



GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Fakultät für
Physik



Bachelor's Thesis

Überwachung des Grid-Ressourcen- Zentrums GoeGrid mit HappyFace

Meta-Monitoring of the Grid Resource Centre GoeGrid with HappyFace

prepared by

Georg Jahn

from Leinefelde

at the II. Institute of Physics

Thesis number: II.Physik-UniGö-BSc-2011/05
Thesis period: 7th April 2011 until 14th July 2011
Supervisors: Dr. Jörg Meyer, Dr. Pavel Weber
First referee: Prof. Dr. Arnulf Quadt
Second referee: Prof. Dr. Johannes Haller

Abstract

Um eine hohe Produktivität zu gewährleisten, wird das Rechen-Cluster GoeGrid (ein ATLAS Tier 2 Zentrum des LHC Computing Grid) ständig von mehreren Programmen überwacht. Zur einfacheren und effektiveren Wartung wird das Meta-Monitoring System HappyFace eingesetzt, das eine Zusammenfassung von Überwachungsinformationen aus verschiedenen Quellen erstellt. Im Rahmen dieser Bachelorarbeit wurde die lokale HappyFace Instanz komplett überarbeitet und erweitert; zudem wurde auch an den HappyFace Kernmodulen gearbeitet. Viele der Änderungen sind in die öffentliche HappyFace Version eingearbeitet worden, um schließlich auch andere Rechenzentren von den Beiträgen profitieren zu lassen.

The computer cluster GoeGrid, employed as an ATLAS Tier 2 centre for the LHC Computing Grid, is constantly monitored by several tools to ensure an effective operation. In order to simplify and improve the maintenance of GoeGrid, the meta-monitoring system HappyFace summarises monitoring information from a number of sources. In the course of this Bachelor's Thesis, the local installation of HappyFace was completely revised, new information sources were included, and a number of changes to the HappyFace framework itself were made. Many contributions were also submitted to the central HappyFace repository, so that other sites may take advantage of them.

Keywords: The HappyFace Project, Meta-Monitoring, GoeGrid, Grid Computing, ATLAS, WLCG

Contents

1. Introduction	1
1.1. The Worldwide LHC Computing Grid	1
1.1.1. Grid Computing	3
1.1.2. The Tier Sites	4
2. The Grid Resource Centre GoeGrid	5
2.1. Establishment	5
2.2. Hardware Setup	6
2.3. Software Setup	6
3. The HappyFace Project	9
3.1. Meta-Monitoring	9
3.2. HappyFace Concepts	10
3.3. Recent Developments	12
3.3.1. Core Module HostProcessing	13
3.3.2. Core Module ModuleHelper	13
3.3.3. Prospect	14
4. Application of HappyFace to GoeGrid	15
4.1. Setup	15
4.2. Modules from Central Repository	16
4.3. Local Modules	18
4.4. Benefits	19
5. Revised and New Modules	21
5.1. LogMessages	21
5.2. Nagios Messages	22
5.3. Hardware Ping Test	23
5.4. ILO Query	24
5.5. PBS Information	25

Contents

6. Summary	27
6.1. Conclusion	27
6.2. Prospect	28
A. Excerpts and Examples	29
A.1. Host Lists	29
A.2. Example for the Usage of <code>ModuleHelper.py</code>	30
A.3. Example for the Usage of <code>LogMessages</code>	33
A.4. HappyFace Installation Scripts	34
B. Sources	35

1. Introduction

In 1994, the ATLAS collaboration published a proposal^[1] of the basic design parameters of the ATLAS EXPERIMENT which is one of the major experiments of the LARGE HADRON COLLIDER (LHC)^[2,3] at the European high energy physics research centre CERN. About fifteen years before the LHC was placed into operation, this paper estimated a raw data output of about 10^6 GB per year as well as a required computing power of more than 10^{12} instructions per second just for the offline data reconstruction – at that time nearly infeasibly high requirements. Till this day, these requirements have even risen, but have proven to be realisable, albeit only with the combined resources of ten thousands of computers networked within a COMPUTING GRID^[4].

Obviously, the administration of these amounts of hardware poses a huge challenge. Even though the computers are grouped into smaller clusters in computing centres all around the world, the maintenance of a single cluster is a non-trivial task and cannot be done all manually. Resources in this magnitude require continuous automated monitoring to ensure a fast identification of failures and to minimise response times. Only with sophisticated software tools for administration and monitoring an efficient and productive operation of a cluster is possible.

The emphasis of this thesis is placed on the monitoring tool HAPPYFACE and in particular its application to the cluster GOEGRID. Some of the services that are used to provide the formerly impossible computing resources within the WORLDWIDE LHC COMPUTING GRID will be described and the focus will be placed on the ongoing efforts of monitoring them – with the goal to push the capacities of the available hardware further to the boundaries.

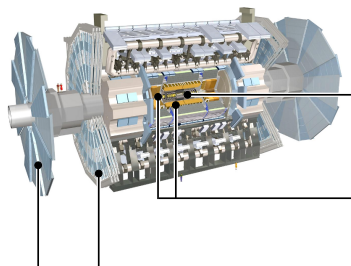
1.1. The Worldwide LHC Computing Grid

It was not only the ATLAS EXPERIMENT which in the nineties pronounced the need of enormous amounts of computing resources: the other three major experiments CMS, ALICE, and LHCb of the LHC estimated similar requirements. The LHC was designed as a proton-proton collider with a center-of-mass energy of 14 TeV at a luminosity of up

1. Introduction

to $10^{34} \text{ cm}^{-2}\text{s}^{-1}$. These unrivalled parameters are dictated by the diverse goals of the collider: the most prominent target is the discovery (or the exclusion) of the Higgs boson which is predicted by the Standard Model of particle physics (SM) with an estimated production cross section of only several $10 \text{ pb}^{[5,6]}$ even at 14 TeV proton-proton collisions in the SM. Likewise, precision tests of the SM and the search for new physics beyond the SM (e.g. supersymmetric particles) require such high luminosities.

In order to achieve those, the LHC is filled with up to 2,808 bunches each containing about 10^{11} protons which travel nearly at speed of light around the 27 km tunnel resulting in a 31.6 MHz average bunch crossing rate (for technical reasons, larger gaps without any bunches are inserted, so that the bunch crossing frequency is 40 MHz)^[3]. For the ATLAS EXPERIMENT, each bunch crossing produces on average about 23 collisions at design luminosity leading to approximately $7 \cdot 10^8$ events per second^[3]. The ATLAS detector consisting of the nine major subdetectors listed in Table 1.1 gathers information about the events through a total of about $9 \cdot 10^7$ read-out channels producing an average of 1.6 MB^[1] raw data per event.



	Detector Element	Channels
Inner Detector	Pixel Detector ^[7]	80,363,520
	Semiconductor Tracker ^[8]	6,279,168
	Transition Rad. Tracker ^[9]	424,576
Calorimeters	Liquid Ar Calorimeter ^[10]	182,468
	Tile Calorimeter ^[11]	463,500
Muon System	Thin Gap Chambers ^[12]	321,072
	Resistive Plate Chambers ^[13]	350,000
	Monitored Drift Tubes ^[14]	354,240
	Cathode Strip Chambers ^[15]	55,296

Table 1.1.: The components of the ATLAS detector and their numbers of read-out channels.

The resulting vast amount of data obviously cannot be stored completely and it would be pointless to do so, since most events only involve well-understood processes. That is why a three level trigger system is introduced: on each trigger level, an algorithm decides whether the event may contain sought-after information or can be discarded. Dedicated clusters of special-purpose processors for the first trigger level and 2,200 dual core processors for the second and third trigger level reduce the event rate to 200 “interesting” events (which is to say 320 MB) per second. This way, the four major experiments of the LHC produced a combined data output of 13 PB in 2010^[16].

To be able to store all this data redundantly, to analyse it, and to make it available

to scientists around the world, a suitable computing infrastructure had to be established parallel to the construction period of the experiments. However, the requirements were beyond the capacities realisable with CERN's budget – the solution was the setup of a GRID involving more than hundred computing centres in 35 countries, the WORLDWIDE LHC COMPUTING GRID (WLCG)^[17].

1.1.1. Grid Computing

The idea of grid computing was particularly coined in 1998 by IAN FOSTER and CARL KESSELMAN in their book “The Grid: A Blueprint for a New Computing Infrastructure”^[4]; both are often called the fathers of the grid. The term grid itself is derived from the power grid, alluding to the basic intention to make computing resources available wherever required – just like from a power socket.

In that sense, a grid bundles computing resources from several (often heterogeneous) domains connected by a network and makes them available to entities that are currently in need of those resources. To achieve this, the individuals involved in the resource-sharing agree on a common GRID MIDDLEWARE, a software handling the basic communication as a platform for resource sharing. In many cases, all participants are grouped into so-called VIRTUAL ORGANISATIONS (VOs) – an example for such a middleware that relies on VOs is GLITE, a framework which is employed by the WLCG. The membership in a VO authorises to access and use certain subsets of the resources, that could be to store a file or to submit a computing job to a cluster.

To distinguish the notion of grid computing from somewhat similar terms like cloud computing, IAN FOSTER published a list of three common aspects for a grid, which are widely accepted as a good definition:^[18]

- A grid coordinates resources that are not subject to centralized control.
- A grid uses standard, open, general-purpose protocols and interfaces.
- A grid delivers non-trivial qualities of service.

The WLCG is an excellent example for a grid, since its members are spread over the whole globe, it uses the open-source middleware GLITE and provides more than 100 PB of disk space plus the capacity of more than 1.3 million CPUs (2011)^[19] – which also makes it the world's largest computing grid^[20].

1. Introduction

1.1.2. The Tier Sites

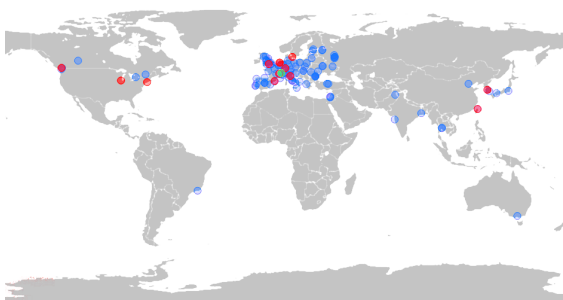
The WLCG is hierarchically structured into four layers that have different duties and functions in the data processing chain. These layers are named starting from Tier 0 to Tier 3; the grid computing centres (the so-called sites) belonging to Tier 0 and Tier 1 are listed below in Table 1.2.

Tier 0 This tier consists of only one site, the CERN Computing Centre (CCC), located on the CERN area near Geneva. All LHC raw data is processed and permanently stored here, then it gets distributed to the Tier 1 sites. The CCC also does the first reprocessing of the raw data.

Tier 1 The national Tier 1 computing centres are large enough to keep a share of a copy of all important Tier 0 data, so that it is redundantly stored. Furthermore, they offer high computing power for data reprocessing and distribute data to the Tier 2 sites.

Tier 2 These regional sites handle the more specific data analysis tasks and detector calibrations as well as a share of Monte Carlo simulations.

Tier 3 The produced LHC data is also provided to institutes that do not have any formal duties for the WLCG. These single computers as well as small local clusters form the Tier 3.



Tier	Sites	Site Names
Tier 0	1	CERN-PROD (Switzerland)
Tier 1	11	TRIUMF-LCG2 (Canada) IN2P3-CC (France) FZK-LCG2 (Germany) INFN-T1 (Italy) NIKHEF (Netherlands) NDGF-T1 (nordic countries) PIC (Spain) ASGC-LCG2 (Taiwan) RAL-LCG2 (UK) USCMS-FNAL-WC1 (USA) BNL-ATLAS (USA)
Tier 2	135	e. g. GOEGRID (Germany)

Table 1.2.: Sites participating in the WLCG grouped by Tiers; site locations on a world map^[21].

2. The Grid Resource Centre GoeGrid

The scientific motivation for cluster monitoring was elaborated in the first chapter; this chapter introduces the subject of the monitoring, the GOEGRID computer cluster. The term GOEGRID refers in fact to both the physical cluster located in Göttingen (Germany) and the association of the multiple communities around it, which profit from the synergy effects arising from the joint cluster operation.

2.1. Establishment

When the German Federal Ministry of Education and Research started an initiative for the construction of a sustainable German grid infrastructure in 2005, the D-GRID^[22], representatives of the communities MEDIGRID^[23] and TEXTGRID^[24] began to plan the setup and the funding of common grid resources in Göttingen. They had already found a local technical partner, the IT service provider GWDG^[25], with the required expertise and capabilities for their venture, when shortly afterwards the High Energy Physics (HEP) of the University Göttingen joined them as the third propellant community, motivating all participants to a joint cluster operation. Even more communities like the theoretical physics institute and the institute for astrophysics of the University Göttingen joined the informal association GoeGrid later on.^[26]

With fundings from the D-GRID initiative, the Helmholtz Alliance “Physics at the Terascale“, the institutes of theoretical physics and of the II. Institute of Physics (HEP) they were able to set up the required computing facilities and to integrate them into the existing infrastructure of the GWDG. In May 2008, the newly founded GRID RESOURCE CENTRE GOEGRID celebrated the official inauguration of the resulting computing infrastructure, after it became WLCG certified in April and thus an official Tier 2 centre within the WLCG^[26].

Within the established formal structure of administration, regular Executive Board and Technical Board meetings were introduced and the participating communities agreed on

2. The Grid Resource Centre GoeGrid

their resource shares, the cluster usage policies and their duties for the maintenance of the cluster. As a result, a job scheduling algorithm which ensures that each community receives cluster computing time proportional to their financial and non-financial contributions was introduced. Furthermore, the responsibilities for the maintenance of hardware and software were assigned to the involved communities.

2.2. Hardware Setup

In Figure 2.1 the status quo of GoeGrid as of July 2011 is illustrated: the central connection hub is formed by four interconnected routers which are also connected to the internet. Each of these routers has a number of worker nodes attached to it, the exact manner of the connection and the type of the compute nodes both vary since the cluster evolved in several extensions over time. The resulting slightly heterogeneous structure is maybe not the most intuitive configuration, but it does not affect the functionality of the cluster since the notion of grid computing is to connect even very heterogeneous systems and the small differences between the single nodes are nearly transparent to the batch system.

Furthermore, 14 storage nodes (often called SE for storage element) and various servers are connected to the central routers. Most of the services are provided in virtual machines (on the VIRT2...6 servers) to make them independent from each other, flexible, and easily reinstallable.

A summary of the provided computing power is shown in Table 2.1: the 289 nodes can provide a total peak computing power of about 27 TFlop/s and have a combined HEPSpec2006^[27] score of 25500, each of the 2,484 cores having 2-3 GB of RAM available. The storage element offers more than half a petabyte of dCache^[28] disk space dedicated for the HEP community and an additional 352 TB is provided via the STORAGE AREA NETWORK (SAN) service for the all communities (January 2011)^[29].

2.3. Software Setup

All communities involved in the GoeGrid have agreed on a common software setup for the worker nodes: as operating system CENTOS 5^[30] (64 Bit) is used, which is very similar to SCIENTIFIC LINUX^[31], but supported for the WLCG certification. On every reboot, each node can be reset to a “fresh” installation by the ROCKS CLUSTER DISTRIBUTION^[32]. Rocks furthermore provides the cluster monitoring tool GANGLIA and several other useful cluster management tools.

The incoming jobs are internally distributed by the batch system PBS^[33] which is

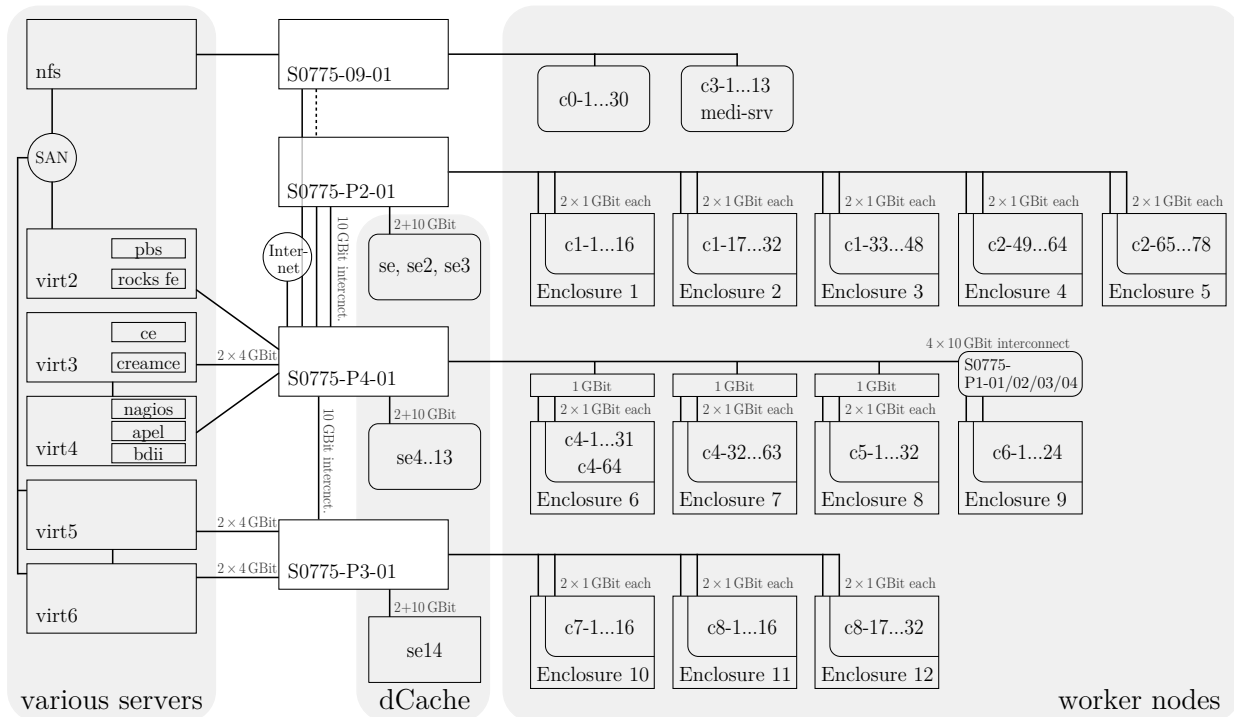


Figure 2.1.: The network topology of the GoeGrid cluster as of July 2011.

Count	Node Type	CPUs	Cores / Node	HEPSpec2006 Benchmark ^[27]	Peak Power [TFlop/s]
30	HP ProLiant DL140 G3	2 × X5355	8 × 2.66 GHz		2.55
78	HP ProLiant BL460c	2 × X5355	8 × 2.66 GHz		6.64
13	MEGWARE Woodcrest	2 × 5160	4 × 3.00 GHz		0.62
96	HP ProLiant BL2x220c G5	2 × E5440	8 × 2.83 GHz		8.69
16	HP ProLiant BL2x220c G6	2 × E5530	8 × 2.40 GHz		1.23
8	HP ProLiant BL460c G6	2 × X5650	12 × 2.66 GHz		1.02
48	DELL PowerEdge 11G M610	2 × X5650	12 × 2.66 GHz		6.13
289	Total Nodes			25,500	26.9

Table 2.1.: The different compute node types in GoeGrid and their combined computing power in July 2011^[26].

basically a combination of the resource manager TORQUE and the job scheduler MAUI. For maximal compatibility, two separate grid middlewares are used: GLITE is the official middleware utilized by the WLCG while GLOBUS TK is for the other communities^[34]. All these middlewares are compatible with the PBS system; thus, job submission to the cluster can be done in a number of ways, as indicated in Figure 2.2. Local users within the network can submit jobs directly to the cluster without the help of any grid services; users with an appropriate grid access can also submit their jobs via any of the supported

2. The Grid Resource Centre GoeGrid

middlewares.

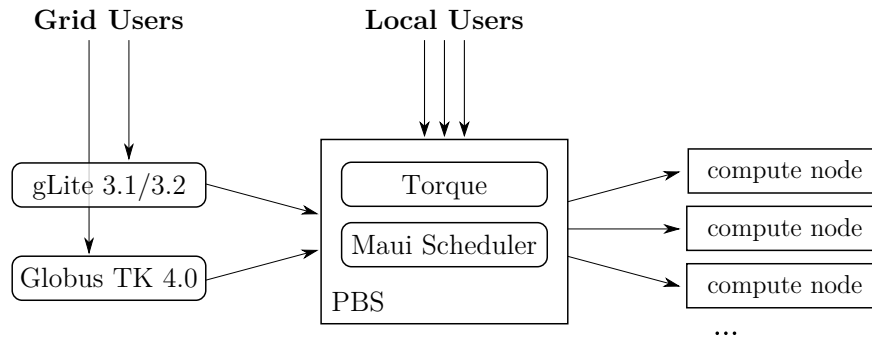


Figure 2.2.: Scheme of the job submission and distribution in GoeGrid^[34].

For the data storage, multiple approaches are taken by the individual communities as already indicated above. The largest disk space is however provided by the DCACHE^[28] system for the HEP (which is a data storage system widely used in the WLCG). Additionally, all communities can use the SAN storage service provided by the GWDG.

In addition to GANGLIA, the infrastructure monitoring tool NAGIOS^[35] monitors the servers and the services that they provide. On top of these systems, HAPPYFACE serves as a summarising meta-monitoring tool, as will be discussed in the next chapter.

3. The HappyFace Project

THE HAPPYFACE PROJECT^[36] (in the following just referred to as HappyFace) is an open-source framework for meta-monitoring of computing resources. It was initially designed to monitor WLCG sites, but can be used for arbitrary monitoring information. The first version of HappyFace was written in 2008 for the German WLCG Tier 1 centre. It became obsolete with the development of HappyFace 2.0 in 2009 which relied on a more sophisticated and generic approach.

This version has now – two years afterwards with continuous enhancements – proven to be a stable and useful tool which is gradually becoming adopted by more sites. Aside from the Karlsruhe Institute of Technology (KIT)^[37] and the University of Göttingen^[38], it is employed in the University of Hamburg^[39], the research centre DESY, and the university RWTH Aachen^[40]. The most recent version together with many available modules can be extracted from the central repository from the university of Karlsruhe; more information on the HappyFace installation can also be found on the development wiki page^[36].

3.1. Meta-Monitoring

As already pointed out, the management of a computer cluster or any other large network infrastructure requires the operation of monitoring software to be able to respond to upcoming problems quickly. In most cases, one will however realise that a single monitoring tool is not able to cover all monitoring aspects: different pieces of information and different technologies often require different monitoring software and for a lot of information sources there is no useful monitoring software available. Thus, it can be a very tedious task for system administrators to look through all available monitoring information. Not only that a general overview over all systems is not possible, some of the information sources also have long generation or loading times.

That is where, in addition to traditional monitoring tools, META-MONITORING systems such as HappyFace come into play. The idea of meta-monitoring is to collect information from different systems and other sources to display a summary of all available information. This allows the administrator to get a quick overview of the current situation and also

3. *The HappyFace Project*

provides several more advantages:

- The interface and the displaying of fetched information are easily customisable to match the individual needs of the monitoring subject. That way, only relevant information is displayed and it can be arranged intuitively.
- Besides the better overview, the meta-monitoring also allows to identify correlations between the different monitoring aspects which enables a deeper understanding of potential problems.
- Since the meta-monitoring system fetches information regularly, it is also able to provide figures about the trend and history of monitored variables; additionally, it can give valuable information when the cause of a problem is already in the past.
- If the system is set up intuitively, even people who do not have detailed knowledge about the cluster or access to all of its monitoring systems are able to determine the source of upcoming problems. This is especially useful for untrained personnel such as shifters, and so on.

Moreover, there are two different approaches for the information displaying: lightweight meta-monitoring systems only display short summaries and include hyperlinks to the source for extended information, whereas more sophisticated systems try to include all important data and build different views from the available input data for deeper analysis. Both concepts can be realised in HappyFace because of its modular approach.

3.2. HappyFace Concepts

HappyFace consists of a framework, the HAPPYCORE, which provides the basic functionality and a variety of modules that can be activated to retrieve information from external sources. This way, it is easy to customise the monitoring to match the specific needs. The core collects the output from all activated modules, arranges it in so-called categories to give a brief overview for different topics and publishes the results on a website. The advantage of a website is that it can be made accessible anywhere where internet is available or the access can be restricted to certain networks or browsers with correct authentication certificates.

One of the key aspects of the created website is simplicity. Every piece of information can be displayed within not more than three clicks and an intuitive rating system is introduced: every module can set its status to be either good, bad or something in between. The status is visualised with an icon (e. g. a smiley, which gives the project its

name). As these smiles are also determined as a summary of each category, an instant overview over the current cluster functionality is possible.

The details of the data flow in HappyFace are shown in Figure 3.1. The HappyFace framework launcher `run.py` is periodically executed, e.g. by a cronjob. It reads default and customized configuration files, the local configuration always having the highest priority, to determine the activated modules. The modules are implemented in python classes inheriting from `ModuleBase`. The classes are initialised and their data retrieval is launched concurrently in parallel threads. All activated modules then begin to read their configuration files and accordingly fetch information from external monitoring sources. They process the retrieved information and write it into an SQLite database including a timestamp. Furthermore, they return PHP code to the HappyFace framework that is able to interpret the saved database entries and to convert them into a human-readable output. All generated pieces of PHP code are then combined with code from the framework (which enables for example the category view and the history functionality) and the result is stored in a main PHP page which can be published by any webserver that supports PHP5. All other required files as theme graphics and module specific cascading stylesheets are also automatically stored in the same folder.

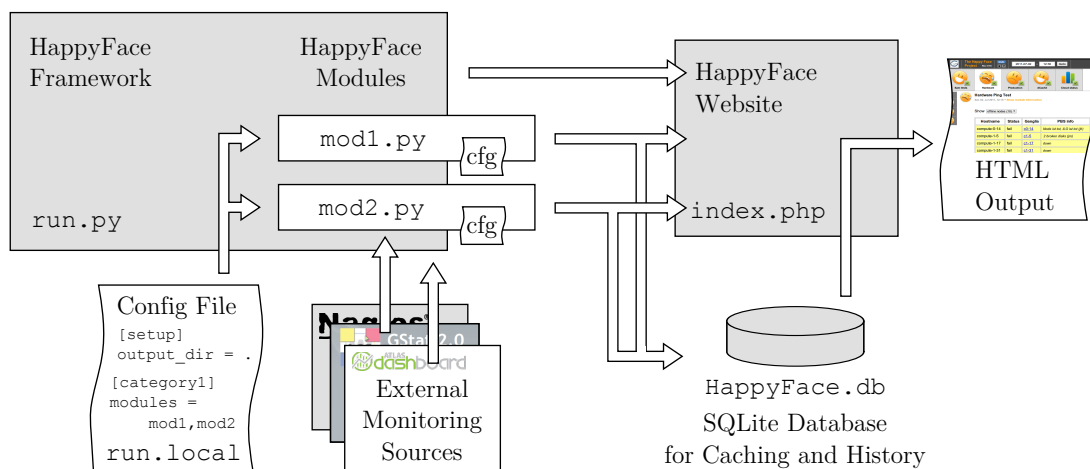


Figure 3.1.: Scheme of the data flow in HappyFace.

Another basic concept of HappyFace is the separation of the data retrieval from the visualisation of the data. The information is fetched via python and only minimally interpreted (for example to reduce its size) before it is stored into the database. The interpretation and visualisation then is handled by PHP to ensure that even old fetched data can be visualised according to the most recent rules. Smaller modules often disregard this rule and store plain HTML code in the database which is not advisable, since it causes the database to grow much faster and is a less generic approach.

3. The HappyFace Project

The last important principle of the framework mentioned here is the detachment of local configurations and modules from the standard ones delivered within the HappyFace package. This concept is reflected in the folder structure of a HappyFace installation: there is a dedicated folder `local` for all local configuration files and modules that do not belong to the HappyFace standard package. The additional modules are treated as if they were in the normal module folder and the local configuration files replace the default configurations.

Eventually, the resulting HappyFace websites and its components are shown in Figure 3.2. Most components are automatically generated by the HappyFace framework, the modules only have to create their output and set their status to be correctly included into the website.

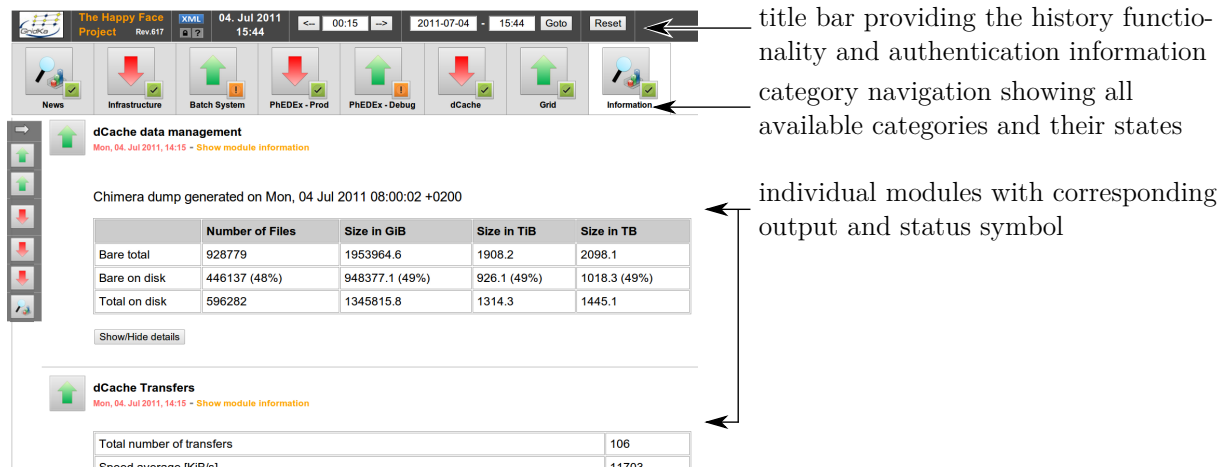


Figure 3.2.: The HappyFace instance monitoring the Tier 1 in Karlsruhe to illustrate the functionality of the HappyFace website^[37].

3.3. Recent Developments

In the course of this Bachelor's Thesis, several changes to the HappyFace framework have been applied. In the following section, some important modifications and additions will be documented. First of all, several layout fixes have been applied to supply the user interface with

- a quicker overview (by a clean layout)
- a more intuitive operation (by a consistent layout)
- a ubiquitary operation (by a compatible layout).

The outcome of this is that HappyFace is even usable with small screen devices – which enables system administrators to check their cluster status even with their internet-capable mobile phone.

3.3.1. Core Module HostProcessing

As a number of modules require lists of hosts, worker nodes or servers of the monitored cluster, the python module `HostProcessing.py` was added to the HappyFace framework. This module provides several useful functionalities in this domain: lists of hosts can easily be extracted from files with a common format which is described in the appendix. All hosts from a subset of defined groups (also called sections) are available via a single method.

Furthermore, the module offers classes to send ping packets to hosts asynchronously and to perform DNS lookups for given host names.

3.3.2. Core Module ModuleHelper

The recently introduced `ModuleHelper.py` provides a variety of helper functions that can make the development of many modules significantly easier while enhancing the information content and the layout of the modules' output. In order to take advantage of this functionality, the module classes only have to be derived from `ModuleHelper.py` in addition to the usual `ModuleBase.py`. Then, smaller helper functions for the simple output of user-friendly (error) messages and more complex tools such as automatic data source selection, customisable table creation and fast HTML table parsing are available.

The usage of the helper module will encourage generic programming approaches as e. g. the storage of properties of monitored entities in associative arrays and the storage of the resulting arrays for all entities once more in an associative array. Such programming concepts induce the creation of easily extendible and versatile code, which is a large advantage for HappyFace modules. Furthermore, the compliance to such schemes leads to the compatibility to many predefined functions that for example care for the database storage of such structures. For further illustration of the simplifications, an example in the appendix describes the methods of unpacking and displaying for such arrays in PHP.

There are even more things that the helper module is able to automatise like HTTP 1.1 downloads, HTML dropdown menus, and so on. Up until now, `ModuleHelper.py` has been kept separately to minimize interference, but it may be moved to `ModuleBase.py` in the future.

3. The HappyFace Project

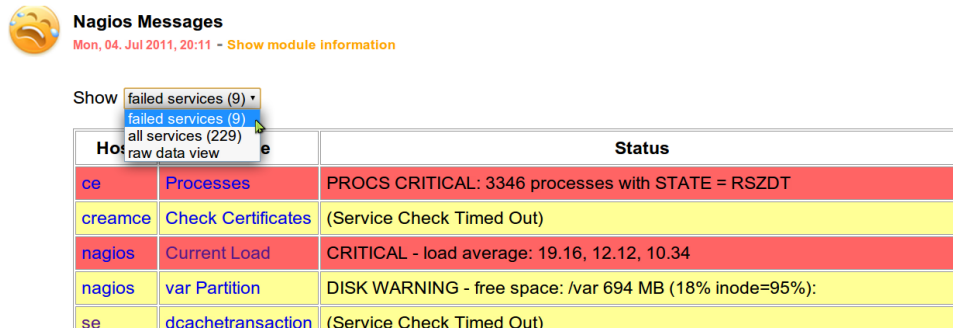


Figure 3.3.: Example of a HappyFace module that relies on ModuleHelper to produce a simple, easily customisable and informative output.

3.3.3. Prospect

The recent developments show that HappyFace is becoming a more and more generic tool for meta-monitoring. Once only used by one cluster in Karlsruhe, it is now employed by at least four WLCG sites and these sites have all begun to contribute their own modules to the central repository. Moreover, the development of modules is becoming simpler with lately introduced functionalities. The availability of more modules will increase the attractiveness of HappyFace in the near future so that it might be adopted by even more computer centres in the next years.

The increasing number of modules also results in a rising complexity for the configuration of HappyFace. It was proposed to create a full list of all available modules together with short descriptions or screenshots so as to counteract this. A similar approach would be the creation of a module that is activated by default and that generates this list automatically.

There are still many new ideas for the development of HappyFace. An important point here might be an increasing interactivensness of the HappyFace website as well as the creation of administration features – To achieve this, the HTTPS user certificate authentication could be used for a number of purposes, e. g. to launch scripts via button clicks on the website or to configure modules directly in the web. It was also suggested to use the `GOOGLE CHART TOOLS`^[41] for more of the generated plots and charts, since these HTML5 charts consume less disk space and provide an interactive feedback for the user.

Nevertheless, HappyFace has already become a stable and sophisticated tool that simplifies cluster maintenances enormously. Its easy setup and high usefulness combined with the universal approach could make it interesting for many computer centres – not only WLCG sites.

4. Application of HappyFace to GoeGrid

Since GoeGrid is a resource centre mainly involved as a Tier 2 in the WLCG, the idea of using HappyFace for meta-monitoring seems to suggest itself. Indeed, HappyFace has now been deployed for GoeGrid monitoring since more than a year – during this time, many standard HappyFace modules haven been integrated and an even higher amount of modules has been individually developed for the utilisation in GoeGrid. However, many of the latter are not local modules anymore given that they have already been published to the central HappyFace repository and therewith also have become a part of the HappyFace package.

The outcome of these efforts is shown in the screenshot of the main HappyFace instance of GoeGrid as in July 2011 in Figure 4.1 – the activated modules evidently cover a big part of all monitoring subjects in the GoeGrid cluster.

4.1. Setup

The setup of HappyFace including the correct meta-monitoring configuration for GoeGrid can be done in three simple steps: at first, the HappyFace prerequisites have to be fulfilled. Instructions for this can be found on the HappyFace developer website maintained at KIT^[36]. Basically, this first step consists of ensuring that Python 2.6, SQLite and a webserver supporting PHP5 are installed.

Secondly, the HappyFace framework and local configuration files have to be copied. For this purpose, a dedicated folder should be created, the HappyFace framework then can be extracted from the central SVN repository. The local configuration files for GoeGrid are stored in a different SVN repository managed by CERN. Two scripts that automatise the download process are presented in the appendix.

When the extraction of all necessary files has been successful, only the integration of HappyFace is missing: the correct configuration of the website publishing directory and a cronjob automatically launching HappyFace complete the installation.

4. Application of HappyFace to GoeGrid

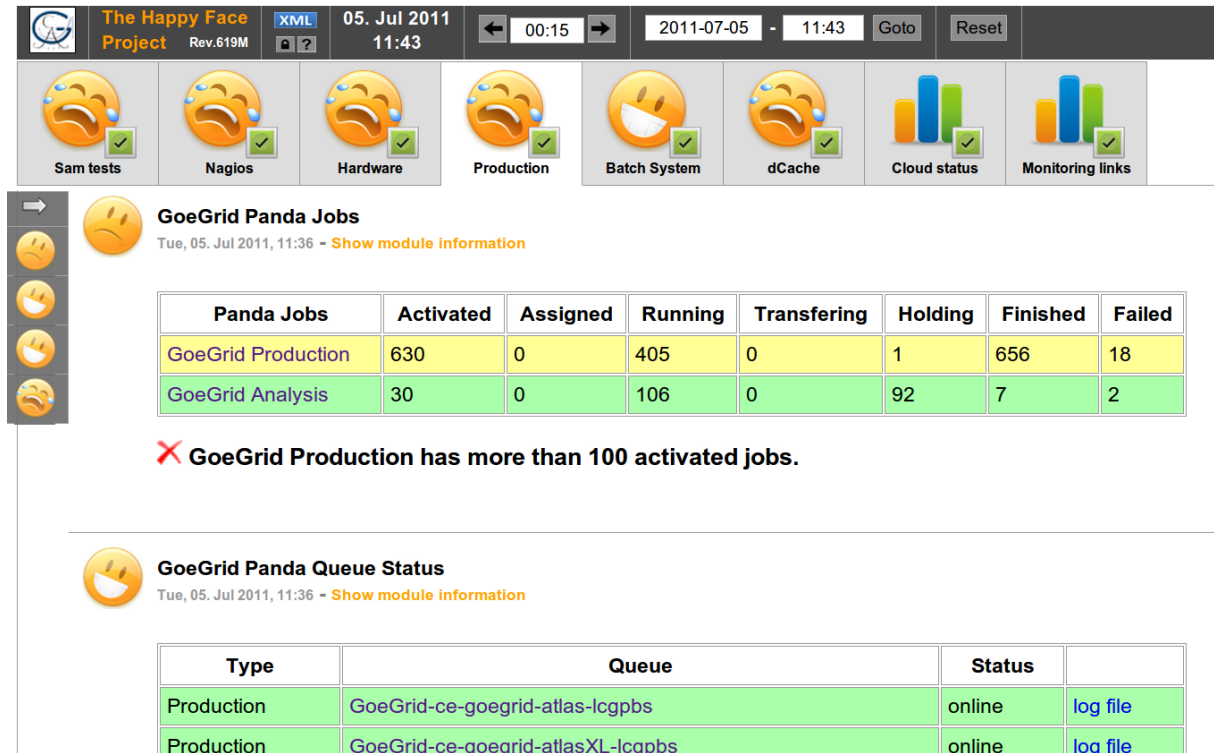


Figure 4.1.: Screenshot of the HappyFace instance that is employed for the GoeGrid^[38].

The two scripts in Listing A.5 and A.6 introduce a practical advantage for the process of updating: to bring a HappyFace instance to the most recent version, only both scripts have to be executed.

4.2. Modules from Central Repository

Currently, there are seven modules from the central repository utilized for GoeGrid meta-monitoring – most of them have once been developed in Göttingen and have been integrated in the HappyFace core package. The fact, that only very few modules created by other sites are in use is due to differences between GoeGrid and the other sites deploying HappyFace: GoeGrid is a Tier 2 centre dedicated to the ATLAS VO, this requires certain services that the other sites do not use. The other Tiers employing HappyFace in Aachen, Karlsruhe and Hamburg are mainly dedicated to the CMS experiment, which requires a different set of services.

The following list outlines the features of the modules from the central repository in GoeGrid in the order of appearance on the HappyFace webpage. For more information on

the recently revised or developed modules `NAGIOS_MESSAGES`, `HARDWARE_PING_TEST` and `ILO_QUERY`, see the next chapter.

GoeGrid SAM ATLAS Table This module was originally developed in Karlsruhe and has been configured to show the results of relevant `SERVICE AVAILABILITY MONITORING (SAM)`^[42] tests for GoeGrid. SAM is an external monitoring framework that automatically monitors gLite services from WLCG sites. The results of the SAM tests are published to the ATLAS SAM dashboard^[43], which is parsed by this module.

Nagios Messages This recently revised module was created in Göttingen to summarise the output of the widespread monitoring tool Nagios^[35]. It displays Nagios warnings and errors and changes its status accordingly.

Hardware Ping Test Because it generates its monitoring information on its own, rather than just reading it from external sources, this module is quite atypical: it sends ping packets to a given list of hosts to determine their online status. Also, it is optionally able to combine the retrieved information with data gathered from a PBS batch system.

Smartd Status The Smartd module is a typical application of the generic `LogMessages.py` module which is described in the next chapter. It is able to parse the log files from the `SMART-DAEMON` of Unix systems which provides information about the “health” of local hard drives.

ILO Query This module was created in Göttingen specifically for worker nodes from the manufacturer HP, which provide a configuration and maintenance interface called `INTEGRATED LIGHTS-OUT (ILO)`^[44]. This service is used to find failures of the HP hardware and to display the enclosure statuses.

dCacheInfoPool This is a module from Karlsruhe that displays general information about a specified dCache group. In Göttingen, it is used four times to monitor the dCache groups `atlas`, `cache`, `dteam` and `OPS`.

dCache Status of Space Tokens As dCache supports disk space reservations (via Space Tokens), this module is able to check the status and the usage of these reservations.

4.3. Local Modules

The twelve local modules for the monitoring of GoeGrid all are either too specific or not mature enough for the submission to the central repository; but in the future, some of them might also be submitted. The following list describes these modules and the next chapter gives more details on the utilization of the module PBS INFORMATION.

SAM Availability for GoeGrid This simple but useful module just displays a number of externally created monitoring diagrams. Yet, it is not very sophisticated, since it does neither download the graphics nor display the correct history.

SAM ATLAS This module displays the result for GoeGrid related SAM tests by querying the SAM database directly, differentiating between critical and non-critical tests.

GoeGrid Panda Jobs If the computing jobs are submitted via the PanDA^[45] ATLAS service to the sites, this module is able to track the counts of current jobs in the queues. It only needs a list of queue names to extract a number of relevant job statistics.

GoeGrid Panda Queue Status This is also a PanDA related module, it is able to check the status (online/offline) of a given set of PanDA queues.

DDM The local performance of the DISTRIBUTED DATA MANAGEMENT (DDM) of the VO ATLAS can be monitored with this HappyFace module. To achieve this, the information is extracted from the ATLAS Dashboard.

Apel Accounting for GoeGrid APEL, a specialized gLite service for the publishing of the local Monte Carlo simulation results and the associated accounting, is monitored by this module.

dCache Hotfile Detection This new module lists the most frequently accessed dCache files to become aware of unusual behaviours. The data is provided by a project on the dCache bottlenecks in GoeGrid^[46].

dCache Web The dCache monitoring web page provides a lot of useful information. The goal of this module is to make use of this, but up until now, it only evaluates the queueInfo subpage.

dCache Pool Availability Since occasionally single dCache pools tended to get stuck, this module was introduced to perform a simple dCache test on each pool and display

any failures. Thus, this module does not query an external monitoring source, but creates all information on its own, like the Hardware Ping Test module.

PBS Information For the monitoring of the distribution of current computing job distributions, two nearly identical modules have been created which follow the notion of other PBS modules from the central repository, since incompatibilities of these modules with the local installation emerged.

Cloudstatus This is a very simple module that just displays the content of the webpage of the CLOUD MONITORING^[47,48], correcting any contained local links. This module could be turned into a more generic module for the integration of arbitrary websites to become part of the HappyFace core.

Monitoring Links As the name indicates, this is not a real monitoring module, since it has only static content, it only displays a specified list of links. It might be possible to extend this module to provide a generic editable content.

4.4. Benefits

For the administration of GoeGrid, HappyFace has indeed proven to be a very useful tool. In many cases, upcoming problems were at first discovered by the administrators thanks to the monitoring information of HappyFace and could thus be resolved more quickly. In other cases, after the failures were identified, an appropriate HappyFace module was designed to report critical situations early. Furthermore, HappyFace helps for a deeper understanding of how the cluster works, how it behaves in special scenarios and what correlations do exist – the more modules provide internal information, the better will be the grasp of the cluster’s behaviour.

The configuration of HappyFace and the development of modules might both take some time, but the efforts pay off: with it, a sustainable maintenance could be realised enabling an easier cluster maintenance in the future.

5. Revised and New Modules

Within the course of this Bachelor's Thesis, all modules that are applied in GoeGrid have been revised and adjusted to the current cluster needs. In doing so, particular attention was paid to establish a consistent and clear layout in addition to the compliance with all HappyFace concepts. The revised modules were integrated into a fresh new HappyFace instance; this way, a completely up-to-date installation without inherited ballast was created.

In this chapter, some modules of interest that were completely rewritten or created from scratch will be highlighted to explain their possible field of application. Moreover, the integration of these modules in a HappyFace instance as well as possible extensions are discussed.

5.1. LogMessages

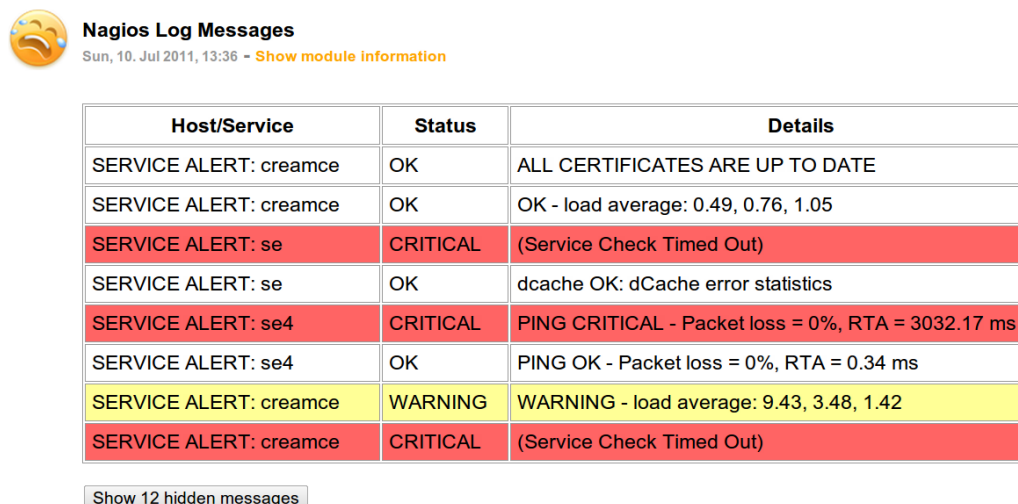


Figure 5.1.: A sample application for the generic LogMessages module: Nagios log file parsing.


The first depicted new HappyFace module is a rather simple one for introduction – nevertheless, LOGMESSAGES is a very generic module that can be employed for a large

5. Revised and New Modules

number of purposes: it is able to read, parse, and reformat arbitrary log files to display them in a well-arranged table showing only items of interest. The module is highly versatile since log files are an important part of virtually any service in large scale compute systems.

In Figure 5.1 an example for the application of LogMessages is depicted: the most recent log file entries of the monitoring system Nagios are parsed and displayed. For a better overview, irrelevant entries are sorted into a hidden table that is only displayed if required and more important entries are highlighted with colours. This module is just a sample, its short configuration can be found in the appendix for illustrational purposes. A more specialized and sophisticated Nagios module is discussed in the next section.

In GoeGrid, the module LogMessages has another application: it monitors the SMARTD log file entries of all nodes in the cluster; from those, warnings with corresponding host-names and times of last report are extracted and displayed as in Figure 5.2. The result is a comprehensive overview of all hard drive problems reported by the SMART-DAEMON.



Smartd Status
Sun, 10. Jul 2011, 14:48 - [Show module information](#)

Time	Host	Description
Jul 10 05:33:15	compute-3-13	Device: /dev/sda, 2 Currently unreadable (pending) sectors
Jul 10 14:33:14	compute-3-13	Device: /dev/sda, 1 Offline uncorrectable sectors
Jul 10 14:27:33	compute-5-5	Device: /dev/sda, 1 Currently unreadable (pending) sectors
Jul 10 13:09:17	compute-8-32	Unable to register ATA device /dev/sdb at line 3 of file /etc/smartd.conf
Jul 10 13:09:17	compute-8-32	Unable to register ATA device /dev/sda at line 2 of file /etc/smartd.conf
Jul 10 13:04:26	compute-8-32	Problem creating device name scan list

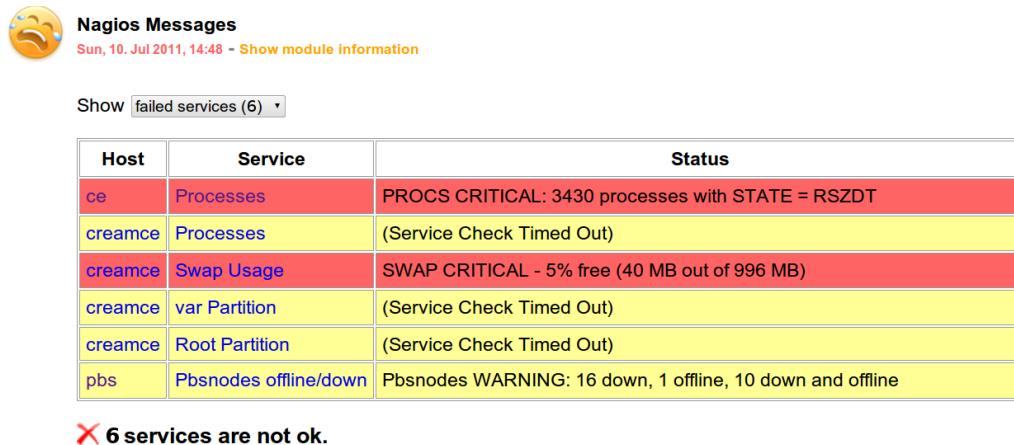
Figure 5.2.: Another application for LogMessages: SMARTD log file monitoring.

5.2. Nagios Messages

The infrastructure monitoring tool Nagios publishes its results in a more elaborate way than in its log files: the results are commonly accessed via an interactive monitoring website which provides – among other things – a full list of all checked services and their test outcomes. This list is in many cases reachable via a URL similar to `http://nagios_server/nagios/cgi-bin/status.cgi?host=all` after appropriate authentication.

The module NAGIOS MESSAGES is able to download such a webpage and extract the relevant table from it. This mechanism was tested for Nagios 3.2.1, but it is likely to work for other Nagios versions, too. If this succeeds, the table is parsed and all found

information is stored. The module then displays services with failures or warnings in a configurable table, classifying their criticality mostly according to the suggestion from the Nagios website. In Figure 5.3, such an output table is depicted, showing detected problems with the server `creamce`.



Nagios Messages
Sun, 10. Jul 2011, 14:48 - [Show module information](#)

Show

Host	Service	Status
ce	Processes	PROCS CRITICAL: 3430 processes with STATE = RSZDT
creamce	Processes	(Service Check Timed Out)
creamce	Swap Usage	SWAP CRITICAL - 5% free (40 MB out of 996 MB)
creamce	var Partition	(Service Check Timed Out)
creamce	Root Partition	(Service Check Timed Out)
pbs	Pbsnodes offline/down	Pbsnodes WARNING: 16 down, 1 offline, 10 down and offline

✘ 6 services are not ok.

Figure 5.3.: The output of the Nagios Messages module during CREAMCE problems.

For the utilization of this module, it is only required to specify the source of the Nagios information together with the user name and password combination. Obviously, NAGIOS MESSAGES is an easy to integrate and yet for many sites very useful module, since Nagios is such a broadly used and versatile monitoring tool. All of its most important results can then be accessed quickly through HappyFace.

5.3. Hardware Ping Test

This module is contrary to others rather complex, because it creates monitoring information on its own: it sends ping packets and issues DNS lookups for all hostnames in a specified host list. If a hostname cannot be resolved or if the pings to the respective host do not succeed, the host is classified as offline and displayed by the module in a configurable table as shown in Figure 5.4.

The retrieved information can also be utilized to build links to monitoring pages relevant for the single computers, e. g. their Ganglia page or the respective ILO (see next section) or enclosure node. An example for this is also depicted in columns three to five in Figure 5.4. If the cluster is managed by a PBS batch system, the module is furthermore able to take this as an additional input: the output of the command `pbsnodes` shows a table with annotations to the hosts' statuses. If available, this command can be set as an additional information source – then, all found annotations are stored and can also be

5. Revised and New Modules

displayed as visible in the last column.

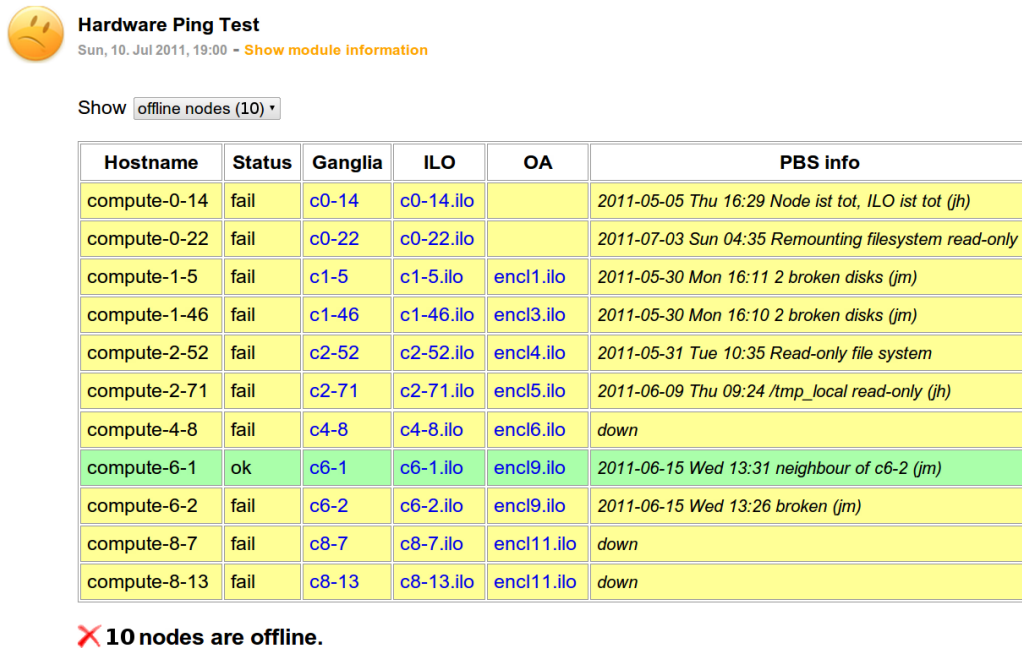


Figure 5.4.: The module Hardware Ping Test showing a list of 10 offline nodes.

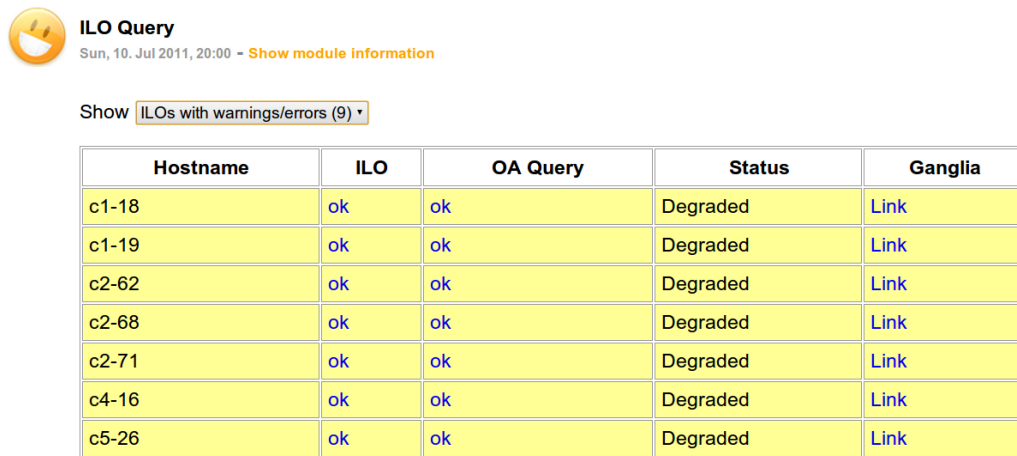
The integration of the ping test module is simple: only a list of all hosts that are to be tested is required, the PBS information is optional. In a second step, the links to the monitoring systems can be set up also via the configuration file. Since the host list may also contain multiple sections of hosts as described further in the appendix, it is also possible to classify several sections as critical hosts (e.g. servers). These hosts will be marked in red and trigger a bad module status if offline.

5.4. ILO Query

Compute nodes from the manufacturer HP feature a separate network interface for several maintenance and monitoring tasks with the name INTEGRATED LIGHTS-OUT (ILO). This network interface provides access to a website that not only publishes status information but also allows rebooting of the nodes and similar operations. Additionally, every node enclosure offers an ILO interface, where the status information of all contained nodes are summarized. The ILO QUERY module retrieves this data and displays all found irregularities.

As in the the Hardware Ping Test module, the displayed information can be configured to match the individual clusters needs. That way, links to other monitoring systems like

Ganglia can be inserted as illustrated in Figure 5.5.



The screenshot shows the 'ILO Query' module interface. At the top left is a smiley face icon. Below it, the text 'ILO Query' is displayed, followed by the date and time 'Sun, 10. Jul 2011, 20:00' and a link 'Show module information'. Below this is a 'Show' dropdown menu currently set to 'ILOs with warnings/errors (9)'. The main content is a table with the following data:

Hostname	ILO	OA Query	Status	Ganglia
c1-18	ok	ok	Degraded	Link
c1-19	ok	ok	Degraded	Link
c2-62	ok	ok	Degraded	Link
c2-68	ok	ok	Degraded	Link
c2-71	ok	ok	Degraded	Link
c4-16	ok	ok	Degraded	Link
c5-26	ok	ok	Degraded	Link

Figure 5.5.: Example output from the module ILO Query: the module has found a number of degraded compute nodes but no critical failures.

Other manufacturers also feature similar maintenance interfaces, DELL for example equips its servers with so-called DELL REMOTE ACCESS CARDS (DRAC)^[49]. It would be a helpful extension of the ILO Query module to be able to retrieve the relevant information from those different interfaces, to generate a vendor independent view of the current hardware status. However, first approaches in this direction showed that this is difficult for DRAC, since it requires login information.

5.5. PBS Information

The batch system PBS is able to export huge amounts of monitoring information, maybe most conveniently via the `qstat` command. Since PBS is a commonly used batch system, several modules for PBS monitoring were already available in the HappyFace core modules. However, these relied on a different PBS configuration and on python modules that were not available for the local HappyFace installation in GoeGrid. For these reasons, new modules with a similar approach were created for GoeGrid.

These modules parse the output of the command `qstat -f` and export the extracted relevant information as an XML file in the same format as modules from the HappyFace core do. The advantage of this intermediate format is the exchangeability of the information source – there also exist converters for other batch systems like LSF or CONDOR. The module PBS Information generates this XML file during the HappyFace initialisation phase, so that all other modules are able to access the extracted information afterwards. Since it uses a compatible format, the modules using this information from the Happy-

5. Revised and New Modules

Face core could be activated theoretically, but they need several yet unavailable Python packages.

From the generated data, graphics of the current job distributions per queue and per user group are created with the help of the Google Chart API^[41]. An example of these interactive diagrams is depicted in Figure 5.6. The diagrams show up until now only the job count distribution as well as the compute time distribution. The next step will be to select other interesting pieces of information to display from the comprehensive available XML data.

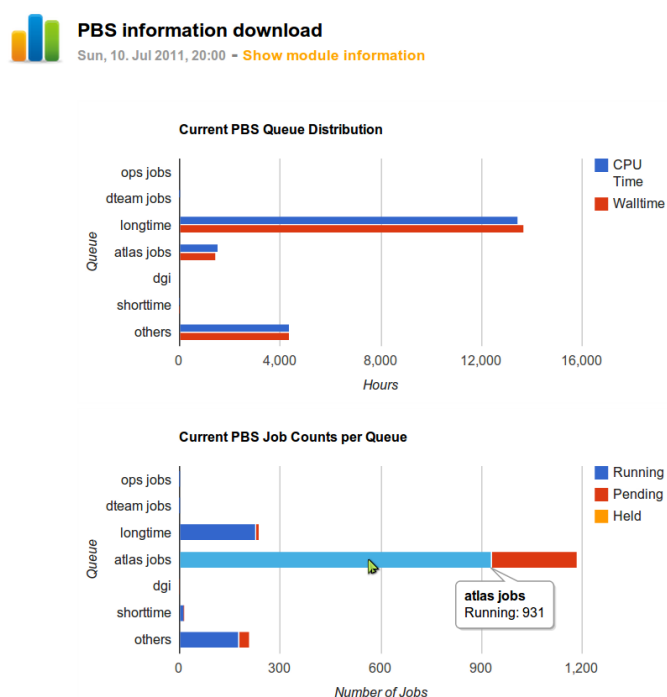


Figure 5.6.: The module PBS Information produces interactive diagrams about the current compute job distribution.

6. Summary

By mid of 2011, the four major experiments at the LHC have gathered a raw data output of about 20 PB which has to be stored and analysed to make it available to thousands of scientists around the world. Moreover, a large computing power is required to extract the particular information that confirm known physical concepts by precision measurements or lead to the discovery of new physical phenomena from this data. In order to supply these computing resources, more than hundred computer centres around the world have joined a grid to contribute to scientific advance – one of them is GoeGrid.

This German cluster has reached a peak computing power of 27 TFlop/s and provides more than half a petabyte of disk space. These significant resources cannot be maintained effectively without the help of several monitoring tools that continuously check the functionality of all components. To take this idea a step further, the meta-monitoring system HappyFace collects data from the available monitoring information sources and summarises the important aspects on a central website. The enhancement of this new approach was the goal of this Bachelor's Thesis. Thus, all local HappyFace modules have been revised and additionally some new ones have been developed. Moreover, a number of changes to the HappyFace framework itself have been applied and some generic local modules have been added to the HappyFace core modules.

6.1. Conclusion

The idea of meta-monitoring has indeed proven to be very useful for GoeGrid: the HappyFace website has become the central point for the discovery of emerging failures and in many cases even provides hints for the identification of the underlying problems. This may be due to the significant improvements of the HappyFace instance within the last time. The revised modules provide a clearer overview in a consistent layout and more useful information is displayed.

Furthermore, special attention was paid to make the modules very generic. Because of the interest of other HappyFace employing sites, these modules were added to the HappyFace package – these were the first modules developed in Göttingen mature enough

6. Summary

to be included in the central repository. As a result, not only GoeGrid, but also other sites can profit from their capabilities and the attractiveness of HappyFace for other computer centres rises.

The same is true for the enhancements of the HappyFace framework: since modules can now often be developed with less effort while producing a comprehensive and adaptable output and some general HappyFace issues were solved, more (grid) sites may consider trying to take advantage of the meta-monitoring framework – and more participants in the HappyFace project would also have a positive effect on the development of the framework.

Last, but not least, the application of HappyFace to the GoeGrid has the aspect of a sustainable maintenance effort: Whenever new, unexpected failures were encountered, a HappyFace module could be written that issues warnings to prevent the same problems from occurring again in the future. This way, the goal of a rising effectivity and reliability of the GoeGrid can be reached.

6.2. Prospect

It has to be emphasised that HappyFace has proven to be a stable and useful utility just as that there are still many ways to enhance and improve not only the local instance of HappyFace, but also the HappyFace framework in general. For the application to GoeGrid it seems to be obvious that there are nevertheless dozens of information sources left from which interesting monitoring aspects could be extracted. For example, the new PBS module provides a lot of information of which only the most important is displayed yet and many detailed statistics could still be evaluated.

Furthermore, the usability of HappyFace in general could be clearly improved by more interactive and administrative features on the website that could for example facilitate the setup and the configuration of HappyFace and its modules. For that purpose, the possibility of HTTPS authentication seems to have great potential.

In a nutshell, HappyFace may be adopted by more computer centres and become an even more versatile tool in the next years to help administrators to ensure the highest effectivity possible of their hardware and the provided services.

A. Excerpts and Examples

A.1. Host Lists

The syntax of host lists that are read by `HostProcessing.py` is illustrated by the file `hostlist.txt` that is used for GoeGrid and printed below in Listing A.1. These standardised files are separated into sections with arbitrary names to establish a logical grouping of the hosts. Comments are introduced with a `#` character, if the line does not begin with it, it is interpreted as a host definition. In addition to the hostname, its configuration hostname and its enclosure name can be appended separated by commas. Furthermore, if the hostnames consist of consecutive numbers, this can be subsumed with the `[a...b]` syntax which is extended to a list containing all numbers between `a` and `b`.

```
[server]
ce
nagios
rocks
pbs
apel
creamce
bdii

10 [virtual]
virt [1...4],      virt [1...4].ilo

[se]
se,               se.ilo
se [2...13],      se [2...13].ilo

[worker-hp]
# short name      config                enclosure
c0-[1...30],      c0-[1...30].ilo
20 c1-[1...16],      c1-[1...16].ilo,      encl1.ilo
c1-[17...32],     c1-[17...32].ilo,     encl2.ilo
c1-[33...48],     c1-[33...48].ilo,     encl3.ilo
c2-[49...64],     c2-[49...64].ilo,     encl4.ilo
c2-[65...78],     c2-[65...78].ilo,     encl5.ilo

[worker-meg]
```

A. Excerpts and Examples

```
c3-[1...13]

[worker-hp]
30 c4-[1...31],      c4-[1...64].ilo,      encl6.ilo
   c4-[31...63],   c4-[31...63].ilo,    encl7.ilo
   c4-64,          c4-64.ilo,          encl6.ilo
   c5-[1...32],   c5-[1...32].ilo,    encl8.ilo

   c6-[1...12],   c6-[1...12].ilo,    encl9.ilo
   c6-[17...28], c6-[17...28].ilo,  encl9.ilo

[worker-dell]
40 c7-[1...16],     c7-[1...16].ilo,    encl10.ilo
   c8-[1...17],   c8-[1...17].ilo,    encl11.ilo
   c8-[17...32],  c8-[17...32].ilo,  encl12.ilo
```

Listing A.1: hostlist.txt

A.2. Example for the Usage of ModuleHelper.py

An example module demonstrating many of the capabilities of `ModuleHelper.py` follows which extracts the weather forecast for the next five days from `http://www.weather.com`. The module consists of effectively less than 50 lines of code (not counting comments), but can display a nice table (which is configurable via the configuration file) summarising the weather forecast as shown in Figure A.1

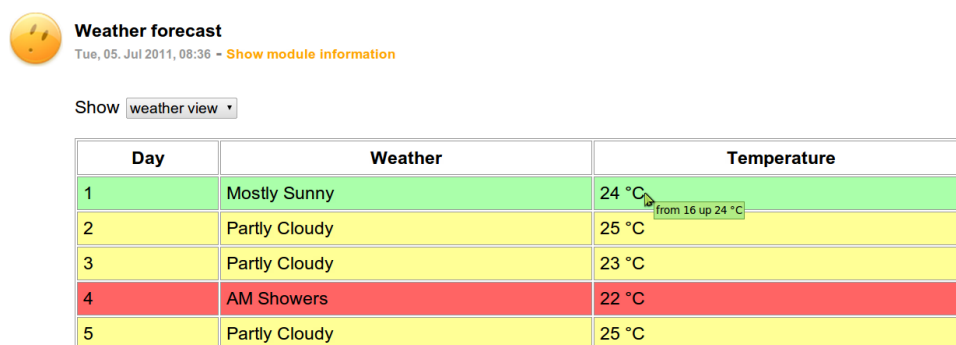


Figure A.1.: Output of a very short example module that displays the weather forecast.

This is achieved by retrieving the input data with `prepareInput` and `fetchInput`, the weather information is extracted with `SearchElement` and then the output is displayed with a number of PHP generating functions.

```
from ModuleBase import *
from ModuleHelper import *
```

A.2. Example for the Usage of ModuleHelper.py

```
class weather(ModuleBase, ModuleHelper):
    def __init__(self, module_options):
        ModuleBase.__init__(self, module_options)

        # prepare weather info source from configuration
        self.prepareInput('weather', 'source')

10

        # read display configuration from config file
        self.display = self.readTableConfig('display')

        self.db_keys['weather_info'] = StringCol()
        self.db_values['weather_info'] = ''

    def run(self):
        # fetch source data
        dat = self.fetchInput('weather', 'source')
20
        # and strip all line ends
        weatherinfo = '\n'.join(dat).replace('\n', '\n').replace('\r', '\n')

        # all weather info is in elements with tag name 'day'
        days = SearchElement(weatherinfo, 'day', '', '').results

        # we only need to fetch the arrays content now
        weather_info = {}
        i = 0
        for day in days:
30
            i += 1
            weather = {}
            weather['day'] = str(i)
            weather['info'] = SearchElement(day, 't', '', '').results[0]
            weather['t_low'] = SearchElement(day, 'low', '', '').results[0]
            weather['t_high'] = SearchElement(day, 'hi', '', '').results[0]
            weather_info[str(i)] = weather

        # pack retrieved information into db
        self.db_values['weather_info'] = self.packArray(weather_info)
40

    def output(self):
        module_content = """<?php

        // unpack stored information from db into var $weather_info
        """ + self.unpackArrayPHP('$data["weather_info"]', '$weather_info',
            '$keys') + """
        """ + self.sortTablePHP(self.display, '$weather_info') + """

        // create raw view from $weather_info into $div_raw
        """ + self.rawDataPHP('$weather_info', '$div_raw', '$keys', 'day') +
            """

50

        // create table
        """ + self.beginTablePHP(self.display, '$table') + """
```

A. Excerpts and Examples

```
foreach ($weather_info as $weather)
{
    // determine status of weather
    $tclass = 'warning';
    if((strpos(strtolower($weather['info']), 'showers')) != FALSE)
        $tclass = 'critical';
60    if((strpos(strtolower($weather['info']), 'sunny')) != FALSE)
        $tclass = 'ok';

    "" + self.addRowPHP(self.display, '$tclass', '$weather', '$table'
        ) + ""
}
"" + self.endTablePHP(self.display, '$table') + ""

$first = 'weather view'; $second = 'raw data view';
"" + self.showDivDropDownPHP(['weather_div_first', '$first', '$table'), ('weather_div_second', '$second', '$div_raw')]) + ""
?>""
return self.PHPOutput(module_content)
```

Listing A.2: weather.py

The code works with the configuration file below.

```
[setup]
mod_title      =      Weather forecast
mod_type       =      rated
weight         =      1.0
definition     =      Extracts the forecast from weather.com.
instruction     =

[weather]
sourceuse      =      download
10 sourcefile   =      /directory/to/file.txt
sourcedownload =      wget|html||http://xoap.weather.com/weather/local
    /GMXX0194?cc=* & link=xoap&par=1068094921&dayf=5&key=d7f30dc6bd9cb8f5&
    unit=m
sourcecommand  =      ['commandtogetweather', 'parameter']

[display]
# specify any columns to be shown
# column{num} =      {column title}[%nowrap%];{column content};{
    column hover text};{column link url}
column1        =      Day%nowrap%;%day%;;
column2        =      Weather%nowrap%;%info%;;
column3        =      Temperature;%t_high% C; from %t_low% up to %
    t_high% C;
20 # token that is used for sorting
sort           =      day(ASC)
```

Listing A.3: weather.cfg

A.3. Example for the Usage of LogMessages

The configuration file for the Nagios log file parsing HappyFace module shown in Figure 5.1 is listed with descriptive comments in Listing A.4. Obviously, for a well-arranged output only few lines are required.

```
[setup]
mod_title      = Nagios Log Messages
mod_type       = rated
weight        = 1.0
definition     = Displays the Nagios log files
instruction    = tail /var/log/nagios/nagios.log

[logfile]
# specify source of information: 'file' for a local file, 'download' for
10 # an http download and 'command' for the output of a command
loguse        = command

logfile       = /var/log/nagios/nagios.log
logdownload   = wget|html||http://servertoquery/nagioslog.php
logcommand    = ['tail', '/var/log/nagios/nagios.log', '-n 20']

[logrules]
# rules for classification in order of descending priority
# each rule consists of a pattern (*/? for arbitrary string/character)
20 # and a class, seperated by ";" - available classes in descending
# criticalness: critical, warning, ignore, hide, delete
rule1         = *SERVICE ALERT*WARNING*;warning
rule2         = *SERVICE ALERT*CRITICAL*;critical
rule3         = *SERVICE ALERT*OK*;ignore
rule4         = *;hide

[logdisplay]
# column specification in format: Title;Group
# a group is a either a list of source columns e.g. 1;2;3 or a regexp
30 # combined with the relevant bracket nums: ^(.{3});2 gives first 3 chars
column1       = Host/Service;^\[[0-9]*\]\s*([A-Za-z0-9\s:,. _-]+);2
column2       = Status;3
column3       = Details;6

# group for sorting and order ascending or descending
sort          = ^\[[0-9]*\]\s*([A-Za-z0-9\s:,. _-]+);2
sortorder     = ascending

# group that has to be unique to eliminate duplicates
40 unique      = 1;2;3;4

# this is the source column delimiter, it can be space, tab, ;, ...
columndelimiter = ;
```

Listing A.4: nagioslog.cfg

A.4. HappyFace Installation Scripts

For the installation as well as the updating of the GoeGrid HappyFace instance, two convenient scripts have been written to simplify the procedure. The first script downloads or updates the HappyFace framework, whereas the second script downloads or updates the local configuration files for GoeGrid from a different SVN repository managed by CERN.

```
svn co https://ekptrac.physik.uni-karlsruhe.de/public/HappyFace/trunk .
```

Listing A.5: update_happycore.sh

```
mkdir HappyFace
mkdir HappyFace/local
svn co https://svn.cern.ch/repos/atlasgrp/Institutes/Goettingen/HappyFace
/trunk HappyFace/local/

echo Please check HappyFace/local/cfg/run.local before running the new
HappyFace instance!
```

Listing A.6: update_happylocal.sh

B. Sources

Table 1.1: *Computer generated image of the whole ATLAS detector*, Joao Pequano
<http://cdsweb.cern.ch/record/1095924>

Table 1.2: World map image from *GStat 2.0 – Geo View*
<http://gstat-prod.cern.ch/gstat/geo/openlayers>

Bibliography

- [1] W. Armstrong, et al., *ATLAS: technical proposal for a general-purpose pp experiment at the large hadron collider at CERN*, CERN/LHCC pages 171–173 (1994)
- [2] D. Boussard, et al., *The Large Hadron Collider conceptual design*, CERN/AC (1995)
- [3] C. Lefevre, *LHC: the guide (english version)* (2008), <http://cdsweb.cern.ch/record/1092437/files/CERN-Brochure-2008-001-Eng.pdf>
- [4] C. Kesselman, I. Foster, *The grid: blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers (1998)
- [5] Z. Kunszt, S. Moretti, W. Stirling, *Higgs production at the LHC: an update on cross sections and branching ratios*, *Zeitschrift für Physik C Particles and Fields* **74**, 479 (1997)
- [6] LHC Higgs Cross Section Working Group, S. Dittmaier, C. Mariotti, G. Passarino, R. Tanaka (Eds.), *Handbook of LHC Higgs Cross Sections: 1. Inclusive Observables*, CERN-2011-002 (CERN, Geneva, 2011), 1101.0593
- [7] G. Aad, M. Ackers, F. Alberti, M. Aleppo, G. Alimonti, J. Alonso, E. Anderssen, A. Andreani, A. Andreazza, J. Arguin, et al., *ATLAS pixel detector electronics and sensors*, *Journal of Instrumentation* **3**, P07007 (2008)
- [8] A. Ahmad, Z. Albrechtskirchinger, P. Allport, et al., *The silicon microstrip sensors of the ATLAS semiconductor tracker*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **578(1)**, 98 (2007)
- [9] T. Åkesson, et al., *Status of design and construction of the Transition Radiation Tracker (TRT) for the ATLAS experiment at the LHC*, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* **522(1-2)**, 131 (2004)

Bibliography

- [10] N. Buchanan, et al., *Design and implementation of the Front End Board for the read-out of the ATLAS liquid argon calorimeters*, Journal of Instrumentation **3**, P03004 (2008)
- [11] T. T. C. G. of the ATLAS Collaboration, *The Production and Qualification of Scintillator Tiles for the ATLAS Hadronic Calorimeter*, Technical Report ATL-TILECAL-PUB-2007-010. ATL-COM-TILECAL-2007-026, CERN (2007)
- [12] K. Nagai, *Thin gap chambers in ATLAS*, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **384(1)**, 219 (1996)
- [13] A. D. Simone, *The calibration of the resistive plate chambers of ATLAS*, Journal of Physics: Conference Series **219(3)**, 032036 (2010), URL <http://stacks.iop.org/1742-6596/219/i=3/a=032036>
- [14] M. Livan, *Monitored drift tubes in ATLAS*, Nuclear Instruments and Methods in Physics Research-Section A Only **384(1)**, 214 (1997)
- [15] P. O'Connor, V. Gratchev, A. Kandasamy, V. Polychronakos, V. Tcherniatine, J. Parsons, W. Sippach, *Readout electronics for a High-Rate CSC detector*, in *Fifth Workshop on Electronics for LHC Experiments, Snowmass Colorado* (1999)
- [16] G. Brumfiel, *Down the petabyte highway*, Nature **469**, 282 (2011)
- [17] *WLCG – Grid computing*, URL <http://lcg.web.cern.ch/lcg/public/grid.htm>
- [18] I. Foster, *What is the Grid? A Three Point Checklist* (2002), URL <http://dlib.cs.odu.edu/WhatIsTheGrid.pdf>
- [19] *2011-2012 Confirmed C-RRB Resource Tables* (2010), URL http://lcg.web.cern.ch/lcg/Resources/WLCGResources-2010-2012_15DEC2010.pdf
- [20] *What is the Worldwide LHC Computing Grid?*, URL <http://lcg.web.cern.ch/lcg/public/overview.htm>
- [21] *GStat Geo View*, URL <http://gstat-prod.cern.ch/gstat/geo/openlayers>
- [22] H. Hegering, *D-Grid: Schritte zu einer nationalen e-Science-Initiative*, E-Science and Grid-Ad-hoc-Netze-Medienintegration **18**, 285 (2004)
- [23] *MediGRID Homepage*, URL <http://medigrid.de>

- [24] *TextGrid Homepage*, URL <http://textgrid.de>
- [25] *Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG)*, URL <http://gwdg.de>
- [26] C. Ay, J. Meyer, A. Quadt, C. Boehme, O. Haan, U. Schwardmann, *Das Göttinger Grid-Ressourcen-Zentrum GoeGrid*, Grid-Technologie in Göttingen **GWDG-Bericht Nr. 74**, 5 (2009)
- [27] *HEPSpec2006 Benchmark*, URL <https://twiki.cern.ch/twiki/bin/view/FIOgroup/TsiBenchHEPSPEC>
- [28] M. de Riese, P. Fuhrmann, T. Mkrtchyan, M. Ernst, A. Kulyavtsev, V. Podstavkov, M. Radicke, N. Sharma, D. L. and Timur Perelmutov, T. Hesselroth, G. Behrmann, T. Zangerl, P. Millar, O. Syngea, A. Petersen, *The dCache Book for 1.9.12-series* (2011), URL <http://www.dcache.org/manuals/Book-1.9.12/Book-a4.pdf>
- [29] J. Meyer, A. Quadt, P. Weber, *ATLAS Tier-2 at the Compute Resource Center GoeGrid in Göttingen*, in *Conference on Computing in High Energy and Nuclear Physics 2010, Taipei, Taiwan, 18-22 Oct 2010* (2010)
- [30] *CentOS Homepage*, URL <http://centos.org/>
- [31] *Scientific Linux at CERN*, URL <http://linux.web.cern.ch/>
- [32] *Rocks Cluster Distribution Homepage*, URL <http://www.rocksclusters.org/>
- [33] *Maui and Torque providing homepage*, URL <http://clusterresources.com>
- [34] U. Schwardmann, *GoeGrid – a resource center for grid related activities in Göttingen* (2009), URL <http://clusterday2011.aei.mpg.de/cgd2009/talks/schwardmann/GoeGrid-Cluster.pdf>
- [35] *Nagios – official provider homepage*, URL <http://nagios.com>
- [36] *The HappyFace Project*, URL <https://ekptrac.physik.uni-karlsruhe.de/trac/HappyFace>
- [37] *HappyFace instance for Karlsruhe*, URL <http://www-ekp.physik.uni-karlsruhe.de/~happyface/gridka/>
- [38] *HappyFace instance for Göttingen*, URL <http://happyface-goegrid.gwdg.de>

Bibliography

- [39] *HappyFace instance for Hamburg*, URL http://wwiexp.desy.de/groups/cms/tier2_monitoring/HappyFaceV2/trunk/webpage/index.php
- [40] *HappyFace instance for Aachen*, URL <http://grid-vo-cms.physik.rwth-aachen.de>
- [41] *Google Chart Tools API*, URL <http://code.google.com/apis/chart/>
- [42] *Service Availability Monitoring (SAM) Overview*, URL <https://twiki.cern.ch/twiki/bin/view/LCG/SAMOverview>
- [43] *ATLAS SAM Dashboard*, URL <http://dashb-atlas-sam.cern.ch/>
- [44] *HP Integrated Lights-Out (ILO) Advanced*, URL <http://h18013.www1.hp.com/products/servers/management/remotemgmt.html>
- [45] T. Maeno, *PanDA: distributed production and distributed analysis system for ATLAS*, Journal of Physics: Conference Series **119(6)** (2008), URL <http://stacks.iop.org/1742-6596/119/i=6/a=062036>
- [46] C. G. Wehrberger, *Hotfile- and Bottleneck-Recognition in the dCache system*, II.Physik-UniGö-BSc-2011/06, Georg-August University Göttingen (2011)
- [47] J. Kennedy, C. Serfon, G. Duckeck, R. Walker, A. Olszewski, S. Nderitu, the Atlas GridKa operations team operations team, *ATLAS computing operations within the GridKa Cloud*, Journal of Physics: Conference Series **219(7)**, 072039 (2010), URL <http://stacks.iop.org/1742-6596/219/i=7/a=072039>
- [48] *German Cloud Monitoring*, URL <http://happyface-goegrid.gwdg.de/cloudmon/cloudmon.html>
- [49] *DRAC: Dell Remote Access Card*, URL http://www.dell.com/content/topics/global.aspx/power/en/ps2q02_bell

Acknowledgements

Working on a thesis may in most cases at some times become exhausting – nevertheless, working on this particular topic was different due to the assistance of many people. It is a pleasure to thank you!

First of all, I would like to express my gratitude to Prof. Dr. Arnulf Quadt, not only because he found a Bachelor's Thesis perfectly tailored to my wishes but also for the welcoming atmosphere of the institute he established. Moreover, I would like to thank him and my second referee Prof. Dr. Johannes Haller for the efforts they put into the correction of this thesis.

I am deeply grateful for my supervisors Dr. Jörg Meyer and Dr. Pavel Weber who turned out to be the best supervisors imagineable – always available within minutes but also always understanding every stressful situation that I had. Thank you for breathing life into this thesis.

It is hard to overstate my gratitude to Christian Wehrberger – for all the help, all the cheering and for being the best apartment fellow possible. Finally, I want to express my profound relationship, friendship, and exaggerated gratitude to him, Miriam Reuter, Lucas Lang, Christina Heinicke, Friedrich Bös, and Thilo Müller von der Grün. We do what we must because we can.

Eigenständigkeitserklärung – Statement of Authorship

Erklärung nach §13(8) der Prüfungsordnung für den Bachelor-Studiengang Physik und den Master-Studiengang Physik an der Georg-August Universität Göttingen:

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbstständig verfasst habe, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe.

Darüberhinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, im Rahmen einer nichtbestandenenen Prüfung an dieser oder einer anderen Hochschule eingereicht wurde.

Declaration according to §13(8) of the Examination Regulations for the Bachelor's Degree and the Master's Degree of the Georg-August University Göttingen:

I declare that this Bachelor's Thesis has been composed by myself, unless otherwise acknowledged in the text. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Furthermore, I declare that this thesis has not been submitted in any previous unsuccessful application for a degree neither at this university nor at any other.

Göttingen, 13.07.2011

(Georg Jahn)