User- and Job-Centric Monitoring: Analysing and Presenting Large Amounts of Monitoring Data

Henrik Eichenhardt, Ralph Müller-Pfefferkorn, Reinhard Neumann, Thomas William
Technische Universität Dresden
D-01062 Dresden, Germany

Abstract

For data analysis or simulations (e.g. in particle physics) single users submit hundreds or thousands of jobs to the Grid. This puts a new burden on the users side - keeping an overview on the status and performance of the jobs in a distributed environment. In this paper, a user- and job-centric monitoring system is presented. It collects job and system specific information, analyses them and presents them graphically to the user. In the analysis of the monitoring data job problems are classified to give hints to the user. The graphical and interactive presentation allows the user to easily keep an overview as well as to dig into more detailed monitoring data.

1 Introduction

One of the most common scenarios of the use of Grid computing is the submission of a large number of jobs, e.g. for the analysis of large data sets, for large scale simulations or parameter scans. A typical example (and one of the major promoters of Grid computing) is particle physics. With the start of the Large Hadron Collider (LHC) at CERN in autumn 2008 dozens of petabyte of data need to be analysed by thousands of physicists. Furthermore, simulations will produce even more data to compare theory and experiments. Thus, the LHC Computing Grid (LCG) was setup to fulfil these needs.

To accomplish an analysis a single physicist will submit hundreds or thousands of jobs to the Grid - each job reading only a small part of the data. Managing such a large number of jobs is difficult - delayed or failed jobs slow down the work of the user. On a desktop machine the user is accustomed to have the ability to monitor the application, e.g. to use operating system tools (like top on Linux/Unix systems) to find out what the resource usage of the application is. In the LCG such tools did not exist or were of limited use (e.g. with textual output).

In this paper, the AMon monitoring system is presented which was designed and implemented not only to provide users with information, but support them in their daily work in the Grid. Section 2 shortly introduces AMon's architecture. Section 3 describes the analysis and classification of the monitoring data to find problematic jobs and section 4 shows the browser-based user interface and its capabilities to visualise the data and to allow an interactive monitoring by the user.

2 The AMon Architecture

The AMon monitoring system was designed and implemented in the High Energy Physics Community Grid project[1] (HEPCG) [7] of the German D-Grid initiative [5].

AMon consists of four components that are distributed in the Grid (see figure 2):

1. Data Collection: On the worker nodes - where the user application is running - a monitoring daemon is started in parallel to the job. It samples a variety of job- and system specific information in configurable time intervals of a few minutes (see table 1). Currently, these data are stored in R-GMA, the Relational Grid Monitoring Architecture [4] which is used in the LCG middleware gLite [1] to publish monitoring data. R-GMA is a kind of a distributed relational database that stores the data in predefined tables for a predefined duration. In general, AMon allows to use any other system to store and retrieve distributed monitoring data.
   The user can switch on and off the monitoring by simply setting an environment variable.

2. Data Gathering: A Web Service is used to gather the monitoring data. Currently, it is possible to access R-GMA or conventional relational databases (like MySQL, e.g. where local monitoring data are stored),

[1] funded by the German Federal Ministry of Education and Research (BMBF) under Grant No. 01AK802C

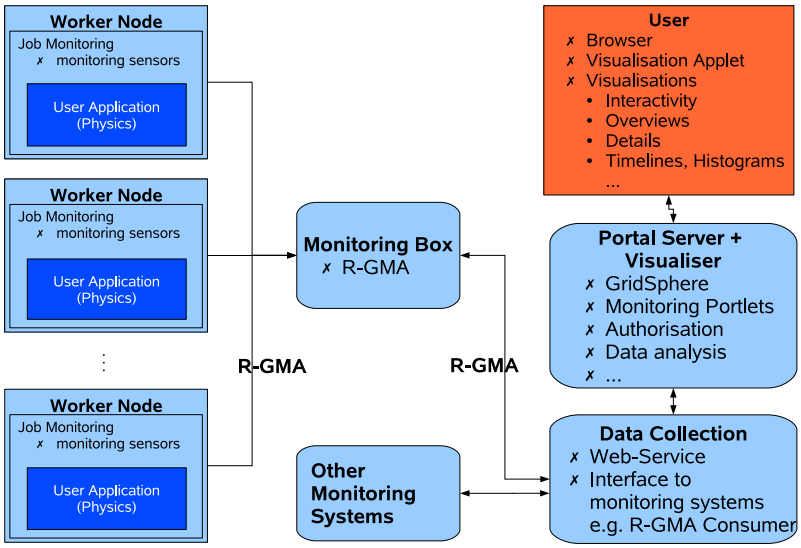

Figure 1. Architecture of the Job and Resource Usage Monitoring System AMon

but in general any monitoring system can be accessed here.

3. Data Analysis and User Interface: The entry point for the users of AMon is a Web portal (based on the portal technology Gridsphere [8]). Portlets (Java classes running in the portal) provide the logic and the look and feel to the user, e.g. asking the Web Service for data and analyse them (see section 3).

4. Visualisation: The preanalysed data can be viewed in Java applets that are embedded in the portlets. Java applets were chosen because they provide the possibility of interactivity with the user and sophisticated visualisation capabilities (see section 4).

AMon allows to persistently store the monitoring data from R-GMA into a relational database (e.g. MySQL). The intention is to allow site administrators to keep the data for debugging purposes.

The access to the data may be restricted by authorisation decisions. Currently, AMon can connect to VOMS, the virtual organisation management systen [2] that is used in LCG to obtain authorisation information for a user.

3 Characterising Job Problems

Though having access to extensive information about jobs is quite valuable for a user, it is of limited help in the case of large jobs sets of hundreds or thousands of jobs. Even with the graphical presentation of the mass of data, it is hard to identify problems. Thus, an analysis of the monitoring data is integrated in AMon that tries to give the users hints on possible problems. It can just give hints because it is not possible with the limited monitoring information available to absolutely classify a job as problematic or good. This depends on the job's intrinsic logic too. For example, an application that reads data in very small portions will show a low rate (MBytes/second) of disk access. Even though this might be normal for the application, from an outside (monitoring) view a low rate might also point to a problem with the disk or network hardware. Hence, only the user can decide - with supporting information from the monitoring system - if the state of a job is critical or not.

3.1 Critical Job States of a Single Job - Single-Job Analysis

To characterise the state of a single job for every measurement point, a set of filters were developed that analyse the monitoring information. Based on the available data (see table 1) new metrics are created that normalise the data to a starting point or calculate rates.

Currently, there are 9 filters that look at the following job activities and system states:

disk usage: critical amount of disk usage reached prone to cause a job abortion due to "disk full"

swap: performance due to insufficient memory and swapped out pages

I/O: performance of the disk access on the host

memory usage: critical amount of memory usage prone to cause swap activity

calculation: degree of CPU utilisation

Sensor group	*Description*
CPU activity	Utilisation of CPU(s): overall as well as divided into groups (user, nice, system, irq, iowait, idle)
CPU load	Load Averages of CPU(s)
Memory usage	Main memory, swap memory, resident and virtual set size of jobs
I/O	File centric monitoring of disk access [14]
Network	Transfer rates of network interface
Disk usage	Disk usage of specific directories and partitions
Time measurements	Runtime and utilised CPU time
Output	Output of user specified files (e.g. stdout, stderr)

Table 1. Summary of monitoring data collected

idle: job is not showing any progress

overload: CPU load on the host

OS activity: operation system activity on the host

network: network performance on the host

The filter evaluation consists of two steps. First, a filter/activity specific set of metrices is created and evaluated. Both, job- and system specific data are used. In the second step the influence of the found state on the job's performance is characterised by looking at job specific performance parameters, like the ratio of used CPU time and runtime. According to the result one of the three severity levels *low, medium* or *high* is set. In summary a state is characterised by a statement like "low job performance due to swap activity".

Currently, the filters are implemented in form of a disqualification procedure: Once a filter matches a measurement point, the evaluation is cancelled. Therefore, the filters have a hierarchy in which they are applied. In the next version of AMon this procedure will be succeeded by one that evaluates all filters and also defines one of three levels (low, medium or high) for the activity.

As an example the swap filter will be explained in the following. The main memory of a Linux system is divided into application memory, page cache, buffer cache and free memory. If there is not enough free memory available, the least recently used application pages are moved (swapped) to disk. Disk access is much slower than memory access. Therefore, if a process needs to access its data that were swapped to disk, the performance of the process can slow down significantly. Such a state of the system should be considered harmful.

The swap filter checks a number of process and system specific metrics to find such a critical state at a measurement point. At first, it looks at the state of the host machine: metrices on the load and memory usage - the ratio of user load and system load average normalised by the number of CPUs $\frac{L_{rel}}{L^{ave}_{n_{cpu}}}$, the overall system CPU load L_{rel}, the CPU load for I/O wait L^{rel}_{io}, the fraction of free main memory M^{rel}_{free} and the fraction of swapped out memory M^{rel}_{swap} are evaluated to match the host with swap activity.

To mark the influence of the swap activity on the job performance the process specific metrices: the ratio of the used CPU time of the process to the running time (CPU utilisation) $\frac{t_{cpu}}{t_{run}}$ and the fraction of main memory used by the process M^{rel}_{proc}). If they exceed predefined values the filter matches. Once the filter is matched it differentiates between three cases according to the job performance: low/medium/high (see table 2).

Figure 2 shows some of the filter variables for an example case. The application has a memory demand above the size of the main memory. Thus, the load of the CPU caused by both waiting for I/O and idling has increased significantly, while the CPU utilisation by the application process is very low. Together with an increased memory and swap usage (not shown) this points to the swapping problem.

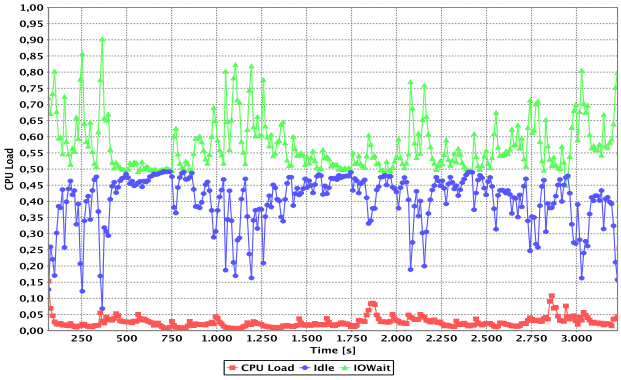

Figure 2. Overall CPU load (L_{rel} = CPULoad), load due to waiting for I/O (L^{rel}_{io} = IOWait) and idling (L^{rel}_{idle} = Idle) for an example application with a large memory demand

	Swap criteria					Performance	State
$\frac{L_{rel}}{LoadAvg}$	L_{rel}	$L_{rel,io}$	M^{rel}_{free}	M^{rel}_{swap}	M^{rel}_{proc}	$\frac{t_{cpu}}{t_{run}}$	
						< 0.2	low
< 0.8	< 0.8	> 0.1	< 0.05	> 0.1	> 0.05	0.2 - 0.8	medium
						> 0.8	high

Table 2. Filter criteria for the Swap Activity Filter

3.2 Comparing Job States - Multi-Job Analysis

Another possibility to find critical job conditions is the analysis of the evolution of the jobs and the distribution of the states identified with the filters. Depending on the severity level of the found states a job status can be defined as either *problematic* or *not problematic*. An example for a problematic state is swapping activity combined with a low job performance, while a high job performance with network activity should not be problematic. Looking at the distribution of these states one can extract more information. Several algorithms were investigated for AMon.

The most straightforward information to get is to take a look at the distribution of states in a single job. Taking the found states at all measurement points in time, a job can be presumed to be problematic if more than 20% of all its states are themselves problematic.

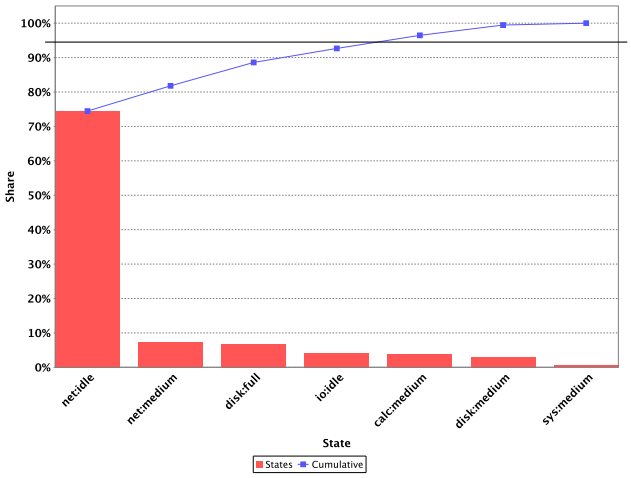

Figure 3. Example of the state distribution of a set of network intensive jobs - all states not included in the 95% integration are considered "rare"

Another algorithm takes a deeper look and compares the states of several jobs. This is only valid in the case of a set of similar jobs, e.g. analyses in particle physics that run the same algorithms on different chunks of data. The basic idea is that similar jobs are similar in their state distribution. Most of the time only a few jobs will have problems or will fail and thus will be observable as a deviation from the "standard" behaviour. Jobs that contain such "rare" problematic states will be marked as problematic. The algorithm finds rare states in the distribution of all states of all jobs that are seen as problematic. "Rare" is defined for a state if it is not included in the 95% confidence interval of the distribution (see figure 3).

A third algorithm looks at the state distribution of calculation intensive jobs. It takes the ratio of the CPU time used by the job to its runtime of the set of jobs and calculates average and standard deviation of that metric. Jobs with a ratio outside of two standard deviations will be selected as problematic.

Further studies on other algorithms are ongoing. Nevertheless, even these quite simple algorithms already have a large potential. First studies showed that they have the ability to distinguish critical and normal jobs. For these studies the old model of the filters was used, which applies the single job analysis filters (section 3.1) in a hierarchical way.

In the test scenario a number of jobs/applications were run each showing a "faulty" behaviour, e.g. large memory consumption or idling, as well as "normal" (calculation intensive) jobs. The jobs were started in a random order so that different combinations of jobs ran on the resources. The host was a cluster with Dual-CPU nodes. Table 3 summarises the results. It shows for every "faulty" job behaviour (job type) the fraction of states of all jobs selected and their share of problematic states as well as the fraction of states not selected (missed) and their share of missed problematic states. The miss rate of problematic states is very low (below 6% for all job types) denoting that most of the problems can be identified. But still there are some problems like high network or I/O activities which are not yet identified well by the filters. The last line shows that the analysis results of the "normal" jobs are affected if they run on machines where they share resources (e.g. multi-core machines which share the main memory). If "faulty" jobs run on the same machine, at about 5% of all measurement points problematic states were also assigned to the "nor-

Job Type	fraction of states selected	fraction of problematic states in selected	fraction of states missed	fraction of problematic states in missed
Idling	73%	80%	27%	1.8%
Memory intensive (more than the main memory available)	48%	50%	52%	3.8%
Overload	95%	82%	5%	0%
I/O intensive	20%	48%	80%	1.4%
System intensive	75%	80%	25%	3.7%
Network intensive	15%	60%	85%	5.2%
”Normal” (calculation)	10%	44%	90%	0.6%

Table 3. Overview of the selection results of the multi state analysis on synthetic jobs

mal” jobs. The reason is that on shared systems the system specific information can not distinguish between the single processes. On dedicated non-shared resources the ”normal” jobs are identified as ”not problematic”.

4 Visualisation of the Data

Another major focus of AMon is the visualisation of the monitoring data and the analysis results. In the case of large job sets only a graphical presentation of information is of use for the job submitter, both for getting an overview on the jobs and to examine detailed monitoring data for debugging purposes.

The visualisation is based on Java applets that run on the users desktop machine. They allow immediate interactivity with the user by clicking with the mouse into the displays. Compared with other dynamic web technologies (e.g. PHP or JSP [12]) that are server based the response to user interactions is much faster. Using applets also improves the scalability of the AMon system as data collection (Web Service), user interface (portal) and visualisation (applets) are separated.

The data are presented in four categories/views: job status information from the middleware, analysis results of the AMon filtering, detailed monitoring data, and user output. In the following some of the associated visualisations are described and shown.

Figure 4 presents the display of the status data from the gLite middleware. The user gets information whether the job is running, pending, aborted etc. The data are shown either graphical in a pie chart or as a list with the additional textual information on each job. The list can be sorted by any column.

The analysis results of the monitoring data (section 3.1) are presented in a display depicted in figure 5. In each row the filter results of the different activities are shown, the severity marked with the traffic light colours. Additionally, hints are given as text. The ”***” columns weights the filter results (severity levels) to create a kind of a summary. Again, the list can be sorted by any column.

The data display (figures 6 and 7) allows the user to go into detail. The timeline shows monitoring data in their temporal development, both as colour coded displays for all jobs or as drawing for a single job. The user can select single jobs and metrices by simple mouse clicks. It is possible to zoom into the data views (by choosing a range with the mouse) or to switch between a view of the data versus the runtime or the measurement time of the application(s).

When switching on the monitoring, the user can define up to five output files of his application to be observed. At every measurement point AMon catches the last thousand characters of these files, which the user then can view in the output display (figure 8).

5 Related Work

Most existing monitoring tools in Grid computing focus on the monitoring of resources and services. Examples are GridICE [3], GStat [13] or the LCG Real Time Monitor [6]. Only very few collect job specific information and provide a user view. G-Monitor [9] retrieves its information only from the resource broker, and thus is restricted to data from the submission process. OCM-G [10] is a Grid job monitoring tool that focuses on parallel jobs which are distributed across multiple sites. It contains an infrastructure to collect data from different processes and supports monitoring of the environment and performance monitoring.

6 Conclusions and Future Work

For the submission of large job sets to the Grid it is necessary to support the users in keeping track on the status of their jobs and provide them with sufficient information on the job conditions. The job- and user-centric monitoring

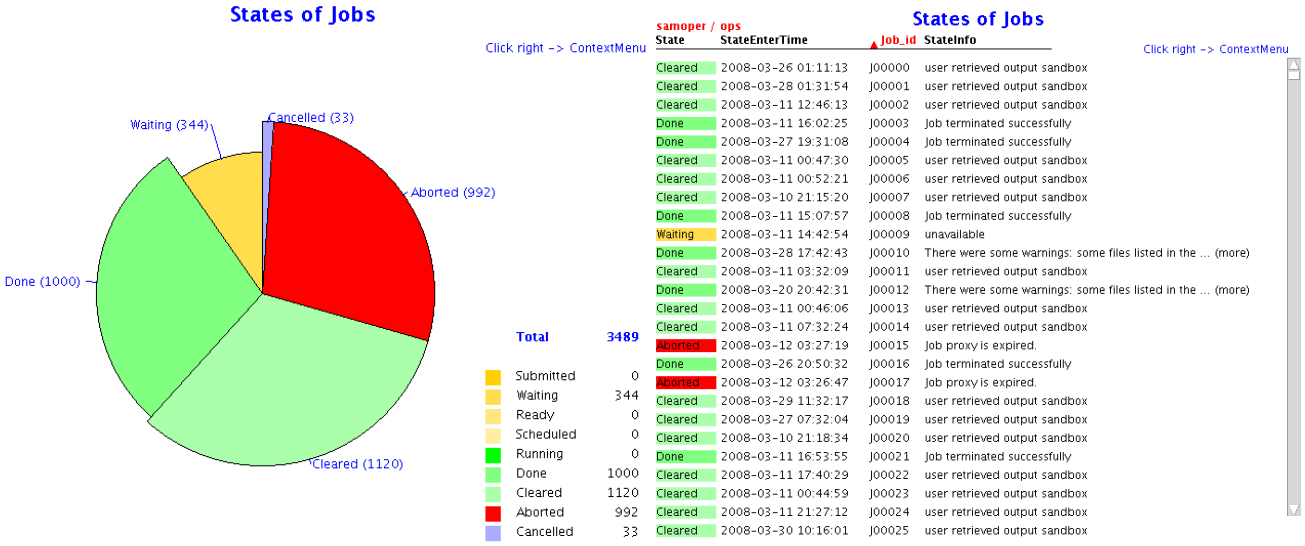

Figure 4. Status Display - status information from the gLite middleware: left - pie chart, right - list with detailed state information

system AMon collects such information and presents them graphically with an easy to use interface. Furthermore, it analyses the data to give hints to the user about possible problems in the job execution.

In the future, the analysis algorithms will be enhanced to improve the problem search and identification. For data collection AMon will be integrated with Globus 4 [11] to use MDS for information storage and retrieval and to provide job-centric monitoring information in Globus 4 environments. The visualisation part will be extended to refine the presentation, for example to present the performance information of the I/O monitoring [14].

References

[1] gLite - Lightweight Middleware for Grid Computing. http://www.glite.org (visited 2008).

[2] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, K. Lőrentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Gener. Comput. Syst.*, 21(4):549–558, 2005.

[3] S. Andreozzi, N. D. Bortoli, S. Fantinel, A. Ghiselli, G. L. Rubini, G. Tortone, and M. C. Vistoli. Gridice: a monitoring service for grid systems. *Future Gener. Comput. Syst.*, 21(4):559–571, 2005.

[4] P. Bhatti, A. Duncan, S. M. Fisher, M. Jiang, A. O. Kuseju, A.Paventhan, and A. J. Wilson. Building a robust distributed system: some lessons from R-GMA. In *CHEP'07*, Sep 2007.

[5] D-Grid. The D-Grid Initiative. http://www.d-grid.de (visited 2008).

[6] GridPP. the LCG Real Time Monitor. http://gridportal.hep.ph.ic.ac.uk/rtm/ (visited 2008).

[7] HEPCG. High Energy Physics Community Grid. http://www.hepcg.org (visited 2008).

[8] J. Novotny, M. Russell, and O. Wehrens. Gridsphere: a portal framework for building collaborations: Research articles. *Concurr. Comput. : Pract. Exper.*, 16(5):503–513, 2004.

[9] M. Placek and R. Buyya. G-Monitor: A web portal for monitoring and steering application execution on global grids. In *CLADE '03: Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments*, page 10, Washington, DC, USA, 2003. IEEE Computer Society.

[10] K. Rycerz, B. Balis, R. Szymacha, M. Bubak, and P. Sloot. Monitoring of hla grid application federates with ocm-g. In *DS-RT '04: Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 125–132, Washington, DC, USA, 2004. IEEE Computer Society.

[11] J. M. Schopf, M. D'Arcy, N. Miller, L. Pearlman, I. Foster, and C. Kesselman. Monitoring and discovery in a web services framework: Functionality and performance of the Globus toolkits MDS4. *Technical Report*, ANL/MCS-P1248-0405, April 2005.

[12] Sun Microsystems. Java Server Pages Technology. http://java.sun.com/products/jsp/ (visited 2008).

[13] The GStat team. Grid statistics (gstat) description. http://goc.grid.sinica.edu.tw/gstat/ (visited 2008).

[14] T. William. *Monitoring von Dateizugriffen in einer Grid-Umgebung*. Number ZIH-R-0703. Technische Universität Dresden, June 2007.

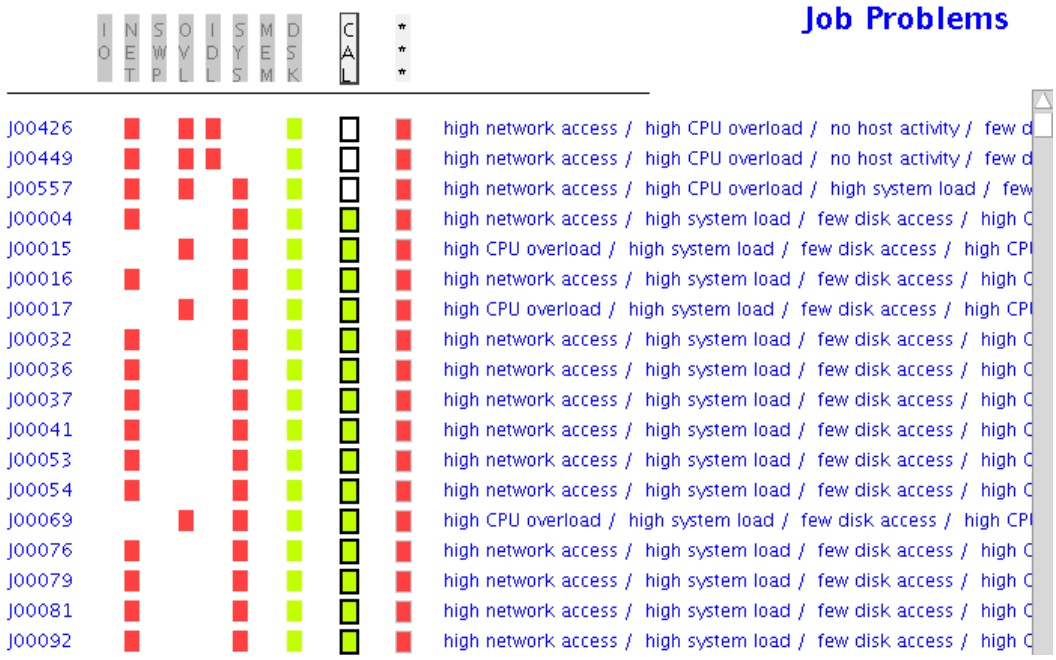

Figure 5. Filter Display - results of the single job analysis: the severities of the filter states are denoted in traffic light colours and are combined with additional textual explanations

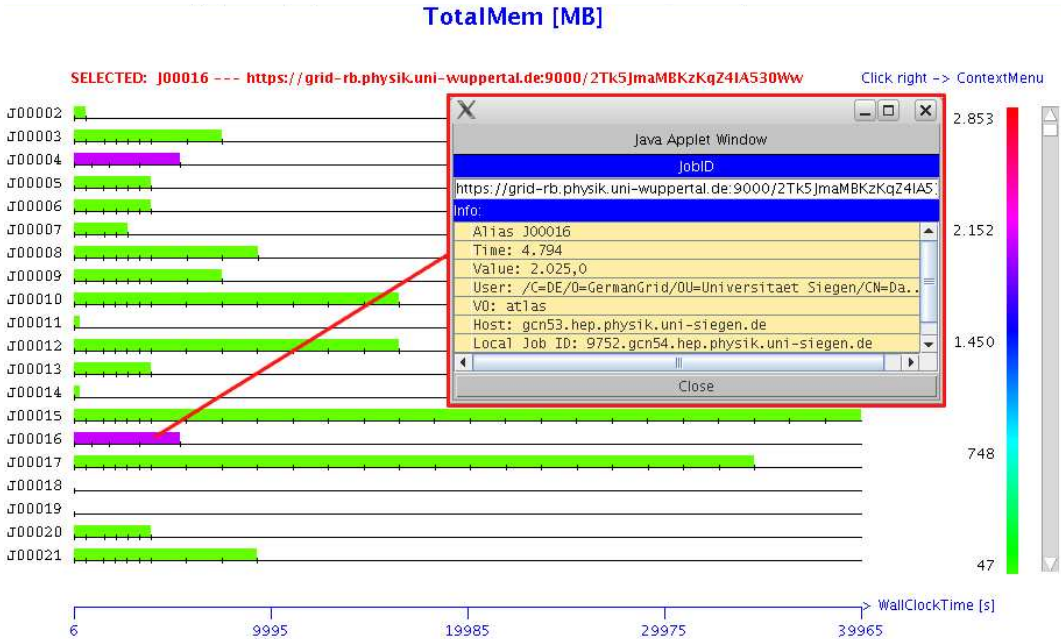

Figure 6. Data Display - colour coded timeline of monitoring data for all jobs (here: real memory usage versus the runtime), the red line denotes that clicking into the display will get a detail information

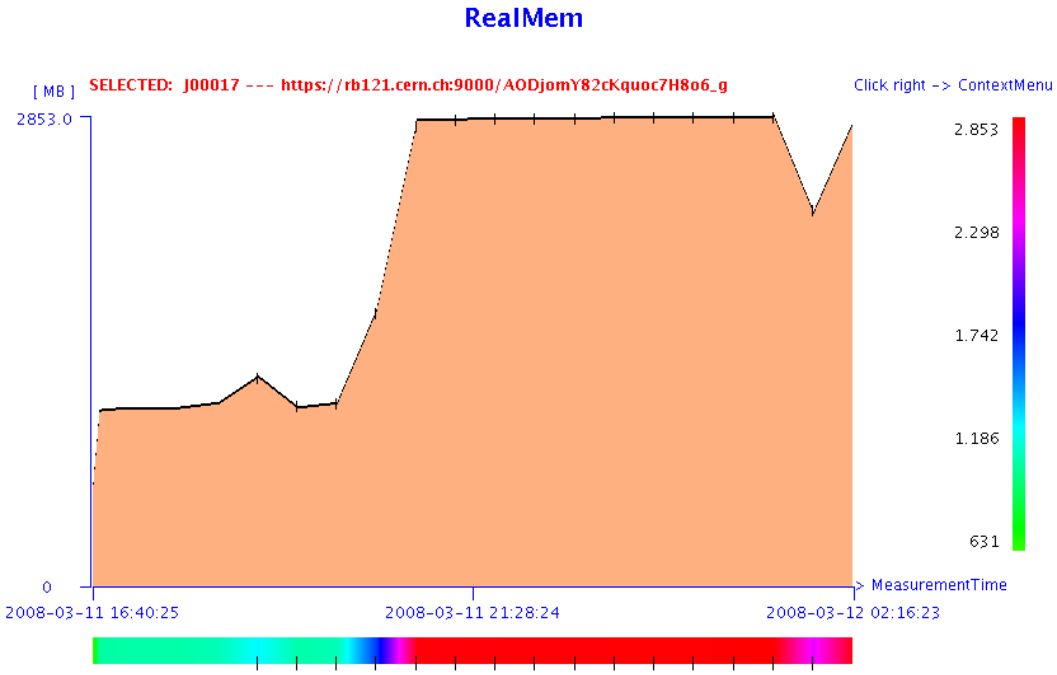

Figure 7. Data Display - data (memory usage) of a single job versus the measurement time

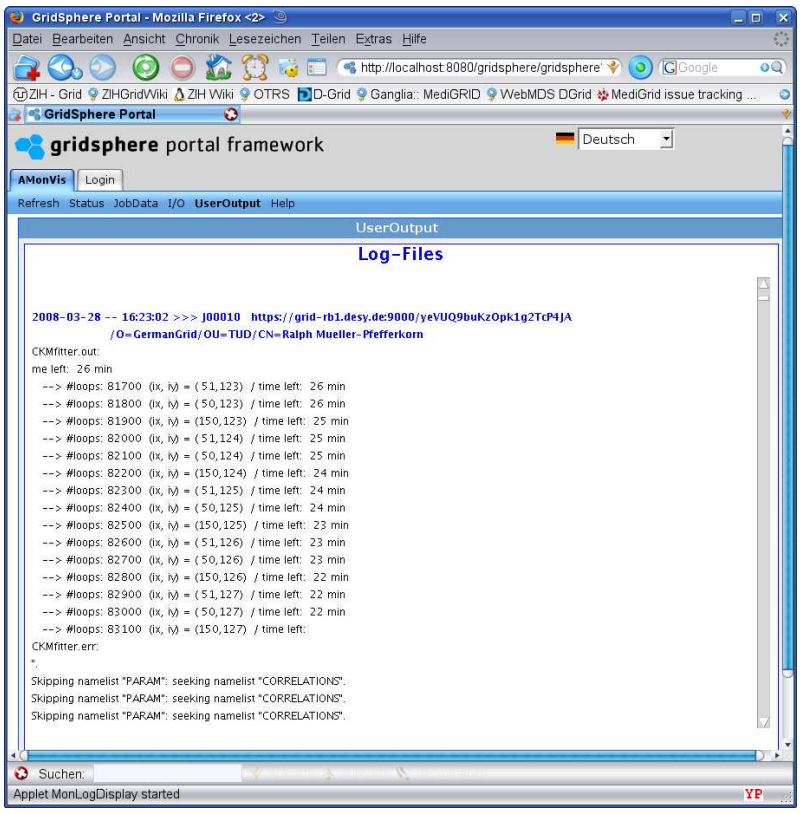

Figure 8. Screenshot of the browser with the Gridsphere based portal and the display with the output of user defined files