Quantifying Performance and Scalability of the Distributed Monitoring Infrastructure SLAte

Marcus Hilbrich, Ralph Müller-Pfefferkorn

Software Quality Lab (s-lab)
Universität Paderborn
marcus.hilbrich@uni-paderborn.de
Center for Information Services and High Performance Computing
Technische Universität Dresden
ralph.mueller-pfefferkorn@tu-dresden.de

Abstract: Job-centric monitoring allows to observe the execution of programs and services (so called jobs) on remote and local computing resources. Especially large installations like Grids, Clouds and HPC systems with many thousands of jobs can have large benefits from intelligent visualisations of recorded monitoring data and semi-automatic analyses. The latter can reveal misbehaving jobs or non-optimal job execution and enables future optimisations to establish a more efficient use of the allocated resources.

The challenge of job-centric monitoring infrastructures is to store, search and access data collected on huge installations. 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, a performance evaluation, and the consequences for scalability are presented in this paper.

1 Introduction

Direct observability of computing tasks is more and more lost by using external and distributed resources for getting calculations done. Thus, misbehaving or obscure behaving jobs are rarely found and the optimisation potential of job execution is not satisfied. 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, giving detailed information about the used resources of running and completed jobs. Grid middlewares or batch systems usually just provide information about the job status like running or finished. Using more detailed information allows a semi-automatic analysis process to observe job execution with a minimal expenditure of time for users and administrators. It also enables the analysis of job failure reasons which is not possible by just checking the exit codes of jobs delivered by batch-systems.

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 cannot easily observe their jobs like on local resources by using desktop monitoring. Resource monitoring does

not solve this problem because it does not provide job specific information. It observes only the characteristics of resource usage, not the impact of single users or concrete jobs.

The architectures we have in focus for job-centric monitoring are Grids (offering huge amounts of heterogeneous resources), Clouds (offering virtual resources on demand) and HPC or cluster systems operated by computing centres. For the measurements and analyses presented in this paper we focused on the D-Grid infrastructure[1] which is a research network for scientists in Germany and the Globus Toolkit 4 (GT4) [Fos05] Middleware's web services which were common for many different Grid projects.

One of the challenges is the development of analysis strategies [HWT13] which handle the huge quantities of job-centric monitoring data or measurement series in general. But before we can analyse the data we have to handle them.

Our answer to the challenge of storing, accessing and searching huge amounts of job-centric monitoring data is the infrastructure SLAte[2] [HMP10]. It is built on a concept of distributed servers organised in three layers. These layers allow to distinguish between different network capabilities (e.g. within a site[3] and between different sites), capacities of various storage locations, and the localities of users or analysis services. The performance of each of the three layers can be increased by installing additional servers and thus can be adapted to needs.

This paper presents work related to job-centric monitoring, shortly describes the SLAte architecture, gives a performance evaluation for the different server types of SLAte and analyses the results in the context of the scalability concept of the job-centric monitoring infrastructure. Closing, conclusions and future work are presented.

2 Related Work

Job or system observation is an already established concept used for different needs. So it has been realised by different fields of research:

Monitoring of local computing resources: For local monitoring command line tools like ps, top or free[4] and graphical ones like Gnome System Monitor[5] can be used. On clusters Ganglia[6] gives information about the utilisation of resources. The impact of a specific user or job cannot be identified directly by these tools.

Tracing and profiling: Profiling tools like GNU gprof[7] give 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

[1] `http://www.dgrid.de/`
[2] SLAte stands for **S**calable **L**ayered **A**rchi**te**cture
[3] A site is a set of resources of a resource provider (in the Grid context) 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/`
[7] `http://sourceware.org/binutils/docs/gprof/`

tracing tools like Score-P [MBB+12] and Vampir [BHJR10]. To record the data the applications need to be instrumented, meaning adapted. Depending on the level of detail of the tracing a significant overhead may be introduced. Moreover, profiling or tracing infrastructures are usually designed to handle just one application/job and only on local resources. Thus it can not be easily adapted for job-centric monitoring.

Accounting: Accounting is used to measure resource utilisation and as a base for the billing of resources usage. An example is SGAS [EGM+]. Only basic and summary information of a job are needed and thus the amount of data to be handled is low. As a database is usually able to handle all accounting data, scalability is only a minor issue.

Resource monitoring: The task of resource monitoring is to record information about the (distributed) resources. Collected are information about e.g. hardware, offered services, outages and utilisation. This allows to create statistics or to assign jobs to resources. Examples are projects like D-Mon [BBK+09] and CMS Dashboard [ACC+10]. A lot of the information collected by these tools are static and have no need for a high frequency of updates.

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

3 Architecture of SLAte

SLAte is designed to increase the performance of handling monitoring data by deploying additional servers. This is needed if more users access the monitoring or new computing resources are added. The network capacity may be one limiting factor, the CPU capacity to process the data or the storage capacity are other ones. When observing many jobs in SLAte the overall performance can be adjusted by the number of deployed servers. For easy access to the distributed monitoring data we provide a unified and global view to easily access data. To this we created a concept based on three layers which is schematically shown in Fig. 1. We use the Short Time Storage (STS) layer to receive data from the monitoring clients running on the compute nodes and to store the data temporarily. The Long Time Storage (LTS) layer accumulates the data from the STS servers, stores them persistently and distributed over multiple servers. The outer layer is the Meta Data Service (MDS) which provides the global view on the data.

The STS servers are to be installed local at a site to be close to the computing nodes 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 might prevent the next job to start), the monitoring information have to be moved to an STS server as early as possible [HMP10]. 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).

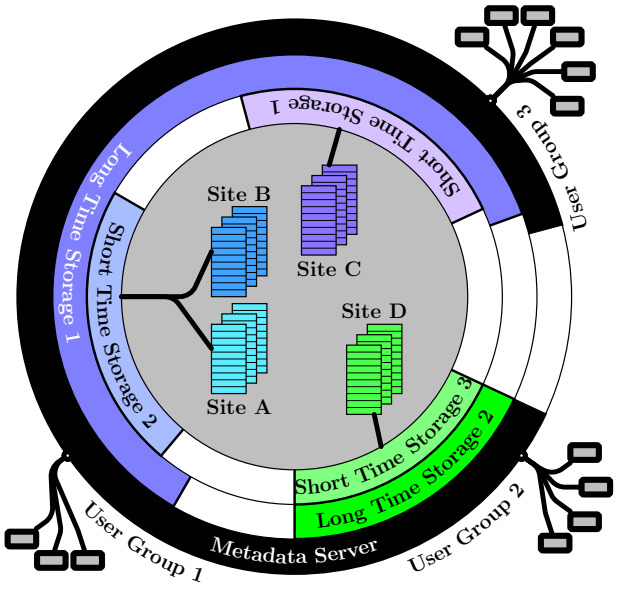


Figure 1: The layer-based SLAte infrastructure.

The monitoring data cached on the STS server have to be moved to the LTS server over a network connection which can be used more efficient when transferring packages of according size. To improve the performance of such a connection the monitoring data is transferred in batches. In most cases the monitoring data of one job are moved at once[8]. By repacking the monitoring data to large packages network congestion is avoided.

Like the STS layer, the LTS layer can consist of multiple servers to store the monitoring data in a distributed way. In contrast to the STS the LTS servers store the data persistently. The monitoring data of several individual STS servers can be merged (in Fig. 1 symbolised by the sites A to C). Furthermore, an LTS server makes locally stored data accessible and searchable to users and analysis systems.

To provide a unified view on the monitoring data distributed over the LTS servers the MDS layer is used (see Fig. 1). This layer can consist of multiple servers too. Unlike the LTS and STS servers an MDS server has a global view to all data. Via a single MDS server monitoring data can be easily accessed, without knowing other MDS or any of the LTS and STS servers.

In our architecture we distinguish between monitoring data and their meta data. The monitoring data are (time) series of measurements (like used CPU-time, load average, used or free memory) while the meta data hold information to search for jobs. Such informations are for instance the job ID, the time frame the job was active in and the executive system.

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

To look up or search for jobs only the meta data are considered. This search can be performed by an LTS or an MDS server with the distinction that an LTS server holds only the meta data of the locally stored data while an MDS server has the meta data of all LTS servers but does not hold the job data. Thus an MDS server can search on all monitoring data and the amount of data transferred to the MDS servers is dramatically reduced. In concrete a search request is answered with a list of storage locations for the data on LTS servers. With this list, a client can access the data directly and download it in parallel.

Not covered with this paper is the topic of protecting user-related data. Our first publication which concerns security for SLAte was [HMP12b]. Later on a very detailed explanation of the used protection concept was shows by [Hil14], which also covers an analysis of the protection demand of the stored data.

4 Performance Evaluation

4.1 Hardware and Software Used

SLAte has been implemented as a demonstration of the layer-based concept in the context of D-Grid. Thus we used software and hardware common for this research federation. The STS, LTS, and MDS servers are based on GT4 web services and use the certificate-based authentication and authorisation of GT4 with the public-key-infrastructure of D-Grid. This is a proper usage scenario with the drawback of web services as performance bottleneck.

As test hardware (see Tab. 1) for the three types of servers we used systems which where common for the D-Grid infrastructure and which could be allocated for exclusive use for our experiments. These were *Sun Fire X4100*™ with dual core *AMD*® *Opteron*™ *256* at a clock rate of 3.0 GHz equipped with 8 GB *DDR* main memory and *Gigabit Ethernet* network connection. It is clear that a more recent system could lead to a higher performance. To classify the results we give a short example of a system we used for production in D-Grid. This is a *Sun Fire X4600*™ with 16 CPU-cores at a clock rate of 2.6 GHz and 32 GB memory. This system was used as GT4 server at the Center for Information Services and High Performance Computing. During the usage we could observe that a submission of 500 to 1000 jobs could reproducibly overload the GT4 execution system. Even a more powerful system, the frontend of a cluster with an eight core *Intel*® *Xeon*® *X5365* at 3.0 GHz and 16 GB *DDR2* main memory was overloaded with about 1000 jobs, but the cluster offered enough nodes to generate load for our experiments without getting a bottleneck.

4.2 Principles of the Measurement

We started with performance measurements of the STS server. One job with eight hours runtime was excuted on the cluster with the job-centric monitoring enabled. We recorded the sampled monitoring data and their transfer to the STS server including the exact tim-

	GT4 server	load generation (clients)
CPU typ	*AMD® Opteron™ 256*	*Intel® Xeon® X5365*
CPU clock	$3.0\,GHz$	$3.0\,GHz$
cors per CPU	2	4
cors per node	2	8
RAM typ	*DDR*	*DDR2*
RAM per node	$8\,GB$	$16\,GB$
nodes per system	1	64
number of nodes used	1	up to 8 (only moderate load)
network	*Gigabit Ethernet*	*10 Gigabit Ethernet*

Table 1: Hardware used for the test system (GT4 server) and the system used for load generation.

ing. Based on the fact that the data volume is independent of the values of measurements (it depends on the number of measurements or runtime) a replayed job causes the same behaviour on the server like a real one. To test the load on the STS server, we needed to increase the amount of monitoring data sets sent to the STS server. This was done by replaying the previously recorded data simultaneously with the original clients and bash scripts to coordinate the timing. In this way we could use one CPU-core (at moderate load) of the cluster to simulate the sending of data of multiple jobs. This allows to simulate much larger systems than we used for our experiments. For the STS server (as like for the other server types) we did multiple tests with different loads. In the following we always refer to the run with the highest load without any overload or data lost.

To analyse the performance of a LTS server the output of STS servers is needed. To avoid to operate multiple STS servers, we recorded the output of a STS server including the timing. This was done by a separate run with lower number of jobs and debugging enabled. This record can be replayed and duplicated to adjust to a predefined load as in the case of the STS above. This was done by instrumenting the STS server's source code for recording and by implementing a client to replay the record.

Finally, to test the MDS server the LTS server was instrumented and recording was done like for the STS server. Another client was implemented to replay the data.

4.3 STS Server

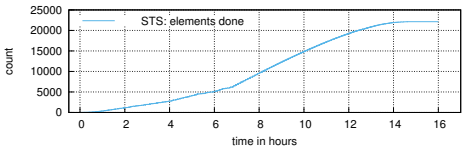
To test a STS server, the number of running jobs (jobs to handle in Fig. 2a) is increased constantly over seven hours. After eight hours the first jobs finish and the number of running jobs decreases until all jobs end after 15 hours. Thus the maximal load is reached after seven hours and holds for one hour. During this experiment it was tested that the STS server is not overloaded, which would result in a undefined state of the GT4 system and data loss.

For a practice-oriented scenario we need an LTS server at high load because this slows

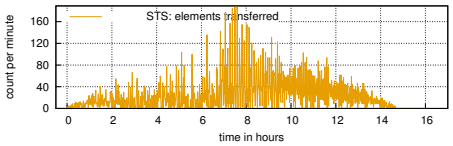
down the data transfer from the STS server. In preexaminations we discovered that the limitation for all our server types is the web server's CPU-load, not the network connection. So we decided to run the LTS server on the same hardware as the STS server. Based on the assumption, that the LTS can handle much more jobs than the STS server [HMP12a] we can justify this to get more practical relevant results. This is verified later on.

In our experiments we could use up to 276 jobs on the limited hardware described above. The result is shown in Fig. 2a. An increased job count leads to an overload of the STS server with discontinued data transfers. Also shown is the number of jobs stored on the LTS server. All job data were transferred and no data loss was seen during the experiments.

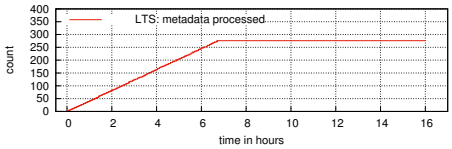
(a) Number of jobs currently using the STS server to store monitoring data and number of jobs whose monitoring data were transferred to the LTS server over time.



(b) Total number of received monitoring data sets over time.



(c) Number of monitoring data measurements processed per minute over time.



(d) Total number of transferred meta data packages over time.

Figure 2: Measurement results for the STS server

The transfer of the individual measurements (elements) to the STS server is shown in Fig. 2b. Each measurement has a size of 0.4 kB. Based on the fact that multiple measurements are done for each job, the number of measurements (22,131) is much higher then the number of jobs (276). Figure 2d shows the number of transferred meta data packages (2.4 kB each). For each job, exactly one package is sent.

According to Fig. 2b (total number of elements) the data rate is not always constant which meets the exception of the measurement . The data rate is constant in case the number of jobs stays constant, so the number of elements sent per time is determined. According to Fig. 2a this is the case from hour seven to eight. Figure 2b shows the expected constant rate (lineare increase) for this time frame. From hour zero to seven the number of jobs rises linear (see Fig. 2a). Thus the rate of transferred elements rises linear, resulting in an quadratic increase of the number of elements in Fig. 2b. After eight hours, the number of jobs decreases linear (see Fig. 2a). Thus the rate of elements is linear lowered, resulting in an increase of the number of elements based on a square root function (the amount as integral of the rate, the rate is a linear function with negativ sign). In short, an increase of the number of jobs results in a more then linear increase of the number of elements while a decrease results in a less than linear increase. In the time frame with the maximum

number of jobs (hour seven to eight) the rate of newly sent elements has its maximum and is constant.

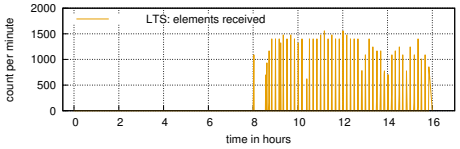
The number of measurements/elements transferred to the STS server is also shown in Fig. 2c. For this illustration, the number of elements was added up for each minute. It shows the already mentioned global change of the data rate as a fine granular fluctuation which is not obvious in the last diagrams. There are even minutes without any data transfer. The reason for the high variance of the transfer rate is the absence of a coordination of the data transfer.

4.4 LTS Server

The LTS server test is based on the outcomes of the analysis of the STS server. The monitoring data is replayed by clients on nodes of the cluster to simulate the STS servers. For each job, two communications are realised. The first one creates an object on the LTS server to hold all data of one job and to transfer the meta data. This is done directly after starting the job and enables users to find the job's information. The second transmission is done after the job has finished. It sends all monitoring data in one package and updates the end time as well as the exit state of the job which is part of the meta data. During the two communications the job is in state "to handle".

The measurement of the jobs handled by the LTS server is shown in Fig. 3a. It shows a behaviour similar to the STS server (Fig. 2a). During the first 7.5 hours the number of handled jobs increases constantly. Afterwards the value is maximal for about one hour, then it starts to decrease.

(a) Number of jobs in processing (meta data received but no monitoring data) and number of jobs persistently stored (meta data and monitoring data received) over time.



(b) Number of received measurements per minute over time.

Figure 3: Measurement results for the LTS server

The number of jobs which could be handled simultaneously by an LTS server at the test conditions is 826. In Fig. 2a the maximum is slightly larger. This is based on the fact that during the first minutes of the experiment (while the Java execution environment still optimises the execution of the LTS server) some data transfers are aborted and redone later. The jobs with the delayed transfer are not counted for the measurement but they show that the server is under high load. A further increase of the number of jobs results in a crash of the GT4 components and data loss.

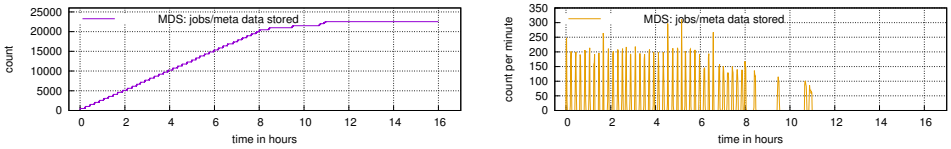
That the performance of an LTS exceeds the one of an STS server meets the expectations from the previous section and the theoretical analyse of SLAte in [HMP12a]. Based on the experiments we can also calculate the slow down factor[9] for the usage scenario $k_{N\_slow} = \frac{276}{826} = 0.33$ caused by using small packages to transfer the monitoring data.

Similar to the STS server measurements we visualise the transfer of the single measurements, even if they are sent as batches for each job of 112.5 kB each. Figure 3b shows that the data transfer starts after the first job has finished. A much higher transfer rate can be reached in comparison to the STS, but the fluctuation of the transfer rate is similar.

4.5 MDS Server

The MDS server only receives the meta data which are transferred as packages of 2.4 kB per job, monitoring data are not transferred. Thus a state like "to handle" is not present.

The number of jobs for which meta data are stored is shown by Fig. 4a. In the first eight hours the number is rising up to 20,470. This is similar to the behaviour of the LTS and STS server but the number of processed jobs is much higher. Afterwards the number of jobs should be constant, but the measurement show a slight increase. This is caused by transfers of meta data which got delayed based on the high load on the MDS server, but no data loss is observed. Again, an overload which leads to data loss is achieved by a further increase of the number of jobs.



(a) Number of transferred meta data packages (one per job) over time.

(b) Number of received meta data per minute over time.

Figure 4: Measurement for the MDS server

Comparing the MDS server to the STS server, the number of handled jobs is much higher with 20,470 compared to 276. The performance achieved for individual measurements (Fig. 2b) for the STS server is very similar to the one of the meta data on the MDS server. As seen at the other server types too, the transfer rate fluctuates quite strong (Fig. 4b) from zero transmissions to high values.

[9]The slow down factor was introduced in [HMP12a] and can be calculated as $k_{N\_slow} = \frac{N_{STS\_in}}{N_{LTS\_in}}$, with the realised network bandwidths N_{STS_in} for a STS server and N_{LTS_in} for the LTS server.

5 Scalability Observations

Scalability for the SLAte architecture is provided by the separation of servers (storage locations), repacking of data and the separation of monitoring and meta data. The separation allows to test individual servers of the STS and LTS layers, based on the fact that servers of one layer do not communicate with each other. They are only connected to a dedicated number of servers in the neighbouring layers. The repacking of the data of one job to one package gives a performance increase by a factor of more than three according to our measurements (276 jobs for a STS server to 826 jobs for a LTS server). This allows to connect one LTS server with three STS servers (using the hard and software used for the measurements).

As already shown in previous work [HMP12a] the MDS server is a bottleneck and defines the maximal size of an installation. Based on the data reduction which transfers only the meta data to the MDS server, the performance can be boosted to 20,470 jobs, which is a suitable number for monitoring a D-Grid like computing environment. For such a SLAte installation, an MDS server can be equipped with 25 LTS server and 75 STS servers. A single STS or LTS server is clearly not able to manage the monitor data of such a Grid infrastructure, this was already forecasted in [HMP10].

With a performance gain of a factor of 3 due to the repacking of the monitoring data and of a factor of 25 due to the distinction of monitoring and meta data (which allows to use multiple LTS servers for a installation), the overall performance increase is 75 compared to a centralised implementation with a performance similar to a STS server.

6 Conclusion and Future Work

This paper presents measurements to confirm the theoretical predication of our architectural concept described in [HMP12a]. We showed that the performance behaviour of the different layers allows to build up of a distributed monitoring infrastructure with a unified view on the data. The performance will be much higher than using a single storage location.

The presented experiments were not based on the most recent hardware. We even expect a higher performance when switching to more recent and more powerful hardware.

An interesting observation is the high fluctuation of the transfer rates. We have to analyse how to better coordinate the transfers to get a more constant and thus higher data rate. This also includes a discussion about replacing the underlying GT4-based web service we used for the current implementation for better performance.

We also observed the predicted bottleneck in the MDS server for storing data, which limits the growth of the infrastructure. In future developments we will address this shortcoming with respect to the scalable data access mechanism in a conceptional way. An additional task is to transfer the Grid implementation we have tested to Cluster and Cloud environments.

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. Müller, 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 Gardfjäll, Olle Mulmo, Åke 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.

[Hil14] Marcus Hilbrich. *Jobzentrisches Monitoring in verteilten heterogenen Umgebungen mit Hilfe innovativer skalierbarer Methoden*. Dissertation, Fakultät Informatik der Technischen Universität Dresden, Germany, September 2014.

[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.

[HMP12a] Marcus Hilbrich and Ralph Müller-Pfefferkorn. Achieving scalability for job centric monitoring in a distributed infrastructure. In Gero Mühl, Jan Richling, and Andreas Herkersdorf, editors, *ARCS Workshops*, volume 200 of *LNI*, pages 481–492. GI, 2012.

[HMP12b] Marcus Hilbrich and Ralph Müller-Pfefferkorn. Identifying Limits of Scalability in Distributed, Heterogeneous, Layer Based Monitoring Concepts like SLAte. *Computer Science*, 13(3):23–33, 2012.

[HWT13] Marcus Hilbrich, Matthias Weber, and Ronny Tschüter. Automatic Analysis of Large Data Sets: A Walk-Through on Methods from Different Perspectives. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 373–380, Dec 2013.

[MBB+12] Dieteran Mey, Scott Biersdorf, Christian Bischof, Kai Diethelm, Dominic Eschweiler, Michael Gerndt, Andreas Knüpfer, Daniel Lorenz, Allen Malony, Wolfgang E. Nagel, Yury Oleynik, Christian Rössel, Pavel Saviankou, Dirk Schmidl, Sameer Shende, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-P: A Unified Performance Measurement System for Petascale Applications. In Christian Bischof, Heinz-Gerd Hegering, Wolfgang E. Nagel, and Gabriel Wittum, editors, *Competence in High Performance Computing 2010*, pages 85–97. Springer Berlin Heidelberg, 2012.