Achieving scalability for job centric monitoring in a distributed infrastructure

Marcus Hilbrich, Ralph Müller-Pfefferkorn

Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden
01062 Dresden
Germany
marcus.hilbrich@tu-dresden.de
ralph.mueller-pfefferkorn@tu-dresden.de

Abstract: Job centric monitoring allows to observe jobs on remote computing resources. It may offer visualisation of recorded monitoring data and helps to find faulty or misbehaving jobs. If installations like grids or clouds are observed monitoring data of many thousands of jobs have to be handled.

The challenge of job centric monitoring infrastructures is to store, search and access data collected in huge installations like grids or clouds. We take this challenge with a distributed layer based architecture which provides a uniform view to all monitoring data. The concept of this infrastructure called SLAte and an analysis of the scalability is provided in this paper.

1 Introduction

Direct observability of computing tasks is more and more lost by using external, distributed resources for getting calculations done. Job centric monitoring is a service which takes the challenge to fill the gap between using external resources and direct observability of jobs. Therefore monitoring data of each job are recorded for future analysis and visualisation. Job centric monitoring gives detailed information about used resources of currently running jobs where grid middlewares or batch systems give just the fact that the jobs are running. Using detailed information enables us to analyse the behaviour of already finished jobs instead of just showing the exit state. So the reason of an aborted job can be analysed and found. This can be an unexpected behaviour of the own job or that ensured computing resources are not available due to hardware problems or misbehaving users. A comparison of jobs is also offered to detect unusual jobs even if they are not aborted.

Job centric monitoring is most benefiting for environments with large numbers of jobs and resources which are shared by various users. In these systems, users can not easily observe jobs like on local resources by using desktop monitoring systems. Resource monitoring does not solve this problem because it does not provide job observability. Observed are the characteristics of resource usage, not the impact of single users or specific jobs.

The architectures we have in focus for job centric monitoring are grids (offering huge amounts of heterogeneous resources), clouds (offering resources on demand) and HPC or cluster systems operated by different computing centres. Currently the D-Grid[1] infrastructure is used for research.

One of the challenges in this field of research is to develop visualisation systems which offer methods to handle thousands of jobs running on different hardware with changing side effects e.g., the influence of jobs of other users utilising the same computing systems. But before we can analyse the data we have to handle them. To get an idea which amount of monitoring data we have to handle we consider an example: A single measurement consists of $15\,kB$ of data[2] (e.g., user name, grid VO, timestamp, CPU time, load average, used and free memory, used and free swap space, IRQ load, job ID, host name). Assuming that a single computing system or resource provider offers capacity for 5000 jobs and a measurement is done every minute ($60\,s$), this results in a network load of $1.25\,MB/s$. Considering only ten of these systems brings us to $12.5\,MB/s$ or 833 packages of $15\,kB$ each per second only for storing these data! This huge number of small packages is easier to handle in a distributed infrastructure instead of a central storage system.

Our answer to the challenge of storing, accessing and searching huge amounts of job centric monitoring data is the infrastructure SLAte [HMP10]. It is build on a concept of distributed servers organised in three layers. These layers allow to distinguish between different network capabilities (within a site[3] and between different sites) and to build a monitoring infrastructure adapted to the network. The performance of each of the three layers can be increased by installing additional servers. Thus the performance of the monitoring infrastructure can be adapted to user needs.

This paper describes the SLAte architecture, major implementation details and related work. Based on the architecture it is demonstrated how the performance of the job centric monitoring infrastructure can be increased to handle the increasing amount of data caused by additional computing resources and users. Afterwards scalability issues and potential bottlenecks are analysed.

2 Related Work

A lot of scientific work has been done on monitoring in general and similar fields of research. These topics are:

Monitoring on local computing resources: For local monitoring command line tools like ps, top or free[4] or graphical ones like Gnome System Monitor[5] can be used. On clusters Ganglia[6] gives information about the utilisation of resources. The im-

[1] `http://www.dgrid.de/`
[2] About $15\,kB$ per measurement is what we measured on our current installations.
[3] A site is a set of resources of a resource provider at a single location.
[4] `http://procps.sourceforge.net/index.html`
[5] `http://library.gnome.org/users/gnome-system-monitor/stable/index.html`
[6] `http://ganglia.info/`

pact of a specific user or job can not be identified directly by these tools.

Tracing and profiling: Profiling e.g., done with GNU gprof[7] gives information which functions of a program are used and how long they are used. This information is often presented as statistical evaluation. Even more detailed information are given by tracing tools like Vampir [BHJR10]. These analyses often tend to significantly slowdown the application. Thus they are not valuable for monitoring many jobs at once. Moreover the infrastructure to handle profiling or tracing data is designed for one job and in most cases the transfer of data over a wide area network is not needed. Thus it can not be easily adapted for job centric monitoring.

Accounting: Accounting is used for billing the use of resources and discovering the utilisation of computing systems. Examples are SGAS [EGM+] and DGAS [PRAGA09]. Based on the fact that only basic information of a job have to be recorded the amount of data to be handled is low. Thus there is no demand on a scalable infrastructure which is suitable for job centric monitoring if a central database is able to handle all accounting data. An other aspect of accounting is a high demand on reliability and methods to verify information while job centric monitoring looks more on scalability and performance.

Logging: Logging means to record events relevant for the reconstruction of the life cycle of a job. These events are e.g., the start, exit and abort of a job like recorded with accounting systems. Another source of relevant events is the program itself. It can report which subtask is executed or if problems occur. Such problems are e.g., missing rights to read or write files, wrong configurations and missing input data. The information is strongly bound to the context of the program and logging has to be intended by the program developers. Thus, not each job can deliver logging data. The logging information is often written to so called log-files or recorded by syslog daemons implementing rfc 5424[8]. An centralised implementation of logging grid jobs is shown by [RSK+08].

Resource monitoring: The task of grid resource monitoring is to record information about grid computers. Collected are information about hardware, used middlewares, offered services, known outages, planed maintenances and utilisation or free resources. This allows to create statistics of reliability and utilisation of resources and services or to assign jobs to handy resources. Examples of this kind of projects are D-Mon [BBK+09], CMS Dashboard [ACC+10] and Ganga [VBC+10]. A lot of the information collected by these tools are static and have no need for a high frequency of updates. Thus all resource monitoring data generated on a huge system like grid are handled by a central database.

The specific needs for handling job centric monitoring data are not properly considered by any of these tools or fields of research. Thus, we developed new concepts for job centric monitoring which are explained in the following sections.

[7] `http://sourceware.org/binutils/docs/gprof/`
[8] `http://tools.ietf.org/html/rfc5424`

3 Scalable, Layered Architecture – SLAte

3.1 Architecture

The architecture of SLAte is designed in a way that installing additional servers increases the performance of handling monitoring data. This is needed if more users want to access monitoring data or new computing resources should be used. As previously introduced the network capacity may be a limiting factor when observing many jobs in SLAte the overall bandwidth can be adjusted. For easy access to the distributed monitoring data we provide an unified and global view.

To get the idea of a scalable monitoring infrastructure to reality, a layer based concept is realised. It consists of three layers schematically shown in Figure 1. The Short Time Storage (STS) layer gets the data from the compute nodes and stores them temporarily. The Long Time Storage (LTS) layer accumulates the data from the STS servers and stores them persistently and distributed over multiple servers. The outer layer is the Meta Data Service (MDS) which provides the global view of the data. In the following these layers are described in more detail.

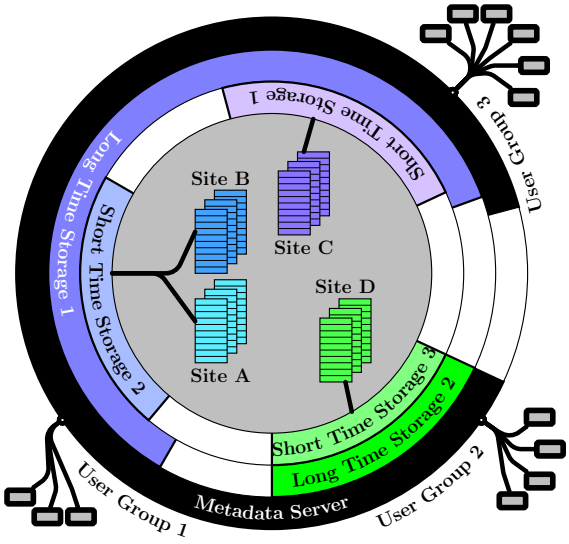

Figure 1: The layer based SLAte infrastructure for job centric monitoring data. Each layer can consist of multiple locally distributed servers. The servers of the MDS layer give a global view to all data which are stored distributed on the LTS servers.

The STS servers should be installed local at a site (e.g., at the frontend of a grid computing resource) to be close to the computing node on which the monitoring data are produced. To avoid that the monitoring data use too much resources on the computing node e.g., if stored in memory and to avoid that the monitoring data are copied after the job is done,

which prevents the next job to start, the monitoring information have to be moved to an STS server as early as possible. This results in a transfer for each single measurement and tends to many small packages which are handled by the local network (within a site or a computing system).

The monitoring data cached on the STS server have to be moved to the LTS server probably over a wide and slow network connection which is not able to handle huge numbers of small packages. To improve the usage of such a connection the monitoring data is transferred in batches. In most cases[9] the monitoring data of one job are moved at once. By repacking the monitoring data to huge packages it is avoided to slow down the network by transferring many small packages thus the effective network bandwidth is increased.

Like the STS layer, the LTS layer can consist of multiple servers to stores the monitoring data in a distributed way. In contradiction to the STS servers the LTS servers stores permanently and the monitoring data of individual STS servers can be merged (in Figure 1 symbolised by the sites A, B and C).

Furthermore an LTS server makes locally stored data accessible to users and visualisation systems. Therefore it can offer services to search and access job monitoring data.

To provide an unified view to the monitoring data distributed over LTS servers the MDS layer is used (see Figure 1). Like the other layers it can consist of multiple servers. Unlike the LTS and STS servers an MDS server has a global view to all data stored in the inner layers. In this way a user or tool easily accesses monitoring data without knowing any of the LTS servers. To realise this we distinguish between monitoring data and their metadata.

The monitoring data are the measurements (timestamp, CPU time, load average, used and free memory and so on) while the metadata hold information to search for jobs. Such an information is for instance the job ID. Usually, this ID is a unique number or a unique identifier like the Globus job ID. But mostly the job ID is not known by the user and it is preferred to search for the users jobs which run during a given time frame. Therefore the start and end time of a job are metadata as well as the name of the owner (more precisely the distinguished name of the certificate which was used to submit the job). Additional metadata are the exit state of the job and the storage location of the data.

To look up or search for jobs only the metadata are considered. This search can be performed by an LTS or an MDS server with the distinction that an LTS server holds only the metadata of the locally stored data while an MDS server has the metadata of all LTS servers. Thus an MDS server can find all monitoring data. The monitoring data is not stored on the MDS server because it is not needed for searching jobs. This dramatically reduces the amount of data transferred to these servers.

[9] To realise online monitoring it is needed to get monitoring data of running jobs. In this case the data on the LTS server are accessed and the data recorded afterwards have to be transferred in an additional package.

3.2 Implementation Details

The servers creating the three layers were implemented using the Grid middleware Globus (GT4) [Fos05] with its WSRF framework. The components are implemented in Java as GT4 resources and services which communicate with each other and the client applications via their SOAP based web services. So the components can be deployed to already running GT4 servers (like the frontend of computing resources) or dedicated GT4 containers can be used.

The security, authentication and authorisation services of GT4 are used to handle the monitoring data of grid users in a secure way. STS, LTS and MDS use grid server certificates to authenticate and authorise the communication. The authorisation of the user is based on users grid certificates.

Parts of the infrastructure we use for job centric monitoring are based on AMon [MPNB+06]. It offers visualisation components which are adapted to SLAte. Recording of data is done by lcg-mon-wn[10].

3.3 Exemplification of Scalability

Like already mentioned, one of the major concepts of SLAte is to be scalable with the demands of users and computing resources. Therefore additional servers can be installed in the three layers. In Figure 2 an example is shown how additional servers are added to increase the performance of SLAte.

Expansion stage 1 shows a minimal installation with one STS, one LTS and one MDS server for one computing resource. These servers are able to handle even more than one computing resource on the same site and additional users (expansion stage 2).

Additional computing resources on other sites require an additional STS server to buffer the monitoring data sent in small packages to the STS server. This configuration is shown as expansion stage 3 (Figure 2).

An additional LTS server adds storage capacity and a network link[11] which can be used to handle more STS servers with additional computing resources. This is shown by expansion stage 4.

Expansion stage 5 adds a MDS server to handle the search requests of an increased user group. In Figure 2 two distinct user groups are shown. If it is not possible to establish distinct user groups load balancing can be implemented with tools like pound[12].

[10] `http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/grid_computing/amon/index_html`

[11] A network link is added if the new LTS server is deployed on a new location.

[12] `http://www.apsis.ch/pound/`

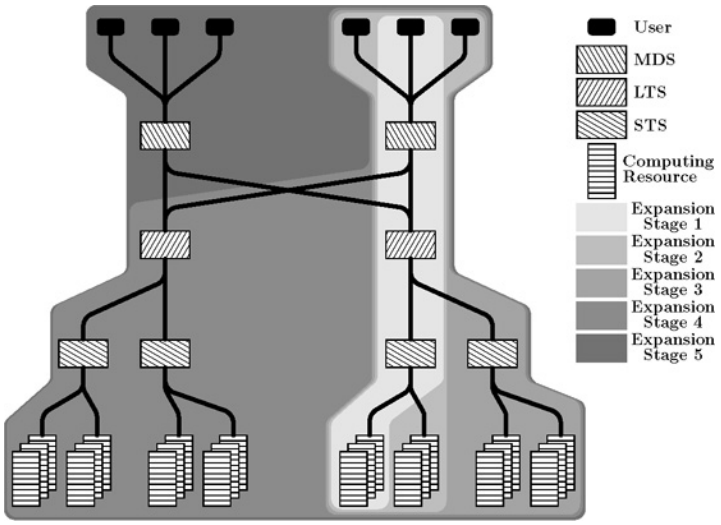

Figure 2: Example of different expansion states of a SLAte installation to handle different amount of users and resources to monitor.

4 Analysis of Scalability

In the following we look into aspects like required storage capacity and network performance for the servers in the three layers of SLAte. We show which criteria influence these aspects in particular the installation of additional servers.

For the analysis we consider a completely symmetrical installation. Every server in a layer has the same performance and each STS server has to handle the same amount of monitoring data. Thus, an STS server represents a computing resource and the load is caused by its job centric monitoring data. As a consequence the number of STS servers defines the overall amount of monitoring data which has to be handled by the SLAte infrastructure. The jobs are also considered as uniform and constant in computing time, measurement interval for monitoring and are equally distributed over the computing resources.

These assumptions are simplifications which only partially reflect the reality. But they ease the scalability analysis of the concepts used by the SLAte infrastructure. In future work we have to test and to verify them in a real installation.

4.1 Storage Capacity

The monitoring data are recorded on computing resources. They consist of the monitoring data and metadata. The amount of metadata k_{S_meta} is about constant. The monitoring data per job depends on the runtime of a job (t_{job}), the amount of monitoring data collected

on an single measurement (S_{single}) and the interval (t_{single}) between two measurements[13]. Due to the uniform jobs, these parameters are fixed and the amount of monitoring data is constant where t_{job}/t_{single} is the number of measurements per job.

$$k_{S\_mon} = S_{single} \cdot \frac{t_{job}}{t_{single}} \tag{1}$$

On an STS server the monitoring data and the according metadata are stored (S_{STS}) as long as the job is running. Afterwards the data is moved to an LTS server. Assuming the worst case scenario that all jobs start and stop at the same time, the storage capability depends on the amount of monitoring data and metadata and on the number of jobs (n_{jobs}) the STS server has to handle:

$$S_{STS} = n_{jobs} \cdot (k_{S\_mon} + k_{S\_meta}) \tag{2}$$

Each STS server sends its data to an LTS server. Thus the data stored on an LTS server depend on the number of STS servers copying data to it (n_{STS}), the data stored on one STS server (equation 2), the time interval the data is stored on an STS server (which is the runtime of a job) and the time the data should be hold on the LTS servers[14] (t_{hold}) which is defined by the configuration of the LTS server:

$$S_{LTS} = n_{STS} \cdot n_{jobs} \cdot (k_{S\_mon} + k_{S\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{3}$$

The amount of metadata, stored on a MDS server can be calculated similar to the metadata stored on an LTS server. Instead of monitoring data and metadata only metadata have to be considered but the information of all STS servers (n_{all_STS}) is stored.

$$S_{MDS} = n_{all\_STS} \cdot n_{jobs} \cdot (k_{S\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{4}$$

Before we start to interpret equation 2, 3 and 4 the number of STS servers per MDS server is replaced with $n_{STS} = n_{all\_STS}/n_{all\_LTS}$ where n_{all_LTS} is the number of all LTS servers. Also the amount of metadata is replaced with its fraction to the monitoring data k_{mon_meta} ($k_{S\_meta} = k_{S\_mon} \cdot k_{mon\_meta}$).

$$S_{STS} = n_{jobs} \cdot (k_{S\_mon} (1 + k_{mon\_meta})) \tag{5}$$

$$S_{LTS} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot n_{jobs} \cdot (k_{S\_mon} + (1 + k_{mon\_meta})) \cdot \frac{t_{hold}}{t_{job}} \tag{6}$$

$$S_{MDS} = n_{all\_STS} \cdot n_{jobs} \cdot (k_{S\_mon} \cdot k_{mon\_meta}) \cdot \frac{t_{hold}}{t_{job}} \tag{7}$$

[13] We assume that the measurements are done in constant time intervals. This simplifies the considerations. For more complex interval distribution it is possible to use the minimum or mean time between measurements.

[14] After some time (weeks or month) the monitoring information is considered as outdated and is removed automatically from the LTS servers

By considering that the amount of monitoring data is much larger then the metadata ($k_{S\_mon} \gg k_{mon\_meta}$) and by combining constants ($t_{hold}/t_{job} = k_1$) we get to the following approximations:

$$S_{STS} \approx n_{jobs} \cdot k_{S\_mon} \tag{8}$$

$$S_{LTS} \approx \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot S_{STS} \cdot k_1 \tag{9}$$

$$S_{MDS} \approx n_{all\_STS} \cdot S_{STS} \cdot k_{mon\_meta} \cdot k_1 \tag{10}$$

These formulas show how storage capacity relates to different aspects. In (8) we see that the needed capacity is proportional to the number of jobs which send monitoring data to an STS server.

The amount of data on an LTS server is proportional to the needed storage on an STS server[15] and can be lowered by increasing the number of LTS servers.

In opposite to the servers in the other layers, the needed capacity of the MDS server can not be decreased by installing additional servers (according to (10)). An important impact has k_{mon_meta}. It gives the fraction of metadata to monitoring data and is quite small (about 0.01 to 0.001). This gives a clear limitation because an MDS server has to be able to store all metadata. On other hand we expect that we can handle this limit and do not need to come up with an architecture overcoming this issue.

4.2 Network Capacity for Storing Monitoring Data

In this section we analyse the network capacities needed. Depending on how the network is used we have to consider factors which reflect this. If packages of reasonable size are transferred we expect nearly the maximum network speed. If small packages are used we have to consider a slowdown of k_{N_slow}. This is the case for the input network capacity of the STS servers (N_{STS_in}) where k_2 is a combination of other constant values:

$$N_{STS\_in} = k_{N\_slow} \cdot \frac{n_{jobs}}{t_{job}} \cdot (k_{S\_mon} + k_{S\_meta}) = k_{N\_slow} \cdot k_2 \tag{11}$$

For the output of an STS server the data is repacked to avoid a slow down of the network. The amount of data to be transferred is the same as for the input.

$$N_{STS\_out} = \frac{n_{jobs}}{t_{job}} \cdot (k_{S\_mon} + k_{S\_meta}) = k_2 \tag{12}$$

An LTS server receives data of multiple STS servers (where n_{all_STS}/n_{all_LTS} is the number of STS servers sending data to one LTS server)

$$N_{LTS\_in} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot N_{STS\_out} = \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \tag{13}$$

[15]The factor $k1$ depends on how long the information is stored and can be expected in the range from 10 to 1000.

and sends the metadata to each MDS server.

$$N_{LTS\_out} = n_{all\_MDS} \cdot \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \cdot \frac{k_{mon\_meta}}{1 - k_{mon\_meta}} \tag{14}$$

Based on the fact that $k_{mon\_meta} \ll 1$ and $k_{mon\_meta} > 0$ (metadata is small in comparison to monitoring data) we get to a approximation:

$$N_{LTS\_out} \approx n_{all\_MDS} \cdot \frac{n_{all\_STS}}{n_{all\_LTS}} \cdot k_2 \cdot k_{mon\_meta} \tag{15}$$

A MDS server (N_{MDS_in}) gets input from all LTS servers. For easier interpretation we used the same relations like for (15).

$$N_{MDS\_in} = n_{all\_STS} \cdot k_2 \cdot \frac{k_{mon\_meta}}{1 - k_{mon\_meta}} \approx n_{all\_STS} \cdot k_2 \cdot k_{mon\_meta} \tag{16}$$

Equations (11) and (12) show us that the network load on the STS server is bound by the resource to monitor (k_2) and that input is the more limiting factor. This aspect is represented by k_{N_slow}.

Additional LTS server lower the network demands per server of this layer. This is shown by (13) and (15).

For the MDS server we see similar limitations like for the required storage capacity. We have the factor k_{mon_meta} which reduces the transfer rate, because we only transfer metadata but the network utilisation raises with the overall amount of data delivered through STS servers. Thus, this aspect does not scale.

4.3 Network Capacity for Accessing Monitoring Data

Before the monitoring data can be read they have to be found. Therefore the MDS server is requested by the user to find relevant job monitoring data. Afterwards the data are read from the LTS servers.

The search requests are partitioned over all MDS servers and occur every t_{search}. The requests consist of k_{S_search} data which is about the amount of metadata. The answer is a list of n_{found} jobs found represented by their metadata k_{S_meta}. With these parameters the resulting network load on the MDS server N_{MDS_access} can be calculated:

$$N_{MDS\_access} = \frac{k_{S\_search} + n_{found} \cdot k_{S\_meta}}{n_{all\_MDS} \cdot t_{search}} \tag{17}$$

$$N_{MDS\_access} \approx \frac{k_{mon\_meta} \cdot k_{S\_mon} \cdot (n_{found} + 1)}{n_{all\_MDS} \cdot t_{search}} = \frac{k_{S\_mon\_meta}}{n_{all\_MDS}} \cdot k_3 \tag{18}$$

Where k_3 is defined as $k_3 = \frac{k_{S\_mon} \cdot (n_{found}+1)}{t_{search}}$.

On the LTS server the monitoring data k_{S_mon} and the metadata k_{S_meta} are requested. This happens every t_{search} for n_{found} jobs. Assuming that these data are equally distributed over all LTS servers, the network load on all LTS servers (N_{LTS_access}) is equal.

$$N_{LTS\_access} = \frac{(k_{S\_mon} + k_{S\_meta}) \cdot n_{found}}{n_{all\_LTS} \cdot t_{search}} \tag{19}$$

Considering that $k_{S\_mon} \gg k_{mon\_meta}$ and that each search delivers a huge amount of found jobs ($n_{found} \gg 1$) we can approximate the network load to:

$$N_{LTS\_access} \approx \frac{1}{n_{all\_LTS}} \cdot k_3 \tag{20}$$

Additional MDS or LTS servers reduce the load per server (according to (18) and (20)). The factor $k_{S\_mon\_meta}$ in (18) which is not present in (20) shows that the number of needed MDS servers is lower than the amount of LTS servers.

5 Conclusions and Outlook

We have shown a concept which handles job centric monitoring data in a distributed infrastructure with an uniform access layer. It distinguishes between monitoring data and their metadata. The concept and the idea to take different network infrastructures into account (within a site and between sites) by organising the servers in different layers is implemented by SLAte.

Handling monitoring data in a distributed way allows scalability in terms of storing capacity and network bandwidth by increasing the number of STS and LTS servers.

Implementing an uniform view to all monitoring data by using MDS servers which hold all metadata tends to bottlenecks. This is due to the fact that metadata of all monitored jobs are stored on each MDS server. But the amount of meta data per job is significantly lower in comparison to the monitoring data. This allows us to drive even huge installations.

A possible strategy to overcome the limitations of the MDS is a distributed search strategy. Therefore a component is needed which delegates a search request to all LTS servers and combines the results of the search. Such a component could replace the MDS servers without the need to change LTS and STS. The disadvantage of this strategy is an increased response time caused by the additional steps in the request handling and by the time to wait for the slowest LTS server.

A further topic of our scientific research is the visualisation and analysis of job centric monitoring data. Currently, we use AMon to e.g., display single jobs or to compare jobs by colour coded bars. To support the users in the monitoring our ongoing work is on the detection of abnormal job behaviour by an automatic analysis and comparison of multiple jobs.

References

[ACC+10] J. Andreeva, M. Calloni, D. Colling, F. Fanzago, J. D'Hondt, J. Klem, G. Maier, J. Letts, J. Maes, S. Padhi, S. Sarkar, D.Spiga, P. Van Mulders, and I. Villella. CMS Analysis Operations. *Journal of Physics: Conference Series*, 219(7):072007, 2010.

[BBK+09] Timo Baur, Rebecca Breu, Tibor Klmn, Tobias Lindinger, Anne Milbert, Gevorg Poghosyan, Helmut Reiser, and Mathilde Romberg. An Interoperable Grid Information System for Integrated Resource Monitoring Based on Virtual Organizations. *Journal of Grid Computing*, 7:319–333, 2009. 10.1007/s10723-009-9134-3.

[BHJR10] Holger Brunst, Daniel Hackenberg, Guido Juckeland, and Heide Rohling. Comprehensive Performance Tracking with Vampir 7. In Matthias S. Mller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 17–29. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-11261-4_2.

[EGM+] Erik Elmroth, Peter Gardfjll, Olle Mulmo, ke Sandgren, and Thomas Sandholm. A Coordinated Accounting Solution for SweGrid Version: Draft 0.1.3 Date: October 7, 2003.

[Fos05] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In Hai Jin, Daniel Reed, and Wenbin Jiang, editors, *Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13. Springer Berlin / Heidelberg, 2005. 10.1007/11577188_2.

[HMP10] Marcus Hilbrich and Ralph Müller-Pfefferkorn. A Scalable Infrastructure for Job-Centric Monitoring Data from Distributed Systems. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proceedings Cracow Grid Workshop '09*, pages 120–125, ul. Nawojki 11, 30-950 Krakow 61, P.O. Box 386, Poland, February 2010. ACC CYFRONET AGH.

[MPNB+06] Ralph Müller-Pfefferkorn, Reinhard Neumann, Stefan Borovac, Ahmad Hammad, Torsten Harenberg, Matthias Hüsken, Peter Mättig, Markus Mechtel, David Meder-Marouelli, and Peer Ueberholz. Monitoring of Jobs and their Execution for the LHC Computing Grid. In Marian Bubak, Michal Turala, and Kazimierz Wiatr, editors, *Proceedings Cracow Grid Workshop '06*, pages 224–231, ul. Nawojki 11, 30-950 Krakow 61, P.O. Box 386, Poland, October 2006. ACC CYFRONET AGH.

[PRAGA09] M. Piro Rosario, Guarise Andrea, Patania Giuseppe, and Werbrouck Albert. Using historical accounting information to predict the resource usage of grid jobs. *Future Generation Computer Systems*, 25(5):499–510, 2009.

[RSK+08] Miroslav Ruda, Jiří Sitera, Aleš Křenek, Luděk Matyska, Zděnek Šustr, and Michal Voců. Job Centric Monitoring on the Grid – 7 years of experience with L&B and JP services. *CESNET Conference*, 2008.

[VBC+10] D C Vanderster, F Brochu, G Cowan, U Egede, J Elmsheuser, B Gaidoz, K Harrison, H C Lee, D Liko, A Maier, J T Mocicki, A Muraru, K Pajchel, W Reece, B Samset, M Slater, A Soroko, C L Tan, and M Williams. Ganga: User-friendly Grid job submission and management tool for LHC and beyond. *Journal of Physics: Conference Series*, 219(7):072022, 2010.