the types that particular field should be converted to. This can be specified both globally, for all fields of given name, or locally for particular event type. Unfortunately, these rules are currently hard-coded into the class. Implemented checks

3 Use cases

3.1 Offline analysis

In order to perform offline analysis, program has to be configured to use the *Events Database*[4.3] and a *Hardware Configuration Database* connection [4.1] needs to be set up.

```
flashlistDbMode=read
flashlistDbType=mysql
flashlistDbHost=localhost
flashlistDbUser=load
flashlistDbName=flashlists_rest
```

3.2 Online analysis

Online analysis requires a slightly different approach: *Events Database* should be disabled in favor of of providing `onlineFlashlistsRoot` entries pointing to a network location to retrieve the flashlists from. Connection to the *Hardware Configuration Database*[4.1] remains a requirement, while a SOCKS proxy[4.2] configuration is also needed.

Listing 1: Sample options for fetching flashlists over the network

```
onlineFlashlistsRoot[0]=http://srv-c2d04-19.cms:9941/urn:xdaq-
    application:lid=400/
onlineFlashlistsRoot[1]=http://srv-c2d04-19.cms:9942/urn:xdaq-
    application:lid=400/
```

3.3 Dumping data for offline analyses

3.3.1 Dumping online data

Originaly the data was saved to and played back from files only. Thus, no mechanism has been implemented to dump the data directly into a database. Instead, provided that flashlists location is supplied and reachable, their contents are dumped into the folder specified by configuration option `THAT_IS_CURRENTLY_LOST_ALONG_WITH_THE_CODE_THAT_USES_IT`. Within that directory a subdirectory is creatred for each flashlist type. Flashlist rows are dumped into the files named 0, 1, 2... and so on - unique rows only, switching to the next file once the current one reaches the size of 256MB. Dumping also involves adding information about the time of fetching the flashlist - the `fetchstamp` column.

Listing 2: Sample directory structure of dumped flashlists data

```
.
+-- urn:xdaq-flashlist:BU
|     +-- 0
```

```
|      +-- 1
|      +-- 2
|      +-- 3
|      +-- 4
|      +-- 5
|         -- 6
+-- urn:xdaq-flashlist:diskInfo
|      +-- 0
|      +-- 1
|      +-- 10
|      +-- 2
|      +-- 3
|      +-- 4
|      +-- 5
|      +-- 6
|      +-- 7
|      +-- 8
|         -- 9
(...)
+-- urn:xdaq-flashlist:StorageManagerPerformance
       -- 0
```

3.3.2 Populating database

Switching `flashlistDbMode` parameter to `write` makes the `Load` class `main` method invoke the `main` method of the dumper to pass control into what used to be a separate application. Apart from

```
flashlistDbMode=write
flashlistDbType=mysql
flashlistDbHost=myDbHost
flashlistDbUser=myDbUser
flashlistDbName=myDbName
flashlistDbPass=myDbPass


flashlistForDbDir[0]=/depot/flashlists13.11/
flashlistForDbDir[1]=/depot/flashlists13.11_2/
flashlistForDbDir[2]=/depot/flashlists13.11_3/
flashlistForDbDir[0]=/depot/flashlists13.11_4/
```

The configuration above assumes each of *root* flashlist directories contains a set of subdirectories named after the flashlist they hold the values from. Listing below shows an example of such directory structure.

- Event DB connection for writing
- Flashlists dumped on the disk

3.4 Dumping online data

As mentioned above, the flashlists were firs dumped into files and only afterwards into a databases.

- Online flashlists connection

4 Configuration requirements

4.1 Hardware Configuration Database

Hardware Configuration DB connection is required for the `HwInfo` class to work. `HwInfo` provides helper methods that can be used in EPL statements to retrieve information about the hardware. In development environment the connection was setup using a port-forwarding option in `/.ssh/config`: `LocalForward 10121 cmsrac11-v:10121`, a hard-coded DB url: `jdbc:oracle:thin:@localhost:10121/cms_omds_tunnel.cern.ch` and an active ssh connection with *cmsusr*. Building the project within *Eclipse* requires the *framework* project as dependency (trunk version as of 26.06.2014).

4.2 proxy

Fetching flashlists as they are published was achieved via configuring ssh (`DynamicForward 1080` option), specifying the address to fetch the flashlists from and settings for SOCKS Proxy:

```
socksProxyHost=127.0.0.1
proxySet=true
socksProxyPort=1080
```

4.3 Events Database

The Events Database was setup during development to facilitate events playback, especially any partial playback involving arbitrary start and end timestamp. Database structure automatically mimicked the structure of published flashlists, ie. a table was created for each flashlist type and a column for each flashlist field. For convenience, all fields were stored as `VARCHAR` and one additional numeric column was added to each table: `fetchstamp` indicating the timestamp of fetching the flashlist. An extra table `fetchstamps` stores all the unique `fetchstamp` values for all other tables. Also, tables storing useful events (ie. used for analyses) have been indexed by `fetchstamp` column. This minimizes "query misses" - attempts to retrieve data for a `fetchstamp` with no corresponding events.

Password can be specified using the `flashlistDbPass` parameter. *MySQL* is the only fully supported database type with some rudimentary *MongoDB* also exists, but needs to be worked on to be useful.

4.4 Overview

4.5 Details

5 Overview of implemented checks

Purpose of the check	Solution notes / related EPL files
During an ongoing run.	
Message on Run Start giving: Run NR, SID, Detectors in, Feds in per detector	Subsystems in uses a custom aggretaion method that turns out to be running all the time (and not on demand)
	`runStartStop.epl`

Message on Run Stop giving: Run Nr, SID, Detectors in, (*)Feds in per detector, avg L1 rate, avg stream A rate, avg dead time (from trigger LAS), duration	Same as for run start, but also: get avg L1 rate, avg stream A rate, avg dead time from trigger LAS, the duration
	`runStartStop.epl`
Message on L1 trigger rate jump of 10% or more	EPL file creates some windows and streams useful for performing other loigic. A javascript method needs to be raplaced with a case statement
	`level1TriggerRate.epl`
Message on subsys going to any of: ERROR, RUNNING_DEGRADED, RUNNING_SOFT_ERROR_DETECTED, PAUSING, PAUSED, RESUMING	
	`subsystemsStateChanges.epl`
Message on any state change of DAQ	
	subsystemsStateChanges.epl
Message on jobcontrol flashlist not being updated after 1 minute	
	jobcontrolNotUpdated.epl
FED dead-time > 1%	Investigates backpressure on the same FED or its main FED.
	deadtimeAndBackpressure.epl
Backpressure > 1%	
	deadtimeAndBackpressure.epl
Message on stream A > 500 Hz	Simplified, does not satisfy: after 10 seconds, repeat message every 10 seconds. Need to sum the last-per-context values
	streamARate.epl

FEDs fraction other	A FedFractions stream is constantly filled with derivatives computed for each subsequently arriving pair of FmmInput events with the same fedId. Fractions (busy+warning +error+ready+oos) must add up to one, otherwise FED spends some time in an illegal (other) state. This fact should be reported and using integrals is preffered for calculations to avoid floating point numbers comparison. Why does it report negative fractions at the beginning? (As well as negative dTime). Never tested on positives.
	fedFractions.epl
Rate 0 for $>$ 10 seconds	
Check Bx alignment and print message if not aligned repeat after 10 seconds	Fails to repeat the message every 10 seconds if the problem persists
	rateZeroTests.epl
Check triggers(events) alignment + print message if not aligned repeat after 10 seconds	Fails to repeat the message every 10 seconds if the problem persists
	rateZeroTests.epl
FEDs stuck in ERROR/OOS/WARNING/BUSY(anything not READY)	Written, running, never seen working yet. Using the FedFractions stream to find events of interest and SuspendedStatements window to suspend/resume the statement upon reception of RateStuckAtZeroEvent or RateFineEvent defined in the same source file as a way to broadcast the message that the expert enters a state where the run is ongoing but the rate has been 0 for long enough or that the rate is fine and experiment is running. ERROR/OOS/WARNING/BUSY have their corresponding fractions on FMMInput. Stuck means fraction = 1. A FED might also be stuck in an other state. Use the integrals from two consecutive events to determine that.
	rateZeroTests.epl
List FEDs with backpressure or deadtime	

	rateZeroTests.epl
Check if number of resyncs and the last resync event number is the same in all FEDs?	Check if the number of resyncs is the same and check if the last event (the resync was seen for) is the same. myrinetResync - number of resync events, myrinetLastResyncEvt - last event that the resync was seen for.
	rateZeroTests.epl

A 'mock' version (with additional comments) of the table below bac be found under the `epl/xml` directory.