Spf4j overview

1. Performance monitoring

1.1. How to record measurements?

a) Via API

Lowest impact:

```
private static final MeasurementRecorder recorder = RecorderFactory.createScalableQuantizedRecorder(forWhat,
        unitOfMeasurement, sampleTime, factor, lowerMagnitude, higherMagnitude, quantasPerMagnitude);
…
recorder.record(measurement);
```

More flexible in case you want to decide on what to measure at runtime:

```
private static final MeasurementRecorderSource recorderSource  =
        RecorderFactory.createScalableQuantizedRecorderSource( forWhat,
        unitOfMeasurement, sampleTime, factor, lowerMagnitude, higherMagnitude, quantasPerMagnitude);
        …
recorderSource.getRecorder(forWhat).record(measurement)
```

b) Via Aspect

Annotate a method you want to measure and monitor performance with the annotation:

```
@PerformanceMonitor(warnThresholdMillis=1, errorThresholdMillis=100, recorderSource = RecorderSourceInstance.Rs5m.class)
```

Start your jvm with aspectj load time weaver:

```
-javaagent:${project.build.directory}/lib/aspectjweaver-${aspectj.version}.jar
```

Make sure your acpectj config file (META-INF/aop.xml) contains:

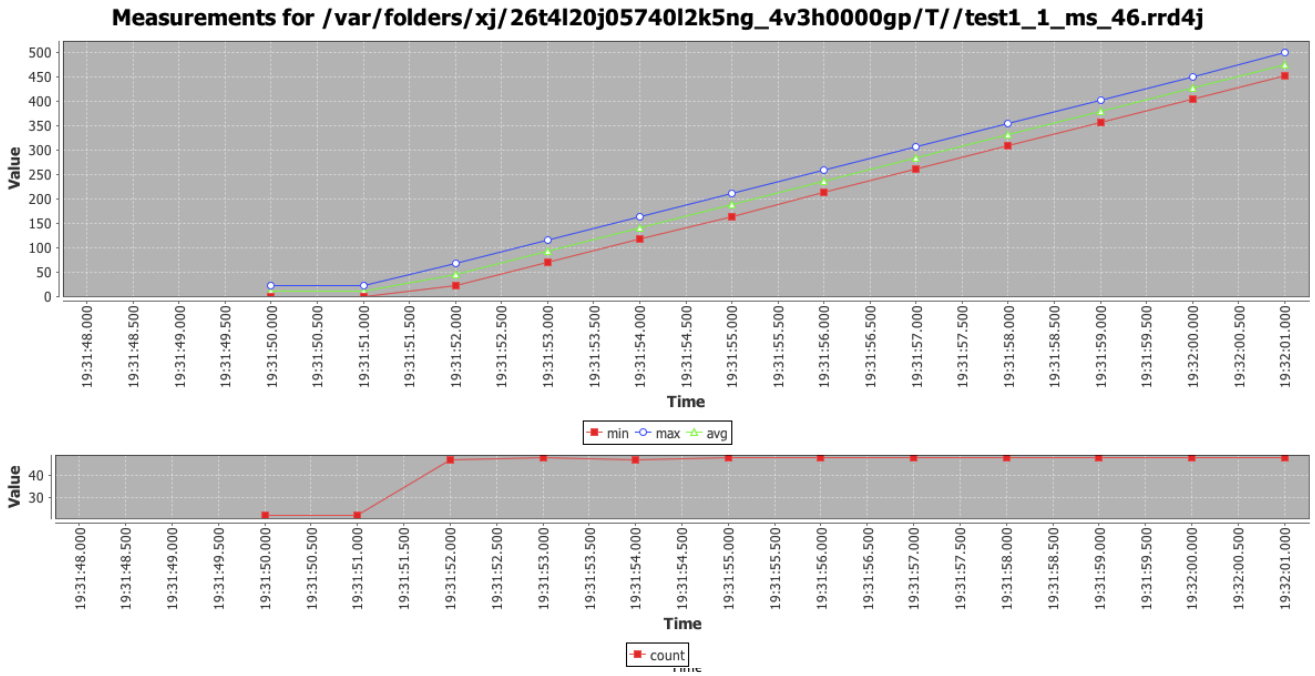
```
<aspectj>
   <aspects>
      <aspect name="com.zoltran.perf.aspect.PerformanceMonitorAspect"/>
   </aspects>
   <weaver options="-verbose">
      <include within="com.*..*"/>
   </weaver>
</aspectj>
```

This will record the execution times of the annotated method and will also log (via spf4j) a message containing the call detail and execution time if the warn or error thresholds are exceeded.

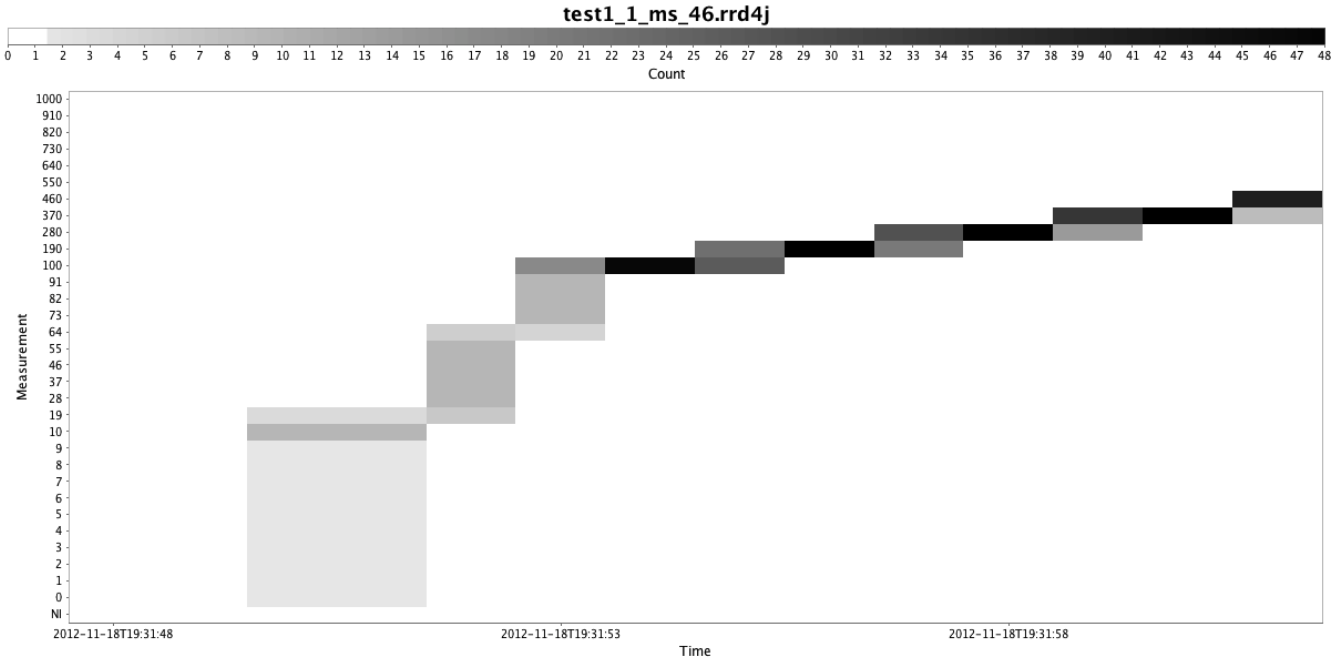
1.2. How to see the recorded measurements?

Via JMX invoke spf4j/RRDMeasurementDatabase/generateCharts this will generate 2 charts that will contain the recorded measurements:

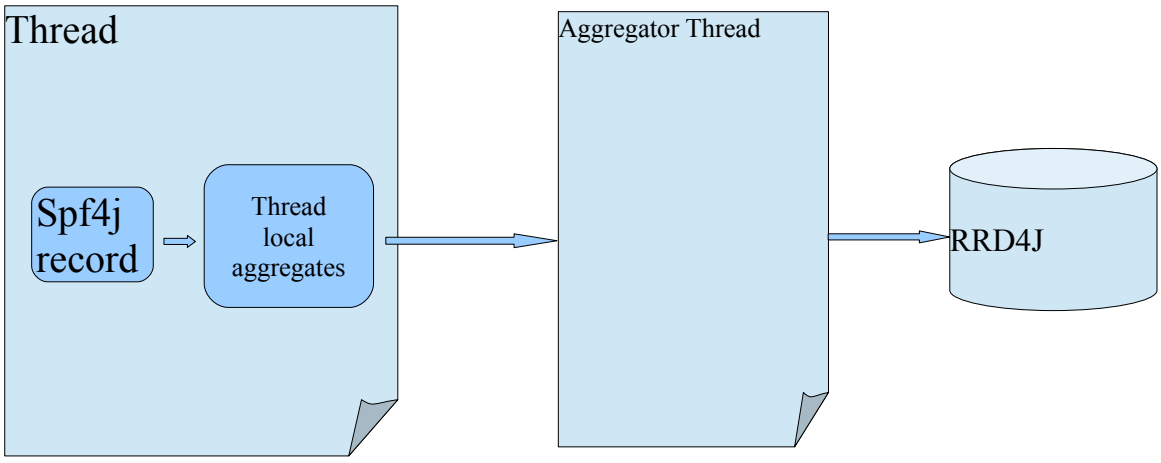
a) min, max, avg , measurement count values/ time axis



b)_Quantitative chart, where on the X axis we have the time, the Y axis we have the recorded values (buckets) and the number of measurements recorded that fall in a particular bucket will be represented by the color(values to collor mapping can be seen in the legend).



1.3. How does it work ?



Charts are generated using jfreechart library.

2. Performance Profiling

2.1. When to profile your code?

1. I recommend to deploy your code with profiling turned on as much as you can. In my case I have profiling data collection turned on in test/qa environments all the time. (with 100ms sampling interval). If you can afford to do it in PROD do it.

2.2. How to profile your code?

Add spf4j to your classpath and the following to your command line:

```
${JAVA_HOME}/bin/java [your jvm args] com.zoltran.stackmonitor.Monitor  -f reportFile.html -ss -si 100 -w 600 -main [your app main class] - -  [your app arguments]
```

This will start your application with profiling enabled, with a 100 ms sampling interval. After the process ends reportFile.html will be generated with a graphic width of 600 pixels. Profiling can also be enabled/disabled via JMX.

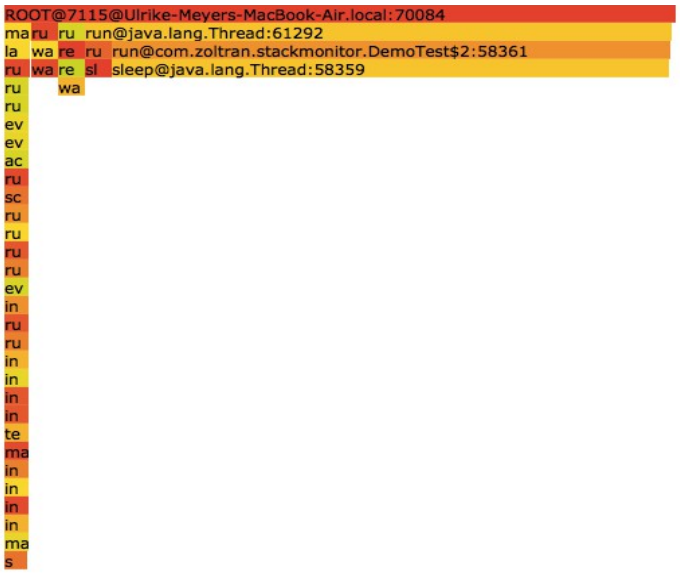
```
Supported arguments:
 -f VAL    : output to this file the perf report, format is HTML
 -main VAL : the main class name
 -md N     : maximum stack trace depth
 -nosvg    : stack visualization will be in svg format
 -si N     : the stack sampling interval in milliseconds
 -ss       : start the stack sampling thread. (can also be done manually via
             jmx)
```

-w N : flame chart width in pixels

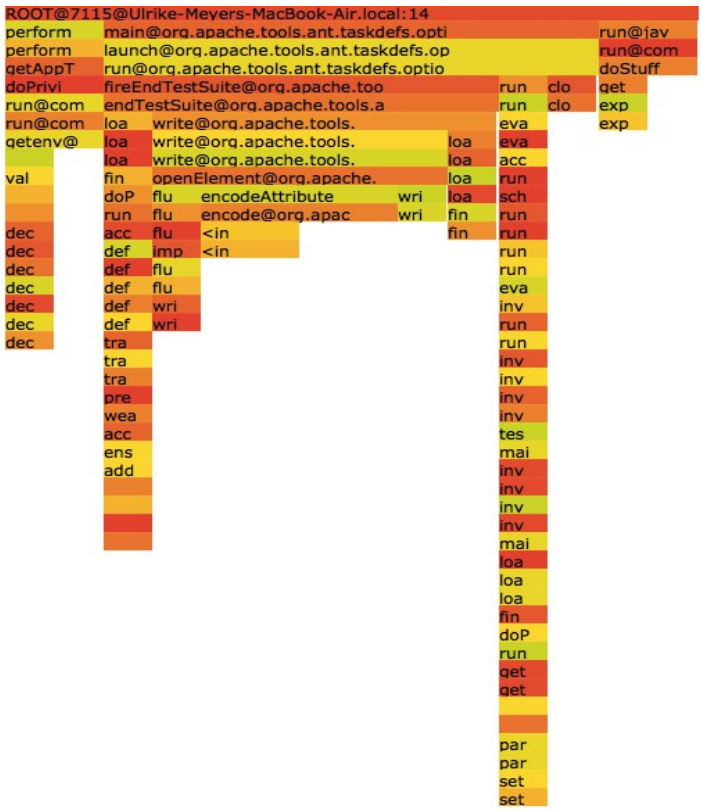
2.3. How to see the profile data?

After the program finishes it will write the data to the reportFile.html

The total view shows the total picture of what your application is doing.



Now as you can see this profile of my simple test code is sleeping most of the time. I you are interested into what the process is doing while it is on CPU, the next report will provide that detail:



Processes are staying quite a bit waiting for things to happen/sleeping and there is not much you can do about that, eliminating those states from the report allows you to see and concentrate on the “useful” parts of your application. As you can see above (if you look at the actual reports hovering with the mouse above a entry will display the details) DemoTest is also doing stuff (doStuff) not only sleeping...

2.4. How does it work?

A sampling thread is started and running in the background. This thread uses Thread.getAllStackTraces() or the JVM MX beans (configurable) to get all stack traces for all threads. Each sample is added to a tree that aggregates the stack trace data.