Manuscript Number: CHEP191

Title: The HappyFace Project

Article Type: Poster

Corresponding Author: Viktor Mauch

Corresponding Author's Institution: KIT

First Author: Viktor Mauch

Order of Authors: Viktor Mauch;Armin Scheurer;Armin Burgmeier;Volker Büge;Günter Quast;Cano Ay;Stefan Birkholz;Jörg Meyer;Friederike Nowak;Arnulf Quadt;Philip Sauerland;Peter Schleper;Hartmut Stadie;Oleg Tsigenov;Marian Zvada

The HappyFace Project

Viktor Mauch [1], Cano Ay [3], Stefan Birkholz [3], Volker Büge [1], Armin Burgmeier [1], Jörg Meyer [3], Friederike Nowak [4], Arnulf Quadt [3], Günter Quast [1], Philip Sauerland [2], Armin Scheurer [1], Peter Schleper [4], Hartmut Stadie [4], Oleg Tsigenov [2] and Marian Zvada [1]

[1]Karlsruhe Institute of Technology, Kaiserstrae 12, 76131 Karlsruhe, Germany
[2]RWTH Aachen University, Templergraben 55, 52056 Aachen, Germany
[3]University of Göttingen, Wilhelmsplatz 1, 37073 Göttingen, Germany
[4]University of Hamburg, Edmund-Siemers-Allee 1, 20146 Hamburg, Germany

E-mail: `mauch@kit.edu`

Abstract.
An efficient administration of computing centres requires sophisticated tools for the monitoring of the local computing infrastructure. The enormous flood of information from different monitoring sources retards the identification of problems and complicates the local administration unnecessarily. The meta-monitoring system "HappyFace" offers elegant mechanisms to collect, process and evaluate all relevant information and to condense it into a simple rating visualisation, reflecting the current status of a computing centre. In this paper, we give an overview of the HappyFace architecture and selected modules.

1. Introduction

The experiments of the Large Hadron Collider (LHC) [1] at CERN [2] create a data output in the order of petabytes per year. Furthermore, scientists all over the world must have access to these data to run their specific analyses. The requirements concerning the computing power and storage space led to the decision to build a network of computing centres, which is called Worldwide LHC Computing Grid (WLCG) [3]. Each integrated computing centre provides data storage and processing power. Standardised interfaces and software environments allow distributing the work load equally on all available sites. A Grid site has very strict obligations concerning the operational availability which is necessary due to the reception and processing of the detector data coming from CERN.

The monitoring of such Grid enabled computing centres is an even more complex. A look at the current monitoring systems of the WLCG reveals numerous difficulties, especially concerning the monitoring of single Grid sites and their services. First of all, there are a lot of valuable monitoring applications providing rather unstructured information. For non-experts, it is very hard to find out relevant information responsible for specific problems and their causes. Furthermore, each monitoring application uses its own technology of presenting data values or graphic output. This complicates the possibility to identify correlations between different error sources.

Additionally, the majority of all monitoring systems is uncomfortable to use. A site operator has to check various different services to access all relevant information and often has to change the settings of the different web sources to reflect his specific needs. Another handicap is their high latency. Several monitoring websites, especially complex systems with a database backend, often require non-negligible amount of time to submit the settings to query the database and finally to retrieve the desired information. Thus, the time required for a regular site check is unnecessarily increased.

2. The meta-monitoring concept

To abolish these drawbacks an automated system with an adaptable configuration could query all relevant monitoring sources and provide a fast and comfortable overview for the local administration. Such a meta-monitoring system would act as a layer aggregating existing monitoring sources. Based on that, ratings can be generated to describe the status of the monitored instance. A preferable meta-monitoring suite provides features like:

- **flexibility:** possibility to deploy the suite on virtually all common operating systems
- **aggregation:** all relevant information should be collected on one website
- **actuality:** the complete status information should be renewed with a regular time interval
- **traceability:** possibility to call the site status for a specific time respectively time scale
- **accessibility:** architecture should be as simple as possible for high performance availability
- **comfortability:** thought out navigation should realize a fast access to all results
- **alert functionality:** visualisation should highlight results via smileys, arrows, ...
- **customizability:** user should be able to implement easily tests and define alert algorithms

Operators respectively administrators would be able to automate site checks and get a quick view on the status of their computing centre. Common Grid users could get a less detailed view of the system to recognize possible problems of a Grid site in case of failure of their jobs.

3. The HappyFace Core System

The HappyFace Project [4] is such a meta-monitoring suit. It provides a modular software framework, designed to query existing monitoring sources to process the gathered information and to create an overview of the status of a whole site and individual services.

The publication "Site specific monitoring of multiple information systems - the HappyFace Project" [5] describes in detail design, technical implementation and system requirements of the software. Therefore, this section just summarizes briefly the most important framework properties.

For a maximum of flexibility, the acquisition of information and its visualisation are fully decoupled. All collected data as well as the derived results are stored into a relational database. To keep the database access as fast as possible, binary files like plot images are stored on a local file system or network accessible from the monitoring machine. The visualisation of specific information is generated via a corresponding database query. This decoupling also allows the implementation of the requested history functionality. Besides offering a dynamic HTML webpage, it is also possible to get all status information results via an XML export.

The framework itself consists of two parts. All basic functionalities of the systems are provided by so-called HappyCore routines responsible for the following tasks:

- regular execution of all active modules
- access to the database and its initialisation
- calculation of the category status values via available rating algorithms
- composition of the final output by generating dynamic web pages

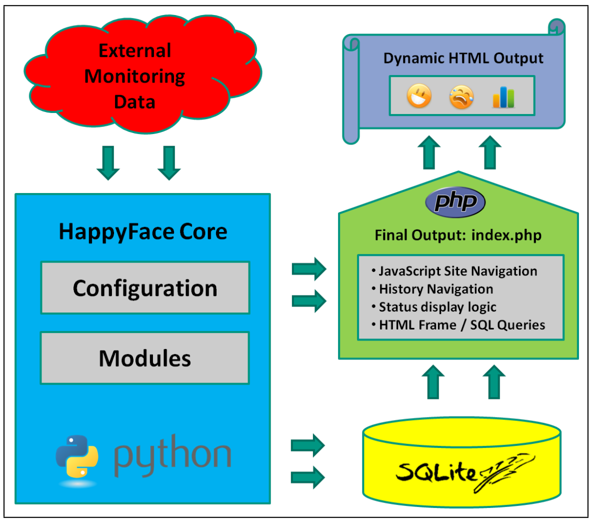


Figure 1. Workflow of framework.

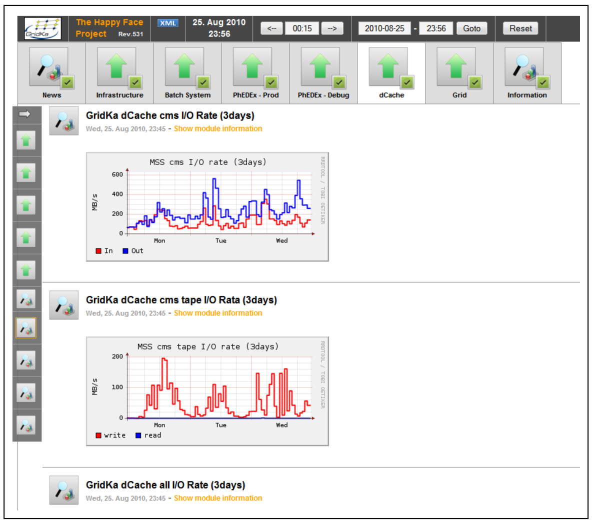

The Happy Face Project Rev.531
XML
25. Aug 2010 23:56
<--
00:15
-->
2010-08-25
23:56
Goto
Reset
News
Infrastructure
Batch System
PhEDEx - Prod
PhEDEx - Debug
dCache
Grid
Information
GridKa dCache cms I/O Rate (3days)
Wed, 25. Aug 2010, 23:45 - Show module information
MSS cms I/O rate (3days)
In
Out
GridKa dCache cms tape I/O Rata (3days)
Wed, 25. Aug 2010, 23:45 - Show module information
MSS cms tape I/O rate (3days)
write
read
GridKa dCache all I/O Rate (3days)
Wed, 25. Aug 2010, 23:45 - Show module information

Figure 2. Screenshot of the website.

The second part of the framework is represented by all specialised test modules inheriting the basic functionality provided by the HappyCore. Each test module has its own configuration and algorithms for processing the collected information. Because of the inheritance, the algorithms and the configuration can be made available in the parent class, which simplifies significantly the administration and integration of new related modules.

The complete framework is written in Python [6] (version 2.4 and beyond). To realize an easy implementation of new modules and functionality, it provides strictly modular inheritance layout. Figure 1 illustrates the workflow of the framework. After the initialisation of the HappyCore system, all enabled test modules collect the requested information from external monitoring sources. Afterwards, this information is processed using the implemented algorithms and the results, configuration parameters and as well the links to the binary data located on the local filesystem are stored in a SQLite [7] database. Finally, each module generates a PHP [8] fragment for its web output, including the logic for the database query. The framework collects all fragments and composes the final website. Each call of the website triggers a database query to retrieve the desired information. By altering the query time, history functionality is intrinsically included.

The enabled modules are grouped to categories concerning their content wise relevance. Each category has a well defined status value derived from the assigned modules. The HTML output presents all categories with the corresponding visualization of their status. In addition, the complete module content is accessible via comfortable category and module navigation bars. The website interface allows accessing all historic monitoring information by selecting a specific timestamp inside the menu bar. A screenshot of the HappyFace Project website used for the monitoring of the Tier-1 centre GridKa is presented in Figure 2.

4. Common Module properties

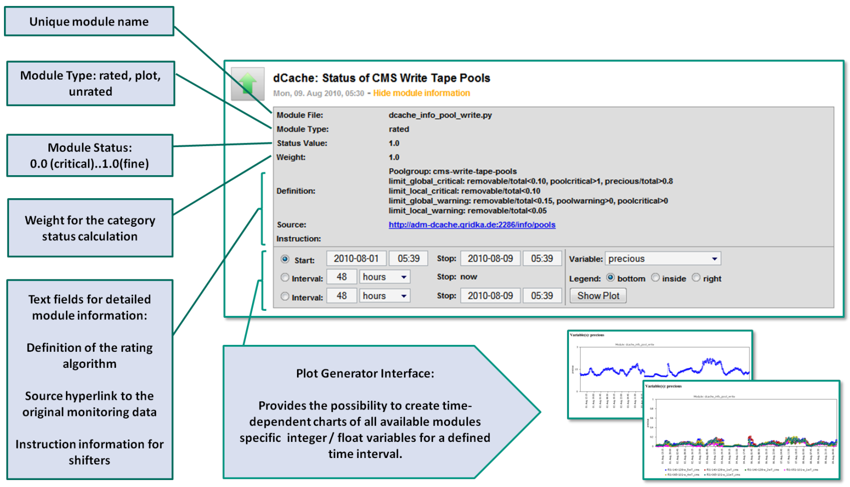
The HappyFace framework allows to develop own individual test modules with an arbitrarily large complexity. However most of the needed functionality is already available in the HappyCore. A simple inheritance from available basic modules simplifies the development enormously. Represented by a Python class, each module consists of 3 basic routines which are executed by the HappyCore:

- **__init__:**
  read out existing configuration file parameters and initialize further database variables
- **process:**
  execute the test procedure and calculate the status value
- **output:**
  create a PHP fragment for the final output including the corresponding SQL queries

Download requests for external data are handled before the module processing. This part as well as the module process routine can crash during the execution because of several possible failures like unavailable download data or other exceptions like division by zero. To prevent that the whole framework is affected a multi-threading environment is used to execute all download requests respectively all module process routines at the same time and break corresponding threads after a defined time-out. This procedure increases the stability and execution speed of the whole framework.

Test modules have a couple of universal properties and functionalities. Besides the module name, there are a few further general variables describing each test module. Figure 3 shows a typical module information box. The variable **module type** specifies the character of the content. Modules with a well defined status value are described as "rated" and visualized with the corresponding status symbol (arrows, smileys, ...). The status values of all test modules within a category are calculated to a global category status value. This contrasts with the module type "unrated" which defines a test whose rating value will not be combined to the category status. In another case the module type "plot" is used for modules which show unusual content like external plot images or basic information without any rating algorithm. To prioritise modules, a **weight** can be specified. Based on the current module status and weight values, different category rating algorithms can be defined by the user in the HappyCore. The category status could be equal with the worst status of all allocated modules. Another reasonable calculation would take the weighted average of all modules.

To understand how the status value is calculated, the variable **definition** is designed to provide the exact rating algorithm for the module's status. Often the user wants to see the very latest information concerning a special monitoring service, therefore the variable **source** contains hyperlinks to the original source. Furthermore the variable **instruction** can be filled with strict procedure descriptions concerning occurring alerts. Especially shifters are dependent on predefined workflows in case a problem occur.


Unique module name
Module Type: rated, plot, unrated
Module Status: 0.0 (critical)..1.0(fine)
Weight for the category status calculation
Text fields for detailed module information:
Definition of the rating algorithm
Source hyperlink to the original monitoring data
Instruction information for shifters
Plot Generator Interface:
Provides the possibility to create time-dependent charts of all available modules specific integer / float variables for a defined time interval.
Start:
Interval:
Stop:
hours
Variable:
Legend:
bottom
inside
right
Show Plot
Variable(s): precious
Module: dcache_info_pool_write

dCache: Status of CMS Write Tape Pools
Mon, 09. Aug 2010, 05:30 - Hide module information

Module File:	dcache_info_pool_write.py
Module Type:	rated
Status Value:	1.0
Weight:	1.0
Definition:	Poolgroup: cms-write-tape-pools limit_global_critical: removable/total<0.10, poolcritical>1, precious/total>0.8 limit_local_critical: removable/total<0.10 limit_global_warning: removable/total<0.15, poolwarning>0, poolcritical>0 limit_local_warning: removable/total<0.05
Source:	http://adm-dcache.gridka.de:2286/info/pools
Instruction:	

● Start: 2010-08-01 05:39 Stop: 2010-08-09 05:39
○ Interval: 48 hours Stop: now
○ Interval: 48 hours Stop: 2010-08-09 05:39
Variable: precious
Legend: ● bottom ○ inside ○ right

Figure 3. General module properties.

The bottom of the module information box provides an interface for a plot generator. The framework is able to show a time-dependent chart of any available numerical variable of the test module. This is a very helpful feature for an investigation concerning occurred alerts in the past. The possibility to compare variables of different modules in a specific time scale allows to localise correlations which help to indicate wherefrom the cause of failure originates.

5. Selection of Essential Modules

Since the regular operation of the HappyFace framework starting in 2008 more and more developers from different Grid sites are contributing new functionalities, improvements and above all new modules for various test scenarios. At the end of 2010 the project repository includes test modules processing most of the available monitoring Grid services. This section contains a small selection of essential modules to demonstrate the current breadth of the HappyFace functionality.

5.1. RSS Feed Module

The RSS Feed Module allows to subscribe Really Simple Syndication (RSS) [9] feeds. This functionality is useful for external sources and service portals which provide news respectively ticket propagation via RSS. It is perfectly suited to instruct shift crews with needful information.

5.2. Job Statistics Module

One important indicator concerning the correct operation of a Grid site is the job statistics. Therefore this module was developed to summarize the current jobs in the batch queue. A hierarchical view visualises the membership of the jobs. The user is able to define a warning trigger based on the fraction of jobs which have a wall time ratio under 10%. A better differentiation of the job statistics concerning the corresponding user and the job efficiency is available in a further sub table. To compare different user, user groups and efficiency values extended plot routines allow to create time-dependent charts including more than one quantity.

5.3. User Space Monitoring

This module provides summarized information concerning the used disk space per user. The information is available via an XML export from the concerning Grid site. To handle privacy for different users there are two different views. The user mode shows only the information for the signed-in user which is not able to see corresponding information of other users. The second mode implemented for administrators allows seeing all detailed information about each user. In addition information about group space usage is also available. The authentication is based on user certificates. Certain certificates can be configured to allow access to the admin mode.

5.4. SAM Visualization Module

The Site Availability Monitoring (SAM) [10] provides a Grid wide information service concerning the availability of the current computing and storage elements of a given Grid site. The corresponding HappyFace module requests the site specific information and provides a summary of the SAM tests. Furthermore there are sub tables providing a summary of the test results with direct hyperlinks to the original runtime test reports. The configuration allows triggering for a specific combination of well working respectively failed computing and storage elements.

5.5. dCache Data Management

The goal of dCache [11] is to provide a system for storing and retrieving huge amounts of data, distributed among a large number of heterogeneous server nodes. This test module processes information like the used space, number of files on disk and corresponding replicas. The input interface reads the Chimera [12] XML export. A sub table displays the available fractions of all data sets on disk. High available data sets are marked with green colour. The warning level is dependent on the timestamp of the last Chimera dump. If the last dump is too old, there has to be a reason why it was not generated and the shown information is not up-to-date.

5.6. dCache Pool Module

A further dCache module processes the pool information provided by XML output. The configuration allows selecting all information for a specified pool group. The module provides a summary of the usage information including the possibility to define a rating value based on the space usage characteristics of the pools. In addition a sub table with detailed information of each pool comes with the functionality to create time-dependent charts of specific pool quantities such as free space, precious space or used space. Therewith it is very easy to see if the load-balancing between the pools works fine.

6. Conclusion & Outlook

The secure operation of computing centres is dependent on sophisticated monitoring infrastructure. A multitude of several sources and their different presentation interfere an efficient administration. The meta-monitoring system the HappyFace Project improves this situation by providing a summary of all important site specific information high available on one website with the possibility to check the centre's status at a previous point in time.

The system was successfully used during several service challenges. Modularity allows providing new modules easily and sharing them among different sites. The XML output functionality allows to request the status of several Grid sites using HappyFace with the goal to create a cross-site status summary. Today several German Grid sites (ATLAS and CMS) use the HappyFace Project for their site monitoring:

- **Tier 1: GridKa - Karlsruhe Institute of Technology**
- **Tier 2: DESY - University of Hamburg**
- **Tier 2: RWTH University**
- **Tier 2: University of Göttingen**
- **Tier 3: Karlsruhe Institute of Technology**

Furthermore the CMS collaboration started to use one central HappyFace instance to collect jobs statistics from all CMS Tier-1 sites. Therefore various XML producers are available for different batch systems. In future it is planed as well to add Tier-2 production sites for monitoring via the HappyFace Project.

References

[1] Large Hadron Collider, `http://lhc.web.cern.ch/lhc/`
[2] Organisation Européene pour la Recherche Nucléaire (CERN), `http://www.cern.ch/`
[3] Worldwide LHC Computing Grid, `http://lcg.web.cern.ch/lcg/`
[4] HappyFace Project, `https://ekptrac.physik.uni-karlsruhe.de/HappyFace`
[5] Site specific monitoring of multiple information systems - the HappyFace Project, `http://iopscience.iop.org/1742-6596/219/6/062057`
[6] Python, `http://python.org/`
[7] SQLite, `http://sqlite.org/`
[8] PHP, `http://www.php.net/`
[9] Really Simple Syndication, `http://www.rssboard.org/`
[10] Site Availability Monitoring (SAM), `http://lxarda16.cern.ch/dashboard/request.py/samvisualization`
[11] DCache, `http://www.dcache.org/`
[12] Chimera, `http://trac.dcache.org/projects/dcache/wiki/Chimera`