Design and implementation of probes for the WLCG SAM framework

Robert Vežnaver

CERN openlab
18th September 2010

oTN-2010-01 openlab Summer Student Report

Design and implementation of probes for the WLCG SAM framework

Student: Robert Vežnaver
Supervisor: Wojciech Łapka
17th September 2010
Version 0.2
Distribution: **Public**

Abstract

The SAM (Service and Availability Monitoring) framework tests the WLCG (Worldwide LHC Computing Grid) infrastructure. This framework is distributed across all NGIs (National Grid Initiatives) and a centralized instance at CERN. The main component of SAM is Nagios, which schedules and executes tests.

The aim of this project was to build a Nagios probe within the SAM framework which checks if the MRS (Metric Result Store) database is receiving enough test results. These results will ultimately help in building a robust Grid Monitoring Infrastructure for they will enable the detection of several issues such as problems with sending test results, problems with the messaging brokers, problems with messaging consumers, etc.

As a by-product of the main goal, procedures were developed which calculate the mean and standard deviation for metric frequencies from the data collected in the MRS database. These values were later used from within MyEGI to plot the normal distribution graph for a given metric.

Introduction

The SAM (Service and Availability Monitoring) framework is a distributed system for distributed monitoring. From a subsystem standpoint, it consists of Nagios instances deployed across NGIs and CERN, a messaging system for returning the results (metrics) of some probes and transferring all results back to CERN, and databases (both locally at NGIs and a central one at CERN).

One may also view SAM from a geographical standpoint. At the NGI level there is one Nagios instance that collects metric data, which is saved inside a MySQL database. The messaging system is used within the grid infrastructure for returning the metrics of passive checks. The transfer of metrics to the MySQL database is done via the org.egee.SendToMetricStore probe, while the transfer of metrics to the CERN central database is done via the org.egee.SendToMsg probe. At the CERN level there is a so-called "Super Nagios" instance (also called ops-monitor) that is responsible for monitoring the Nagioses of all NGIs. This Nagios also monitors the messaging system and the central Oracle database installation at CERN.

The SAM database (both MySQL and Oracle) currently consists of three database components (although merging is in progress):

- ATP (Aggregated Topology Provider), which describes the topology of services
- MDDB (Metric Description Database), which describes the metrics
- MRS (Metric Store), which holds the metric data

The NCG (Nagios Configuration Generator) is used for automatic generation of Nagios configuration files out of templates and the SAM database. As this project was primarily focused on operating on the MRS database and implementing an active Nagios probe, there is no need to explain the messaging subsystem.

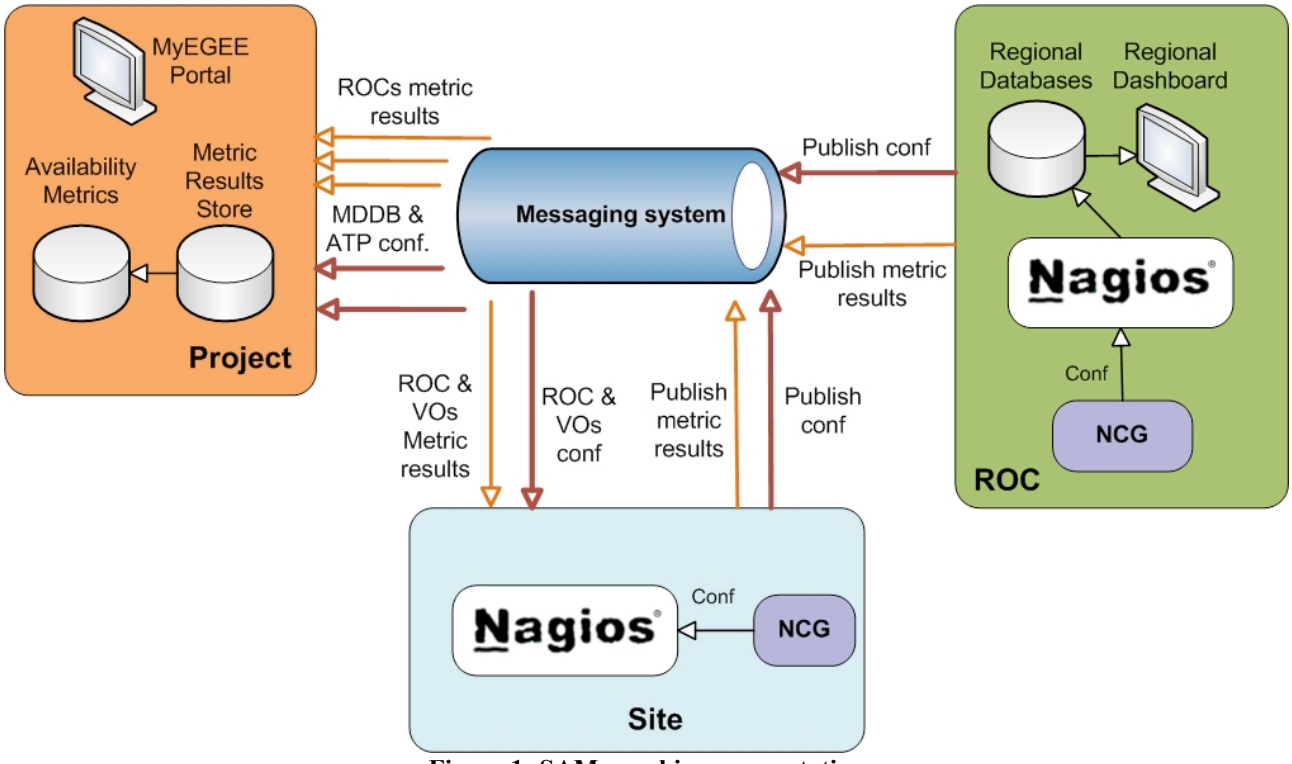

Figure 1: SAM graphic representation

1 Stored procedures in MRS

The *getFreshMetrics* stored procedure is the materialization of the main goal. This procedure tells the end user (usually the NGI system administrator) whether the metrics received from the NGI are "fresh" enough. If the metrics are fresh enough, it means that test results are coming in the MRS database within the expected timeframe and that the messaging system is working fine. If not, there may be a problem somewhere.

One of the main problems is defining "fresh". Each metric has a frequency (time between checks), which is actually the frequency of the Nagios probe (set up for each probe in NCG) that returns it. In the case of active Nagios probes, these frequencies are good values because in most cases active Nagios probes both finish and return the metrics before the start of the next check or simply do not return anything. So, taking into account these frequencies and adding a little bit of extra "wiggle" time (usually the time it takes the metric to transfer into the MRS database), the procedure works fine. The problem lies in the passive checks. During the alpha phase of the procedure development, it has been seen that passive tests in most cases return later than the specified frequency, which does not mean they do not return at all (obviously, "fresh" has a rather broad definition). To account for these deviations, a side procedure named *calculateMetricFreq* was created.

The *calculateMetricFreq* stored procedure is used for dynamic statistical calculation of metric frequencies. It takes into account all metrics in the *metricstore.metricdata* table, which are within the past two weeks. First it groups them by NGI, flavour, and metric and calculates the frequency for each metric (based on the time the metric arrived). After that it calculates the mean and standard deviation for each NGI, flavour, and metric pair. Based on the central limit theorem (CLT), the assumption is that all metric frequencies will be approximately normally distributed, with a mean shift towards the value they were set up with. This assumption was tested in *Mathematica*, which showed that metrics really do behave in this way.

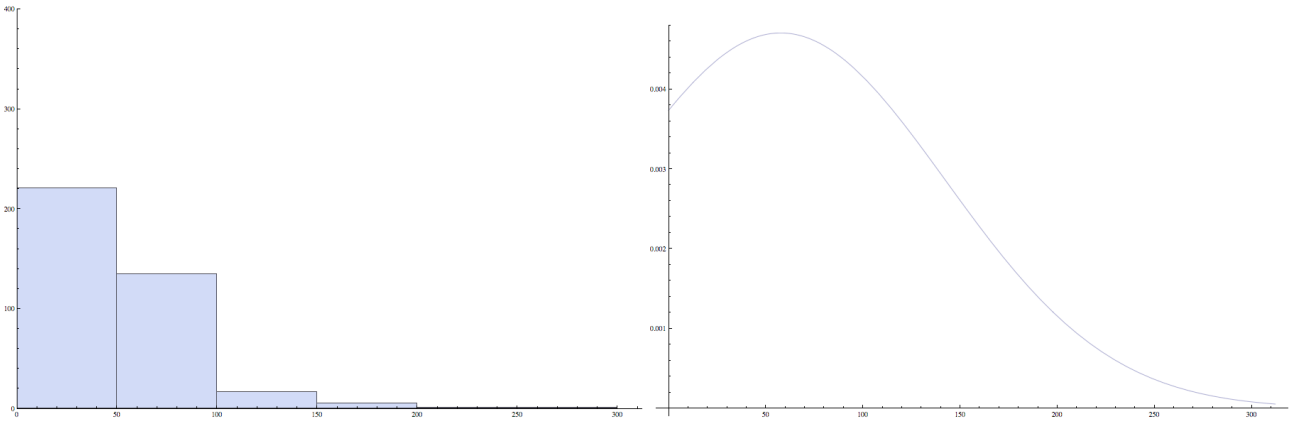

Figure 2: Histrogram and calculated normal distribution for NorthernEurope / CREAM-CE / JobSubmit
μ = 57.77, σ = 84.88

During the calculation, the procedure takes into account the fact that Nagios tends to increase the frequency of probes when they go into critical state. Discarding all frequencies that are two times higher than the original one mitigates this issue. The resulting data containing the metric frequency statistics is put into the *metricstore.mon_frequency_statistics* table. This table is generated weekly.

Taking into account the frequency statistics, a "fresh" metric is one that has the following property:

$$check\_time > now - \lceil freq * (devPerc/100 + 1) + devTime) \rceil$$

$$Nagios_{freq} < \mu_{freq} + 2\sigma_{freq} < 2 * Nagios_{freq} \Rightarrow freq := \mu_{freq} + 2\sigma_{freq}$$

$$\mu_{freq} + 2\sigma_{freq} < Nagios_{freq} \Rightarrow freq := Nagios_{freq}$$

$$\mu_{freq} + 2\sigma_{freq} > 2 * Nagios_{freq} \Rightarrow freq := 2 * Nagios_{freq}$$

check_time is the time when the result was available, *now* is the current time (both in UNIX timestamp format), *devPerc* is the allowed offset in percentage, *devTime* is the allowed offset in minutes. $Nagios_{freq}$ refers to the original frequency set up in Nagios, while $\mu_{freq} + 2\sigma_{freq}$ refers to the calculated mean and standard deviation.

getFreshMetrics filters out all fresh metrics from *metricstore.metricdata_latest* and puts the data in a temporary table named *tmpFilteredMetrics*. After that it collects all configured metrics for given parameters and puts them in the *tmpMetricCount* temporary table. It then joins these two tables and outputs data so the end user may see the number of fresh metrics, total metrics, and a percentage of fresh in total metrics. The output is ordered by NGI, flavour, and metric. The procedure may filter by a list of profiles, NGIs, service flavours, metric names, and FQANs.

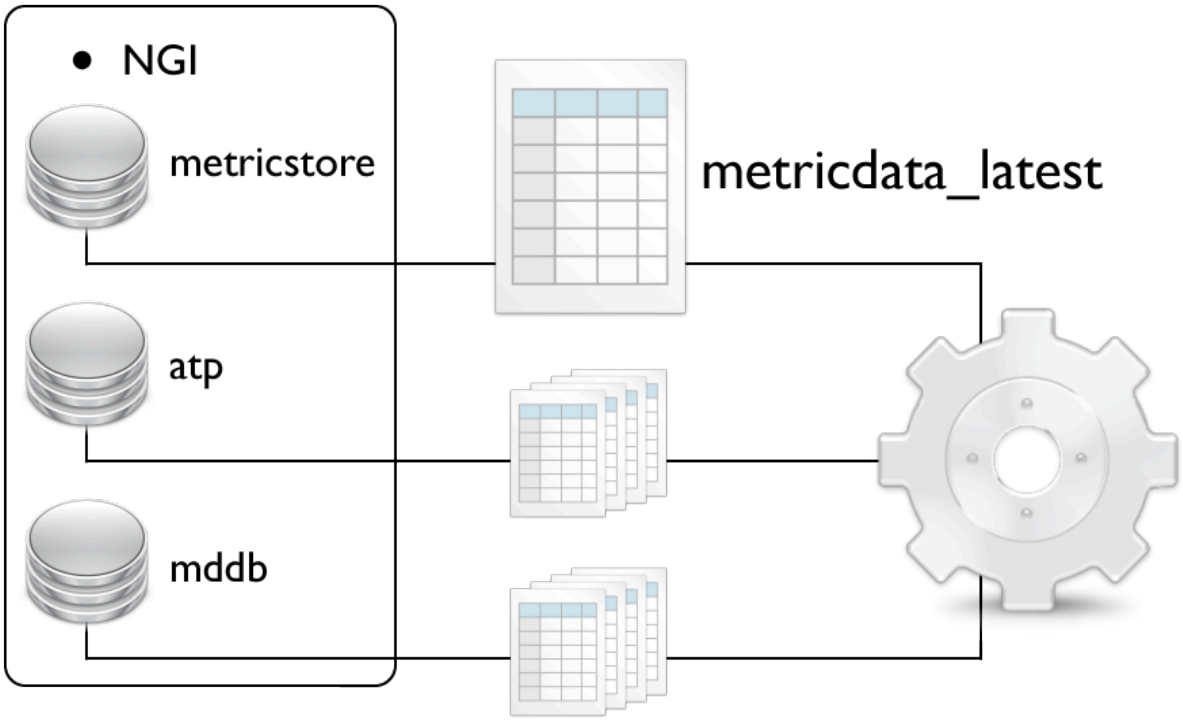

Figure 3: Graphic overview of getFreshMetrics stored procedure input

NGI	flavour_name	metric_name	fqan_name	received	total	percentage
UKI	CE	hr.srce.GRAM-CertLifetime	NULL	43	47	92
UKI	CE	org.sam.CE-JobSubmit	/ops/Role=lcgadmin	36	47	77
UKI	CE	org.sam.WN-Bi	/ops/Role=lcgadmin	32	47	69
UKI	CE	org.sam.WN-CAver	/ops/Role=lcgadmin	32	47	69
UKI	CE	org.sam.WN-Csh	/ops/Role=lcgadmin	32	47	69
UKI	CE	org.sam.WN-Rep	/ops/Role=lcgadmin	31	47	66
UKI	CE	org.sam.WN-SoftVer	/ops/Role=lcgadmin	32	47	69
UKI	CREAM-CE	hr.srce.CREAMCE-CertLifetime	NULL	7	7	100
…	…	…	…	…	…	…

Figure 4: Example output of getFreshMetrics

Although the stored procedures for MySQL and Oracle share a lot of code, there are some differences that need to be explained.

1.1 MySQL

Because there is no function for splitting a list of string in MySQL, alongside aforementioned procedures, there was another stored procedure created as a by-product of the main goal. *list2table* is a MySQL stored procedure which takes a comma-separated list of *varchars* and splits them into a temporary table. The procedure has only two parameters: the list, and the desired name of the temporary table that will hold the data. It must be called as a prepared procedure.

Apart from having every list in a separate temporary table, the *getFreshMetrics* procedure relies heavily on other temporary tables, namely *tmpFilteredMetrics* and *tmpMetricCount*. As temporary tables in MySQL are created locally, they are only visible after the execution of the procedure and only in the session in which they were created.

1.2 Oracle

Since *calcMetricFreq* is based on a cursor, which goes through the *metricdata* table, the only real difference between the MySQL and Oracle (or PL/SQL) version is in the syntax. Also, it should be noted that PL/SQL does not have a function which outputs date and time in a UNIX timestamp format, so the MRS *to_unixts* function was used instead.

The *getFreshMetrics* procedure in Oracle uses nested tables to hold the multiple lists. The lists are retrieved from the MRS function *split* which splits a comma-separated list into piped rows. The output is then put into a nested table. Oracle uses global temporary tables, and as a procedure in Oracle cannot return a table (unlike MySQL), apart from *tmpFilteredMetrics* and *tmpMetricCount*, it also uses the *tmpFreshMetrics* as a temporary data holder.

2 Nagios probe

The Nagios probe acts as a wrapper and finer-grained filter for the stored procedure inside the MRS database. It produces output in the standard Nagios format, as well as HTML formatted output that enable the probe to display both warning and critical information at once. The probe is configured via *Hash.pm*, one of the main NCG templates file. The configuration for the probe is as follows:

```
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{native} = "Nagios";
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{probe} = 'check_missing_probes_mrs';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{metricset} = "nagios";
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{config}->{path} =
$NCG::NCG_PLUGINS_PATH_GRIDMON;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{config}->{interval} = 5;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{config}->{timeout} = 30;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{config}->{retryInterval} = 3;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{config}->{maxCheckAttempts} = 3;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{flags}->{NOHOSTNAME} = 1;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{flags}->{PNP} = 1;
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--profileName'} =
'ROC_OPERATORS';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--NGIName'} = 'ANY';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--flavourName'} =
'ANY';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--metricName'} = 'ANY';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--fqanName'} = 'ANY';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--devPerc'} = '5';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--devTime'} = '15';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--warning'} = '90';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--critical'} = '70';
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--extra-opts'} =
'send_to_db@/etc/nagios/plugins/send_to_db.ini';
```

The settings up to *{flags}->{PNP} = 1* are more or less standard for any Nagios probe configured in NCG. *{flags}->{PNP} = 1* enables pnp4nagios. All parameters except the last three are directly passed to the stored procedure. The *{'--warning'}* and *{'--critical'}* setting set the global threshold such that if any metric percentage goes over it automatically raises an alert in Nagios. The *{'--extra-opts'}* setting is used for sending extra options contained in a text file to a probe (in this case, the database connection information).

The warning and critical threshold may be configured for each metric separately using the file */etc/nagios/plugins/check_missing_probes.cfg*. This is an Apache-style XML configuration file in which the users may specify their own threshold regarding of the situation. It has to be defined per flavour and per metric, although the user might save some time writing a simple regular expression. A sample configuration file looks like this:

```
<flavour CE>
 <metric org.sam.*>
  warning  70
  critical 60
 </metric>
</flavour>
<flavour CREAM-CE>
 <metric org.sam.*>
  warning  70
  critical 60
 </metric>
</flavour>
```

There is a slight difference in configuring the Nagios probe for NGIs, as to configuring it for the ops-monitor. The line:

```
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{parameter}->{'--NGIName'} = 'ANY';
```

… is replaced by:

```
$WLCG_SERVICE->{'org.egee.MrsCheckMissingProbes'}->{attribute}->{NGI_NAME} = "--NGIName";
```

Because the ops-monitor instance has one check per NGI, an *attribute* has to be defined which tells NCG that there is a variable that needs to be changed per site. These attributes are defined per site in *ncg.localdb*. For example:

```
HOST_ATTRIBUTE!sam-ap-roc.cern.ch!NGI_NAME!"AsiaPacific"
```

2.1 Perl and Nagios::Plugin

The probe is written in Perl 5 using the Nagios::Plugin module. The algorithm is quite simple:

1. Read configuration file
2. Connect to database
3. Call stored procedure with given parameters
4. For each returned row
    I. Set warning and critical threshold to global
    II. If there are different thresholds in the configuration file overwrite previous setup
    III. If percentage is below critical threshold add to critical stack, else if below warning threshold add to warning stack
    IV. Add data to perfdata (so it can be picked up by pnp4nagios)
5. Generate text output
6. Generate HTML output
7. Disconnect from database
8. Exit with appropriate alert level

There is a slight difference between the Oracle and MySQL version of the probe. When connecting to Oracle, the probe tries to read */etc/mrs.d/oracle-mrs.conf*, which contains Oracle connection parameters. Another thing is that when calling the procedure in Oracle, the probe does a *select* query on *tmpFreshMetrics*.

Service State Information

Current Status: **CRITICAL** (for 1d 19h 51m 1s)

Status Information: CRITICAL: AsiaPacific:CE:org.sam.WN-Bi(15/26) AsiaPacific:CE:org.sam.WN-Csh(15/26) AsiaPacific:CE:org.sam.WN-SoftVer(15/26) AsiaPacific:CREAM-CE:org.sam.WN-Bi(2/6) AsiaPacific:CREAM-CE:org.sam.WN-CAver(2/6) AsiaPacific:CREAM-CE:org.sam.WN-Csh(2/6) AsiaPacific:CREAM-CE:org.sam.WN-Rep(2/6) AsiaPacific:CREAM-CE:org.sam.WN-SoftVer(2/6) CERN:CREAM-CE:org.sam.CREAMCE-JobSubmit(1/2) CERN:CREAM-CE:org.sam.WN-Bi(1/2) CERN:CREAM-CE:org.sam.WN-CAver(1/2) CERN:CREAM-CE:org.sam.WN-Csh(1/2) CERN:CREAM-CE:org.sam.WN-Rep(1/2) CERN:CREAM-CE:org.sam.WN-SoftVer(1/2) WARNING: AsiaPacific:CE:org.sam.WN-CAver(16/26) AsiaPacific:CREAM-CE:org.sam.CREAMCE-JobSubmit(4/6)

NGI	Flavour	Metric name	FQAN	R/T	%
AsiaPacific	CE	org.sam.WN-Bi	/ops/Role=lcgadmin	15/26	58
AsiaPacific	CE	org.sam.WN-Csh	/ops/Role=lcgadmin	15/26	58
AsiaPacific	CE	org.sam.WN-SoftVer	/ops/Role=lcgadmin	15/26	58
AsiaPacific	CREAM-CE	org.sam.WN-Bi	/ops/Role=lcgadmin	2/6	34
AsiaPacific	CREAM-CE	org.sam.WN-CAver	/ops/Role=lcgadmin	2/6	34
AsiaPacific	CREAM-CE	org.sam.WN-Csh	/ops/Role=lcgadmin	2/6	34
AsiaPacific	CREAM-CE	org.sam.WN-Rep	/ops/Role=lcgadmin	2/6	34
AsiaPacific	CREAM-CE	org.sam.WN-SoftVer	/ops/Role=lcgadmin	2/6	34
CERN	CREAM-CE	org.sam.CREAMCE-JobSubmit	/ops/Role=lcgadmin	1/2	50
CERN	CREAM-CE	org.sam.WN-Bi	/ops/Role=lcgadmin	1/2	50
CERN	CREAM-CE	org.sam.WN-CAver	/ops/Role=lcgadmin	1/2	50
CERN	CREAM-CE	org.sam.WN-Csh	/ops/Role=lcgadmin	1/2	50
CERN	CREAM-CE	org.sam.WN-Rep	/ops/Role=lcgadmin	1/2	50
CERN	CREAM-CE	org.sam.WN-SoftVer	/ops/Role=lcgadmin	1/2	50
AsiaPacific	CE	org.sam.WN-CAver	/ops/Role=lcgadmin	16/26	62
AsiaPacific	CREAM-CE	org.sam.CREAMCE-JobSubmit	/ops/Role=lcgadmin	4/6	67

CentralMrsCheckMissingProbes CRITICAL

Figure 5: HTML output from Nagios probe

2.2 pnp4nagios PHP template

pnp4nagios is a plugin for Nagios which puts all perfdata from a given probe into a rrd database, and then reads from this database and creates graphs. Since visualization is an important part of this project, a special pnp4nagios PHP template was created which colours each metric percentage with a different colour, and groups metrics which have the same flavour and NGI into one graph. This enables the user to check the fluctuation in metrics even when there is no critical or warning alert.

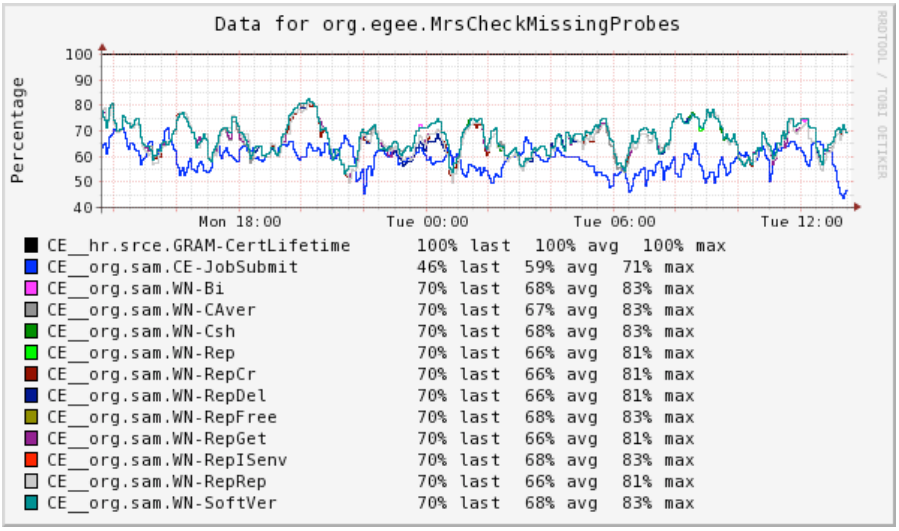

Figure 6: Metric fluctuation of CEs in 24h

3 MyEGI

MyEGI is the main visualization tool to present a grid-aware view of the data collected by the SAM framework. It is written in Python using the Django framework. It uses the Model-View-Controller design pattern and an Object Relational Mapper for database abstraction.

Since this project created a new table that gathers metric frequency statistics, it seemed obvious to visualize this data to the end user. The MyEGI portal already had a view for metric details, so a link was added on that view which points to a plot generator that plots the normal distribution based upon the parameters of the selected metric.

A new class was added to the model representing the *tmpMetricFreq* table, and a view method was added which returns a png image generated by numpy and matplotlib. This provides the user a feeling of how well the metric is actually behaving over the period of two weeks.

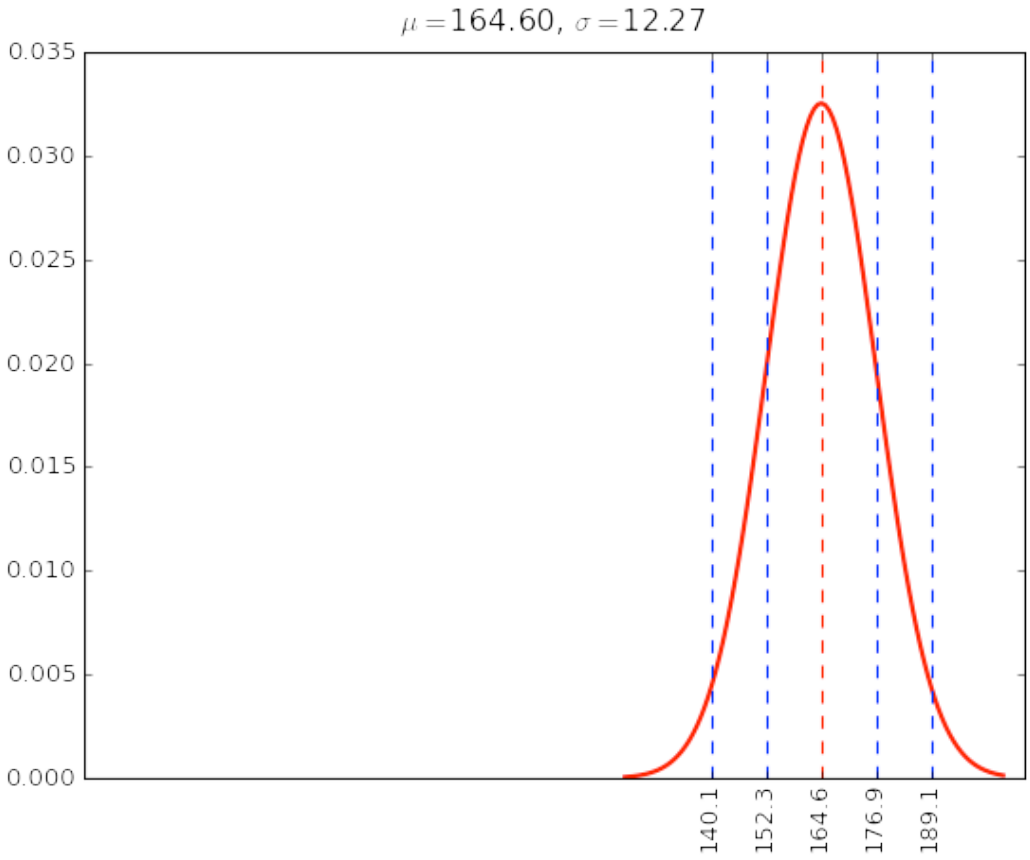

Figure 7: MyEGI normal distribution plot for a metric

4 Bibliography

[1] https://twiki.cern.ch/twiki/bin/view/EGEE/GridMonitoringNcgYaim

[2] https://twiki.cern.ch/twiki/bin/view/EGEE/GridMonitoringNcgOverview

[3] https://twiki.cern.ch/twiki/bin/view/LCG/SAMProbesMetrics

[4] https://twiki.cern.ch/twiki/bin/view/LCG/SamCern

[5] http://dev.mysql.com/doc/refman/5.1/en/stored-routines.html

[6] http://infolab.stanford.edu/~ullman/fcdb/oracle/or-plsql.html

[7] http://stanford.edu/dept/itss/docs/oracle/10g/appdev.101/b10807/toc.htm

[8] http://nagiosplug.sourceforge.net/developer-guidelines.html

[9] http://docs.pnp4nagios.org/pnp-0.4/start

[10] http://search.cpan.org/dist/Nagios-Plugin/lib/Nagios/Plugin.pm