Towards Intelligent Management of Very Large Computing Systems

Eugen Volk, Jochen Buchholz, Stefan Wesner, Daniela Koudela, Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, Bernd Freisleben, Götz Isenmann, Jürgen Schwitalla, Marc Lohrer, Erich Focht, Andreas Jeutter

Abstract The increasing complexity of current and future very large computing systems with a rapidly growing number of cores and nodes requires high human effort on administration and maintenance of these systems. Existing monitoring tools are neither scalable nor capable to reduce the overwhelming flow of information and provide only essential information of high value. Current management tools lack on scalability and capability to process a huge amount of information intelligently by relating several data and information from various sources together for making right decisions on error/fault handling. In order to solve these problems, we present a solution designed within the TIMaCS project, a hierarchical, scalable, policy based monitoring and management framework.

Eugen Volk, Jochen Buchholz, Stefan Wesner
High Performance Computing Center Stuttgart, Nobelstrasse 19, D-70569 Stuttgart, Germany
e-mail: {volk, buchholz, wesner}@hlrs.de ·
Daniela Koudela, Technische Universität Dresden, Zentrum für Informationsdienste und Hochleistungsrechnen (ZIH), D-01062 Dresden, Germany
e-mail: daniela.koudela@tu-dresden.de ·
Matthias Schmidt, Niels Fallenbeck, Roland Schwarzkopf, Bernd Freisleben
Department of Mathematics and Computer Science, University of Marburg
Hans-Meerwein-Str. 3, D-35032 Marburg, Germany
e-mail: {schmidtm, fallenbe, rschwarzkopf, freisleb}@informatik.uni-marburg.de ·
Götz Isenmann, Jürgen Schwitalla, Marc Lohrer
science + computing ag, Hagellocher Weg 73, D-72070 Tübingen, Germany
e-mail: {isenmann, j.schwitalla, lohrer}@science-computing.de ·
Erich Focht, Andreas Jeutter
NEC High Performance Computing Europe, Hessbruehlstrasse 21b, D-70565 Stuttgart, Germany
e-mail: {efocht, ajeutter}@hpce.nec.com

1 Introduction

Operators of very large computing centres face the challenge of the increasing size of their offered systems following Moores or Amdahls law already for many years. Until recently the effort needed to operate such systems has not increased similarly thanks to advances in the overall system architecture, as systems could be kept quite homogeneous and the number of critical elements with comparably short Mean Time Between Failure (MTBF) such as hard disks could be kept low inside the compute node part.

Current petaflops and future exascale computing systems would require an unacceptable growing human effort for administration and maintenance based on an increased number of components. But even more would the effort rise due to their increased heterogeneity and complexity [1, 2, 3]. Computing systems cannot be built anymore with more or less homogeneous nodes that are similar siblings of each other in terms of hardware as well as software stack. Special purpose hardware and accelerators such as GPGPUs and FPGAs in different versions and generations, different memory sizes and even CPUs of different generations with different properties in terms of number of cores or memory bandwidth might be desirable in order to support not only simulations covering the full machine with a single application type, but also more coupled simulations exploiting the specific properties of a hardware system for different parts of the overall application. Different hardware versions go together with different versions and flavours of system software such as operating systems, MPI libraries, compilers, etc. as well as different, at best individual user specific, variants combining different modules and versions of available software fully adapted to the requirements of a single job. Additionally the operation model from purely batch might be complemented by usage models allowing more interactive or time controlled access for example for simulation steering or remote visualization jobs.

While the problem of detecting hardware failures such as a broken disk or memory has not changed and still can be done similarly as in the past by specific validation scripts and programs between two simulation jobs the problems that occur in relation with different software versions or only in specific use scenarios are much more complex to be detected and are clearly beyond what a human operator can address with a reasonable amount of time. Consequently the obvious answer is that the detection of problems based on different type of informations collected at different time steps needs to be automated and moved from the pure data level to the information layer where an analysis of the information either leads to recommendations to a human operator or at best triggers a process applying certain counter measures automatically.

A wide range of monitoring tools such as Ganglia [4] or ZenossCore [5] exist that are neither scalable to the system sizes of thousands of nodes and hundred thousands of compute cores, cannot cope with different or changing system configurations (e. g. this service is only available if the compute node is booted in certain OS modes), and the fusion of different informations to a consolidated system analysis state is missing, but more important they lack a powerful mechanism to analyse the informations monitored and to trigger reactions to change the system state actively to bring the system state back to normal operations.

Another major limitation is the lack of integration of historical data in the information processing, the lack of integration with other data sources (e. g. planned system maintenance schedule database) and the very limited amount of counter measures that can be applied. In order to solve these problems, we propose in scope of the TIMaCS [6] project a scalable, hierarchical policy based monitoring and management framework. The TIMaCS approach is based on an open architecture allowing the integration of any kind of monitoring solution and is designed to be extensible for information consumers and processing components. The design of TIMaCS follows concepts coming from the research domain of organic computing (e. g. see references [7] and [8]) also propagated by different computing vendors such as IBM in their autonomic computing [9] initiative.

In this paper we present the TIMaCS solution in form of a hierarchically structured monitoring and management framework, capable to solve the challenges and problems mentioned above.

2 Related Work

There are lots of tools available supporting the monitoring and management of large systems but they all originate from one of the two domains. Either the tools [10] (like Nagios [11], Ganglia [4], Zenoss [5]) are designed to monitor systems with only rudimentary management capabilities like executing a specific command for each failing sensor state. But they don't care about implications resulting from other failures. Additionally their scalability is limited in the focus of future high performance computing resources, i. e. current techniques visualizing the status will no longer be adequate due to the huge amount of data. Or the tools are designed to manage systems like Tivoli [12] which means to force to set up the machines according to an overall configuration which is normally done regardless of the underlying state. This is mostly done on a regular basis to force global changes down to all systems and even install needed software if not available and so on. But changing configurations in reaction to failing systems or services is not covered by them, so no real error handling can be done.

3 TIMaCS - Solution

The project TIMaCS (Tools for Intelligent System Management of Very Large Computing Systems) is initiated to solve the above mentioned issues. TIMaCS deals with the challenges in the administrative domain upcoming due to the increasing complexity of computing systems especially of computing resources with a performance of several petaflops. The project aims at reducing the complexity of the manual administration of computing systems by realizing a framework for intelligent management of even very large computing systems based on technologies for virtualization, knowledge-based analysis and validation of collected information, definition of metrics and policies.

The TIMaCS framework includes open interfaces which allow easy integration of existing or new monitoring tools, or binding to existing systems like accounting, SLA management or user management systems. Based on predefined rules and policies, this framework will be able to automatically start predefined actions to handle detected errors, additionally to the notification of an administrator. Beyond that the data analysis based on collected monitoring data, regression tests and intense regular checks aims at preventive actions prior to failures.

We seek for developing a framework ready for production and its validation at the High Performance Computing Center Stuttgart (HLRS), the Center for Information Services and High Performance Computing (ZIH) and the Distributed Systems Group at the Philipps University Marburg. NEC with the European High Performance Computing Technology Center and science + computing are the industrial partners within the TIMaCS project. The project funded by the German Federal Ministry of Education and Research started in January 2009 and will end in December 2011.

Following subsections describe the TIMaCS framework, presenting its architecture and components.

3.1 High Level Architecture

The description of the TIMaCS architecture provided in this section is based on earlier papers of the authors [13, 14]. In contrast to earlier papers, which described TIMaCS architecture on a very high level, this paper presents the architecture more detailed, describing each component.

The self management concept of the proposed framework follows IBM autonomic computing reference architecture [9]. Self managing autonomic capabilities in computer systems perform tasks that IT professionals choose to delegate to the technology according to predefined policies and rules. Thereby, policies determine the type of decisions and actions that autonomic capabilities perform [9].

The TIMaCS framework is designed as a policy based monitoring and management framework with an open architecture and a hierarchical structure. The hierarchy of the TIMaCS framework is formed by management layers acting on different levels of information abstraction. This is achieved by generating state information for groups of different granularity: resources/node, node-group, cluster, organization. These granularities form abstraction layers. A possible realization of the hierarchies can be achieved in a tree-like structure, as shown in Figure 1.

The bottom layer, called resource/node layer, contains resources or compute nodes with integrated sensors, which provide monitoring information about resources or services running on them. Additionally, each managed resource has integrated Delegates. These are interfaces, which allow to execute commands on managed resources. Furthermore, there exist other Delegates which are not directly integrated in resources (e. g. Job Scheduler), but have an indirect possibility to influence those resources (e. g. by removing error nodes from the batch queue).

Each management layer consists of dedicated nodes, called TIMaCS nodes, with monitoring and management capabilities organized in two logical blocks. The monitoring block collects information from the nodes of the underlying layer. It aggregates the information and makes conclusions by pre-analysing information, creating group states of certain granularity and triggering events, indicating possible errors. The management block analyses triggered events applying intelligent escalation strategies, and determines which of them need decisions. Decisions are made in accordance with predefined policies and rules, which are stored in a knowledge base filled up by system administrators when configuring the framework. Decisions result in commands, which are submitted to Delegates and executed on managed resources (compute nodes) or other components influencing managed resources, like a job-scheduler, capable to remove defective nodes from the batch queue. The hierarchic structure of the TIMaCS framework allows reacting on errors locally with very low latency. Each decision is reported to the upper layer to inform about detected error-events and selected decisions to handle the error-event. The upper layer, which generally has more information, is now able to intervene on received reports

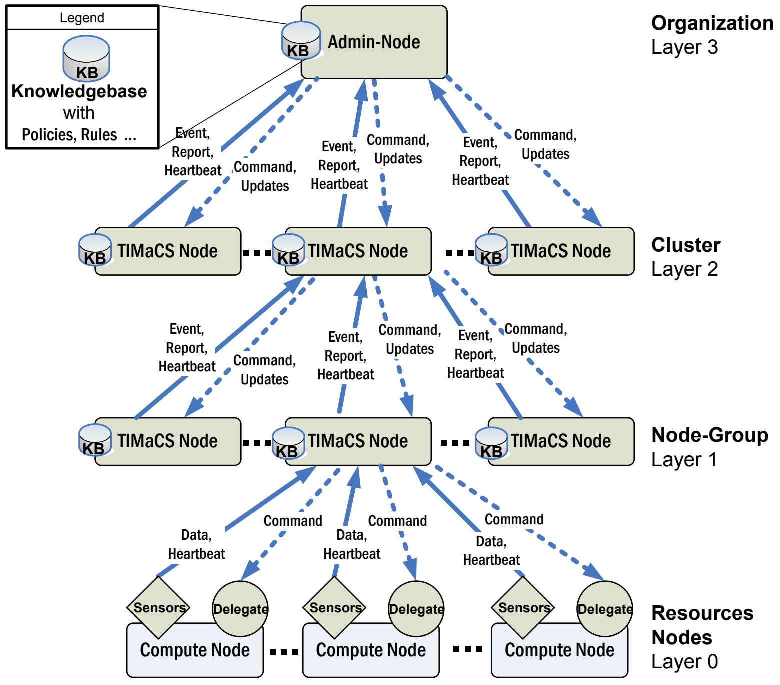

Fig. 1 The hierarchy of TIMaCS

by making new decisions resulting in new commands, or an update of the knowledge base of the lower layer. Only escalated reports/events require the effort of an administrator for deeper analysis.

On top of the framework an Admin-Node is settled, which allows administrators to configure the framework, the infrastructure monitoring, to maintain the Knowledge Base and to execute other administrative actions. All nodes are connected by message based communication infrastructure with fault tolerance capabilities and mechanisms ensuring the delivery of messages following the AMQP standard.

The following subsections describe the architecture of the TIMaCS components in detail, explaining monitoring, management and virtualisation of the system.

3.2 Monitoring

The monitoring capability of the TIMaCS node provided in the monitoring block, consists of Data-Collector, Storage, Aggregator, Regression Tests, Compliance Tests and the Filter & Event Generator, as shown in Figure 2.

The components within the monitoring block are connected by messaging middleware, enabling flexible publishing and consumption of data according to topics. These components are explained in the subsequent sections.

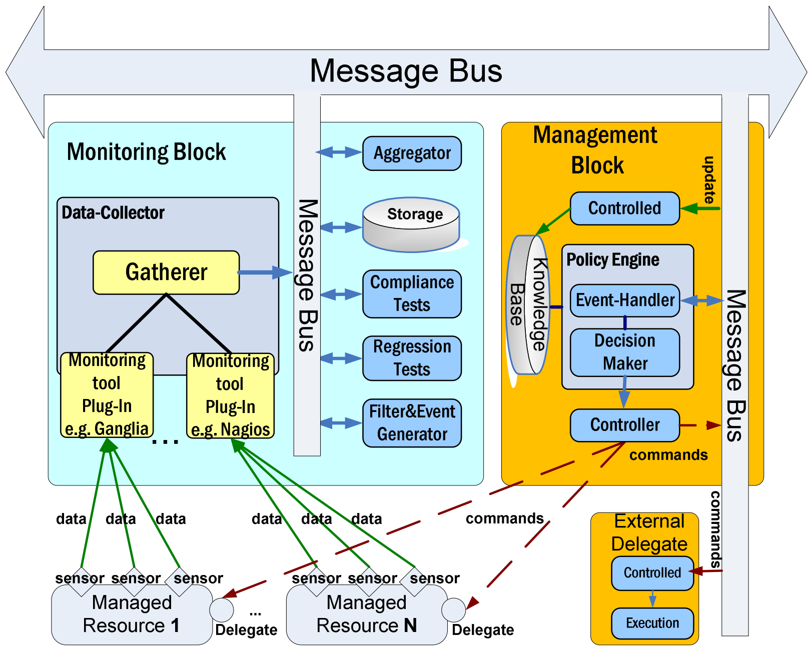

Fig. 2 Monitoring and Management in TIMaCS

3.2.1 Data-Collector

The *Data-Collector* collects metric data and information about monitored infrastructure from different sources, including compute nodes, switches, sensors or other sources of information. The collection of monitoring data can be done synchronous or asynchronous, in pull or push manner, depending on the configuration of the component. In order to allow integration of various existing monitoring tools (like Ganglia [4] or Nagios[11]) or other external data-sources, we use a plug-in-based concept, which allows the design of customized plugins, capable to collect information from any data-source, as shown in Figure 2. Collected monitoring data consist of metric values, and are semantically annotated with additional information, describing source location, the time, when the data were received, and other relevant information for data processing. Finally, annotated monitoring data are published according to topics, using AMQP based messaging middleware, ready to be consumed and processed by other components.

3.2.2 Storage

The *Storage* subscribes to the topics published by the Data-Collector, and saves the monitoring data in the local round robin database. Stored monitored data can be retrieved by system administrators and by components analysing data history, such as Aggregator or Regression Tests.

3.2.3 Aggregator

The *Aggregator* subscribes to topics produced by the Data-Collector, and aggregates the monitoring data, i. e. by calculating average values, or the state of certain granularity (services, nodes, node-groups, cluster etc.). The aggregated information is published in new topics, to be consumed by other components of the same node (i. e. by the Filter & Event Generator), or those of the upper layer.

3.2.4 Regression Tests

Regression Tests help cutting down on system outage periods by identifying components with a high probability of soon failure. Replacing those parts during regular maintenance intervals avoids system crashes and unplanned downtimes.

To get an indication, if the examined component may break in the near future, regression tests evaluate the chronological sequence of data for abnormal behaviour. The algorithm, which analyses those data, we call regression analysis. Since different metrics may need different algorithms for obtaining usable hints of the proper functioning of a component, TIMaCS allows for different regression analyses, which are implemented through an open interface.

One of the implemented regression analyses is the linear regression. There, a linear function is fitted to the data and the slope is returned. This algorithm is especially useful for predicting the state of a hard disk and evaluating memory errors on a DIMM.

TIMaCS distinguishes between online- and offline-regression tests. Online-regression tests are performed on a regular time interval and evaluate the most recent historical data being delivered by the publish/subscribe-system. Offline-regression tests on the contrary are only performed on request. They query the database to obtain their data for evaluation.

3.2.5 Compliance Tests

Compliance Tests enable early detection of software and/or hardware incompatibilities. They verify, if the correct versions of firmware, hardware and software are installed and they test, if every component is on the right place and working properly. Compliance tests are only performed on request since they are designed to run at the end of a maintenance interval or as a preprocessor to batch jobs. They may use the same sensors as used for monitoring but additionally they allow for starting bench marks.

Both compliance and regression tests are an integral part of TIMaCS. Therefore, they can easily be automated and help reducing the manual administrative costs.

3.2.6 Filter & Event Generator

The *Filter & Event Generator* subscribes to particular topics produced by the Data-Collector, Aggregators, and Regression- or Compliance Tests. It evaluates received data by comparing it with predefined values. In case that values exceed permissible ranges, it generates an event, indicating a potential error. The event is published according to a topic and sent to that components of the management block, which subscribed to that topic.

The evaluation of data is done according to predefined rules, defining permissible data ranges. These data ranges may differ depending on the location, where these events and messages are published. Furthermore, the possible kinds of messages and ways to treat them may vary strongly from site to site and in addition it depends on the layer the node belongs to.

The flexibility obviously needed can only be achieved by providing the possibility of explicitely formulating the rules by which all the messages are handled. TIMaCS provides a graphical interface for this purpose, based on eclipse Graphical Modelling Framework [15].

3.3 Management

The management block is responsible for making decisions in order to handle error events. It consists of the following components: Event Handler, Decision Maker, Knowledge Base, Controlled and Controller, as shown in Figure 2. Subsequent sections describe these components in detail.

3.3.1 Event Handler

The *Event Handler* analyses received reports and events applying escalation strategies, to identify those which require error handling decisions. The analysis comprises methods evaluating the severity of events/reports and reducing the amount of related events/reports to a complex event. The evaluation of severity of events/reports is based on their frequency of occurrence and impact on health of affected granularity, as service, compute node, group of nodes, cluster etc. The identification of related events/reports is based on their spatial and temporal occurrance, predefined event relationship patterns, or models describing the topology of the system and dependencies between services, hardware and sensors. After the event has been classified as ”requiring decision”, it is handed over to the Decision Maker.

3.3.2 Decision Maker

The *Decision Maker* is responsible for planning and selecting error correcting actions, made in accordance with predefined policies and rules, stored in the Knowledge Base. The local decision is based on an integrated information view, reflected in a state of affected granularity (compute node, node group, etc.). Using the topology of the system and dependencies between granularities and sub-granularities, the Decision Maker identifies the most probable origin of the error. Following predefined rules and policies, it selects decisions to handle identified errors. Selected decisions are mapped by the Controller to commands, and are submitted to nodes of the lower layer, or to Delegates of managed resources.

3.3.3 Knowledge Base

The *Knowledge Base* is filled up by the system administrators when configuring the framework. It contains policies and rules as well as information about the topology of the system and the infrastructure itself. Policies stored in the Knowledge Base are expressed by a set of objective statements prescribing the behaviour of the system on a high level, or by a set of (event, condition, action) rules defining actions to be executed in case of error detection, thus prescribing the behaviour of the system on the lower level.

3.3.4 Controller, Controlled and Delegate

The *Controller* component maps decisions to commands and submits these to *Controlled* components of the lower layers, or to *Delegates* of the managed resources.

The *Controlled* component receives commands or updates from the *Controller* of the management block of the upper layer and forwards these, after authentication and authorization, to adressed components. For example, received updates containing new rules or information, are forwarded to the Knowledge Base to update it.

The *Delegate* provides interfaces enabling the receipt and execution of commands on managed resources. It consists of *Controlled* and *Execution* components. The *Controlled* component receives commands or updates from the channels to which it is subscribed and maps these to device specific instructions, which are executed by the *Execution* component. In addition to *Delegates* which control managed resources directly, there are other *Delegates* which can influence the behaviour of the managed resource indirectly. For example, the virtualization management component, presented in Section 3.4, is capable to migrate VM-instances of affected or faulty nodes to healthy nodes.

3.4 Virtualization in TIMaCS

Virtualization is an important part of the TIMaCS project, since it enables partitioning of HPC resources. Partitioning means that the physical resources of the system are assigned to host and execute user-specific sets of virtual machines. Depending on the users' requirements, a physical machine can host one or more virtual machines that either use dedicated CPU cores or share the CPU cores. Virtual partitioning of HPC resources offers a number of benefits for the users as well as for the administrators. Users no longer rely on the administrators to get new software (including dependencies such as libraries) installed, but they can install all software components in their own virtual machine. Additional protection mechanisms including the virtualization hypervisor itself guarantee protection of the physical resources. Administrators benefit from the fact that virtual machines are easier to manage in certain circumstances than physical machines. One of the benefits of using TIMaCS is to have an automated system that makes decisions based on a complex set of rules. A prominent example is the failure of certain hardware components (e. g. fans) which leads to an emergency shutdown of the physical machines. Prior to the actual system shutdown, all virtual machines are live-migrated to another physical machine. This is one of the tasks of the TIMaCS virtualization component.

The used platform virtualization technology in the TIMaCS setup is the Xen Virtual Machine Monitor [16] since Xen with para-virtualization offers a reasonable tradeoff between performance and manageability. Nevertheless, the components are based on the popular *libvirt* [1] implementation and thus can be used with other hyper-

[1] http://libvirt.org/

visors such as the Kernel Virtual Machine (KVM). The connection to the remaining TIMaCS framework is handled by a Delegate that receives commands and passes them to the actual virtualization component. A command could be the request to start a number of virtual machines on specific physical machines or the live migration from one machine to another. If the framework relies on a response, i. e. it is desirable to perform some commands synchronously, the Delegate responds back to an event channel.

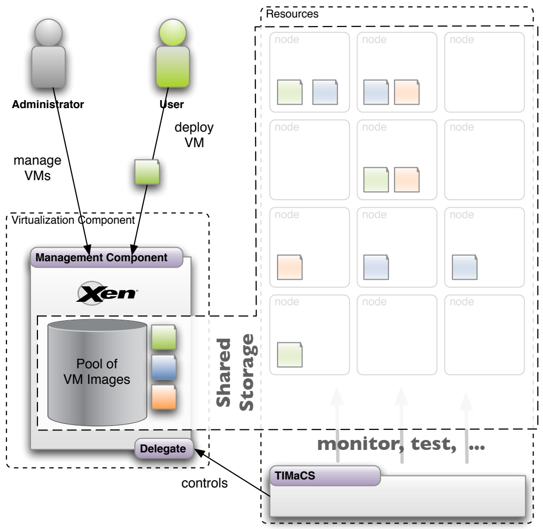

Fig. 3 TIMaCS Virtualization Components

Figure 3 describes the architecture of the TIMaCS virtualization components. The image pool plays a central rule since it contains all virtual machines' disk images either created by the user or the local administrator. Once a command is received via the Delegate, the virtualization component takes care of executing it.

3.5 Communication Infrastructure

To enable communication in TIMaCS, all TIMaCS nodes of the framework are connected by a scalable message based communication infrastructure supporting publish/subscribe messaging pattern, with fault tolerant capabilities and mechanisms ensuring delivery of messages, following the Advanced Message Queuing Protocol (AMQP) [17] standard. Communication between components of the same node is done internally, using memory based exchange channels bypassing the communication server. In a topic-based publish/subscribe system, publishers send messages or events to a broker, identifying channels by unique URIs, consisting of topic-name and exchange-id. Subscribers use URIs to receive only messages with particular topics from a broker. Brokers can forward published messages to other brokers with subscribers that are subscribed to these topics.

The format of topics used in TIMaCS consists of several sub-keys (not all sub-keys need to be specified):

```
<source/target>.<kind>.<kind-specific>
```

- The sub-key *source/target* specifies the sender(group) or receiver(group) of the message, identifying a resource, a TIMaCS node or a group of message consumers/senders.
- The sub-key *kind* specifies the type of the message (data, event, command, report, heartbeat, . . .), identifying a type of the topic consuming component.
- The sub-key *kind-specific* is specific to *kind*, i. e., for the kind ”data”, the kind-specific sub-key is used to specify the metric-name.

The configuration of the TIMaCS communication infrastructure comprises setup of the TIMaCS nodes and AMQP based messaging middleware, connecting TIMaCS nodes according to the topology of the system. This topology is statically at the begining of the system setup, but can be changed dynamically by system updates during run time. To build up a topology of the system, a connection between TIMaCS nodes and AMQP servers, the latter are usually colocated with TIMaCS nodes in order to achieve scalability, must follow a certain scheme. *Upstreams*, consisting of event-, heartbeat-, aggregated-metrics and report-messages, are published on messaging servers of the superordinated management node, enabling faster access to received messages. *Downstreams*, consisting of commands and configuration updates, are published on messaging servers of the local management node. This ensures that commands and updates are distributed in an efficient manner to adressed nodes or group of nodes.

Using AMQP based publish/subscribe system, such as RabbitMQ [18], enables TIMaCS to build up a flexible, scalable and fault tolerant monitoring and management framework, with high interoperability and easy integration.

4 Conclusion

Challenges in the area of administration of very large computing systems have led to the design of the TIMaCS solution, a scalable policy based monitoring and management framework. From the system monitoring point of view, the presented TIMaCS framework reduces the overwhelming information flow of monitoring-data by handling and filtering it on different levels of abstractions. At the same time, it increases the information value delivered to the system administrator, comprising only necessary and important information. The usage of compliance- and regression tests enables administrators to realize preventive actions, allowing to check the status of the infrastructure preventively. The plug-in based monitoring concept enables integration of various existing monitoring tools like Ganglia, Nagios, ZenossCore and other information sources, so providers are not forced to replace their existing monitoring installation.

One big issue for system administrators, especially on HPC resources, is the capability to take actions in case of an error in a predefined way. The TIMaCS management framework supports different automation and escalation strategies to handle errors based on policies, including notification of an administrator, semi-automatic to fully-automatic counteractions, prognoses, anomaly detection and their validation. Automated error handling reduces the system recovery time. The hierarchic structure of the framework allows reacting on errors locally with very low latency, although the full system status can be used for making a decision. In addition, the upper layers can intervene if their more global view leads to another decision.

The virtualization concept used in TIMaCS enables administrators easy partitioning and dynamic user assignment of very large computing systems, allowing setup, migration or removal of single compute nodes out of a heterogeneous or hybrid system.

Using the AMQP based publish/subscribe system enables TIMaCS to build up a flexible, scalable and fault tolerant monitoring and management framework, with high interoperability and easy integration.

By installing the TIMaCS framework the administrator will be enabled to specify rules and policies with simple access to all monitored data. So it is not necessary to know any details about the underlying monitoring systems since the sensor information is standardized. So defining error handlings becomes very simple and they can also be activated in different manners from manually on demand to fully automated when the actions are well tested. Going far beyond current practice it is even possible to define lots of different cluster configurations and set them up in different partitions of the whole system in parallel or changing over time. So it is possible to cut off a specific part of the cluster for urgent computing with higher storage bandwidth for some time or change the scheduling according to the changing submit behaviour from weekdays to weekends. It might even be possible with some further developments to allow users to define which hardware characteristics they need i. e. minimal network but high compute power may result in a restriction of the network so that the overall network performance can be assured to other users. This may result in more detailed payment models.

Thus TIMaCS will give the administrator a tool so that he can manage increasing system complexity and handle current issues like errors and even be prepared to do changes very dynamically, which currently need a very long time consuming actions with a lot of manual overhead, so that in practice it is done only once when installing the cluster or extending it.

Acknowledgements The results presented in this paper are partially funded by Federal Ministry of Education and Research (BMBF) through the TIMaCS [6] project.

References

1. Strohmaier, E., Dongarra, J., J., Meuer, H., W., Simon, H., D.,: Recent trends in the marketplace of high performance computing, Parallel Computing, Volume 31, Issues 3-4, March-April 2005, pp 261-273
2. Wong, Y., W., Mong Goh R, S, Kuo, S., Hean Low, M., Y.: A Tabu Search for the Heterogeneous DAG Scheduling Problem, 2009 15th International Conference on Parallel and Distributed Systems
3. Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., Yelick, K.: A view of the parallel computing landscape, Communications of the ACM, v.52 n.10, October 2009
4. Ganglia web-site, http://ganglia.sourceforge.net/
5. Zenoss web-site, http://www.zenoss.com
6. TIMaCS project web-site, http://www.timacs.de
7. organic-computing web-site, http://www.organic-computing.de/spp
8. Wuertz, R.P.: Organic Computing (Understanding Complex Systems), Springer 2008
9. IBM: An architectural blueprint for autonomic computing, http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_V7.pdf, IBM Whitepaper, June 2006. Cited 16 December 2010.
10. Linux Magazin, Technical Review, Monitoring, 2007
11. Nagios web-site, http://www.nagios.org
12. IBM Tivoli web-site, http://www-01.ibm.com/software/tivoli/
13. Buchholz, J., Volk, E.: The Need for New Monitoring and Management Technologies in Large Scale Computing Systems. In: Procedings of eChallenges 2010, to appear.
14. Buchholz, J., Volk, E. (2010): Towards an Architecture for Management of Very Large Computing Systems. In: Resch, M.,Benkert, K., Wang, X., Galle, M., Bez, W., Kobayashi, H., Roller, S. (ed) High Performance Computing on Vector Systems 2010, Springer, Berlin Heidelberg.
15. eclipse Graphical Modeling Project (GMP) http://www.eclipse.org/modeling/gmp/
16. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the Art of Virtualization, in SOSP '03: Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003, ACM Press, Bolton Landing, NY, USA
17. Advanced Message Queuing Protocol (AMQP) web-site, http://www.amqp.org
18. RabbitMQ web-site, http://www.rabbitmq.com