

CHEMNITZ UNIVERSITY
OF TECHNOLOGY

Faculty of Computer Science
Computer Architecture Group

Diploma Thesis
**Monitoring of large-scale Cluster
Computers**

Stefan Worm

stefan.worm@hrz.tu-chemnitz.de

February 12, 2007

Supervisor: Prof. Dr.-Ing. W. Rehm^{*}

Advisors: Dipl.-Inf. Torsten Mehlan^{*}, Dipl.-Inf. Torsten Hoefler^{**}

^{*} Chemnitz University of Technology, ^{*} Indiana University

The original (PDF) version of this document is available at:
<http://archiv.tu-chemnitz.de/pub/2007/0003/>

Keywords: monitoring, remote monitoring, local monitoring, cluster monitoring, cluster management, cluster, computer, cluster computer, network, network load, performance, scalability, Chemnitz High-Performance Linux Cluster, CHiC, OFED, InfiniBand, port counters, netgauge, Abinit, Nagios, plugin, plug-in

Worm, Stefan:
*Monitoring of large-scale Cluster
Computers*
Diploma Thesis, Chemnitz University
of Technology, 2007.

Abstract

The constant monitoring of a computer is one of the essentials to be up-to-date about its state. This may seem trivial if one is sitting right in front of it but when monitoring a computer from a certain distance it is not as simple anymore. It gets even more difficult if a large number of computers need to be monitored. Because the process of monitoring always causes some load on the network and the monitored computer itself, it is important to keep these influences as low as possible. Especially for a high-performance cluster that was built from a lot of computers, it is necessary that the monitoring approach works as efficiently as possible and does not influence the actual operations of the supercomputer.

Thus, the main goals of this work were, first of all, analyses to ensure the scalability of the monitoring solution for a large computer cluster as well as to prove the functionality of it in practise. To achieve this, a classification of monitoring activities in terms of the overall operation of a large computer system was accomplished first. Thereafter, methods and solutions were presented which are suitable for a general scenario to execute the process of monitoring as efficient and scalable as possible.

During the course of this work, conclusions from the operation of an existing cluster for the operation of a new, more powerful system were drawn to ensure its functionality as good as possible. Consequently, a selection of applications from an existing pool of solutions was made to find one that is most suitable for the monitoring of the new cluster. The selection took place considering the special situation of the system like the usage of InfiniBand as the network interconnect. Further on, an additional software was developed which can read and process the different status information of the InfiniBand ports, unaffected by the vendor of the hardware. This functionality, which so far had not been available in free monitoring applications, was exemplarily realised for the chosen monitoring software.

Finally, the influence of monitoring activities on the actual tasks of the cluster was of interest. To examine the influence on the CPU and the network, the self-developed plugin as well as a selection of typical monitoring values were used exemplarily. It could be proven that no impact on the productive application for typical monitoring intervals can be expected and only for atypically short intervals a minor influence could be determined.

Task of the Diploma Thesis

Today's cluster computers are used in different sizes. The number of small and medium installations with up to 128 nodes is growing continuously. At the upper end of the range very large clusters with several thousands of compute nodes were designed. The administration of such systems requires a supervising and monitoring system that can be used in those different kinds of parallel computers in a scalable manner.

The aim of this work is to analyse existing cluster monitoring mechanisms and to design extensions to these already existing tools. First the most relevant freely available solutions shall be considered. Thereby this thesis should especially elaborate how to detect irregular behaviour patterns or errors and moreover how to react on them. Furthermore, the scalability to a large number of nodes and the influence on other computation processes shall be analysed.

After the phase of analysing, missing functions and error cases should be identified. In addition convenient methods of statistical appraisal shall be found to identify possible accumulations of errors and as a consequence the detection of increasing probability of a failure of a component. Furthermore suitable behaviour patterns which can be defined as a reaction to errors shall be identified. It should be discussed in which way automatic interventions are possible. Hence, the achieved specification of an enhanced monitoring system should be adapted in the most relevant parts so that it can be integrated into the Chemnitz High-Performance Linux Cluster (CHiC). In addition, a special focus of this work shall be on the scalability and the smallest possible influence on other compute processes.

Theses

- I It is possible to run a cluster monitoring system with only minimum effects on user tasks.
- II A system that monitors only the necessary things is less error-prone and therefore easier to maintain.
- III A monitoring system that presents the information in a suitable way does relieve the administrators from unnecessary maintenance work.
- IV The outage of the cluster has to be minimised.
- V Most application runtimes (e.g. ABINIT) are not significantly influenced by standard monitoring activities.
- VI One server can handle the monitoring of hundreds of components, each of them with a set of values that has to be checked.
- VII The integrated monitoring of all components of a system, instead of the use of independent approaches for each of them, is to be favoured.
- VIII It exist no vendor independent InfiniBand network interface port counter monitoring software that supports established open source monitoring applications.
- IX The size and quantity of the network packets of the monitoring system have a measurable influence on the performance of the communication network.

Contents

List of Figures	ix
List of Tables	ix
Listings	ix
Abbreviations and Acronyms	xv
1 Introduction	1
1.1 Cluster Computers	1
1.2 Cluster Management	3
1.3 Summary	5
2 Cluster Monitoring	7
2.1 Monitoring as Part of Management	7
2.2 A Monitoring Model	9
2.3 Generation of Data	10
2.3.1 Local and Remote Monitoring	10
2.3.2 Communication Methods	10
2.3.3 Overview about Monitoring Objects	11
2.3.4 Performance and Scalability	11
2.4 Processing of Data	14
2.4.1 Data Validation and Storage	14
2.4.2 Combination of Monitoring Values	14
2.4.3 Filtering and Analysis	18
2.5 Dissemination of Information	19
2.6 Presentation of Results	21
2.7 Summary	22
3 Chemnitz High-Performance Linux Cluster (CHiC)	23
3.1 Introduction to the CHiC	23
3.2 Experiences from the CLiC System	24
3.3 Summary	29
4 Evaluation of Monitoring Approaches	31
4.1 Selection of a Monitoring Application	31
4.2 Nagios and the Plugin Topology	37
4.3 The InfiniBand Interconnection Network	38

4.3.1	Introduction to Design and Features	39
4.3.2	Constitution of the Port Counters	39
4.4	Design and Implementation of a Port Counter Monitoring Plugin	40
4.4.1	Preliminary Considerations	40
4.4.2	The <code>check_iberr</code> Script	42
4.5	Summary	44
5	Evaluation of the Performance Impact of Monitoring Activities	47
5.1	Introduction to the Test Configuration	47
5.2	Impact regarding the Execution of Applications	47
5.2.1	Influence on Abinit due to Local and Remote <code>check_iberr</code> Script Execution	48
5.2.2	Influence on Abinit due to Local and Remote Nagios Plugins . .	50
5.2.3	Influence of Local and Remote Nagios Plugins via IPoIB and GbE on Four Local Abinis Jobs	52
5.3	Impact regarding the Network Performance	54
5.3.1	Network Performance with and without Remote and Local Execution of Nagios Plugins via IPoIB and GbE	54
5.3.2	Network Performance with and without Execution of the <code>check_iberr</code> Script	56
5.3.3	Network Performance with and without Execution of Nagios Plugins Depending on the Delay of their Execution	58
5.4	Quantitative CPU and Network Load Analysis	59
5.4.1	Influence of Nagios Plugins on Clients and the Monitoring Server	59
5.4.2	Influence of the <code>check_iberr</code> Script on Clients and the Monitoring Server	60
5.4.3	Exemplary Monitoring Server Test with Nagios Plugins and the <code>check_iberr</code> Script	61
5.5	Summary	63
6	Conclusion and Outlook	65
A	Source Code Listing of the <code>check_iberr</code> Perl Script	67
B	Monitoring Server and Client Configuration	73
B.1	Definition of Hosts and Services on the Monitoring Server	73
B.2	Definition of the Check Commands on the Monitoring Server for Direct Execution	76
B.3	Definition of the Check Commands on the Monitoring Server for Execution via NRPE	76
B.4	Definitions on the Monitoring Client	77
	Bibliography	79
	Index	85

List of Figures

2.1	Correlation of Management and Monitoring	9
2.2	Object that is Controlled and Monitored	17
3.1	Exemplary Timetable as Basis for Alerting Methods	25
3.2	Monitoring of the Infrastructure with Corresponding Importance	27
5.1	Influence on Abinit due to Local and Remote <code>check_iberr</code> Script Execution	49
5.2	Influence on Abinit due to Local and Remote Nagios Plugins	50
5.3	Influence of Local and Remote Nagios Plugins via IPoIB and GbE on Four Local Abinis Jobs	53
5.4	Network Performance with and without Remote and Local Execution of Nagios Plugins via IPoIB and GbE	55
5.5	Network Performance with and without Execution of the <code>check_iberr</code> Script	57
5.6	Network Performance with and without Execution of Nagios Plugins Depending on the Delay of their Execution	58
5.7	Network Packet Size Regarding the Communication of Various Nagios Plugins and the <code>check_iberr</code> Script	62

List of Tables

1.1	Layers of Integrated Management and Functional Areas of Management	5
2.1	Monitoring Objects and their States	12
2.2	Errors of Event Classification	18
4.1	InfiniBand HCA port counters	40
5.1	Nagios Plugins	51

Listings

A.1	<code>check_iberr.pl</code>	67
B.1	Configuration of the Hosts and Services on the Nagios Server	73
B.2	Nagios Server Direct Command Execution Configuration	76
B.3	Nagios Server Command Execution Configuration via NRPE	76
B.4	NRPE Monitoring Client Configuration	77

Abbreviations and Acronyms

ACM	Association for Computing Machinery	— page 83
AES	Advanced Encryption Standard	— page 38
AMD	Advanced Micro Devices, <i>Inc.</i>	— page 23
AP	Access Point	— page 12
API	Application Programming Interface	— page 34
BLAS	Basic Linear Algebra Subroutines	— page 23
CHiC	<u>C</u> hemnitz <u>H</u> igh-Performance <u>L</u> inux <u>C</u> luster	— page iv
CLiC	<u>C</u> hemnitz <u>L</u> inux <u>C</u> luster	— page 23
CPMD	Car-Parrinello Molecular Dynamics	— page 24
CPU	Central Processing Unit	— page 11
DB	DataBase	— page 12
DDR	Double Data Rate	— page 86
DES	Data Encryption Standard	— page 38
DFT	Density Functional Theory	— page 47
DGEMM	Double-precision GEneral Matrix Multiply	— page 23
DHCP	Dynamic Host Configuration Protocol	— page 59
DMA	Direct Memory Access	— page 86
DNS	Domain Name Service	— page 12
DPS	Distributed Processing System	— page 13
FCAPS	Fault-, Configuration-, Accounting-, Performance- and Security- (management)	— page 4

FTP	File Transfer Protocol	– page 10
GB	<u>G</u> igabit (10^9 bit = $1.25 * 10^8$ byte = 125 MB)	– page 23
GBE	<u>G</u> igabit <u>E</u> thernet (network interconnect)	– page 47
GFLOPS	<u>g</u> iga (10^9) <u>f</u> loating point <u>o</u> perations per <u>s</u> econd	– page 23
GNU	<u>G</u> NU is <u>n</u> ot <u>U</u> nix	– page xii
GPL	GNU <u>G</u> eneral <u>P</u> ublic <u>L</u> icense	– page 87
GPU	Graphics Processing Unit	– page 12
GUI	Graphical User Interface	– page 21
GUID	(InfiniBand) Global Unique IDentifier	– page 42
HA	High-Availability	– page 2
HCA	(InfiniBand) Host Channel Adapter	– page 57
HDD	Hard Disk Drive	– page 11
HP	High-Performance	– page 2
HPC	High-Performance Cluster	– page 2
HTTP	HyperText Transfer Protocol	– page 12
HVAC	Heating, Ventilation and Air Conditioning	– page 87
HW	HardWare	– page 5
I/O	Input / Output	– page 12
IA	Intel Architecture (IA-32, IA-64)	– page 31
IB	InfiniBand	– page 23
IBA	InfiniBand Architecture	– page 39
IBTA	InfiniBand Trade Association	– page 81
ICMP	Internet Control Message Protocol	– page 11
ID	IDentifier	– page 11
IEC	International Electronical Commission	– page 4
IEEE	Institute of Electrical & Electronics Engineers	– page 83

IETF	Internet Engineering Task Force	— <i>page 11</i>
IM	Instant Message	— <i>page 20</i>
IMAP	Internet Message Access Protocol	— <i>page 12</i>
IP	Internet Protocol	— <i>page 12</i>
IPC	Inter-Process Communication	— <i>page 53</i>
IPDPS	(IEEE) International Parallel & Distributed Processing Symposium	— <i>page 81</i>
IPMI	Intelligent Platform Management Interface	— <i>page 23</i>
IPOIB	Internet Protocol over InfiniBand	— <i>page 50</i>
ISO	International Organization for Standardization	— <i>page 4</i>
IT	Information Technology	— <i>page 1</i>
JTC	Joint Technical Committee	— <i>page 81</i>
LAN	Local Area Network	— <i>page xv</i>
LB	Load-Balancing	— <i>page 2</i>
LID	(InfiniBand) Local Identifier	— <i>page 42</i>
MAC	Media Access Control	— <i>page 11</i>
MAD	(InfiniBand) Management Diagram	— <i>page 49</i>
MPI	Message Passing Interface	— <i>page 24</i>
MRTG	Multi Router Traffic Grapher	— <i>page 34</i>
MUA	Mail User Agent	— <i>page 89</i>
NFS	Network File System	— <i>page 12</i>
NIC	Network Interface Card	— <i>page 57</i>
NMS	Network Management System	— <i>page 33</i>
NRPE	Nagios Remote Plugins Executor	— <i>page 48</i>
NSCA	Nagios Service Check Acceptor	— <i>page 38</i>
NTP	Network Time Protocol	— <i>page 26</i>
OFED	OpenFabrics Enterprise Distribution	— <i>page 42</i>

OOB	Out-Of-Band	— page 48
OS	Operating System	— page 11
OSCAR	Open Source Cluster Application Resources	— page 84
OSI	Open Systems Interconnection	— page 4
PBS	Portable Batch System	— page 27
PCI.....	Peripheral Component Interconnect	— page 91
PCI-X.....	PCI eXtended	— page 39
PCIe	PCI <u>e</u> xpress	— page 39
PMEO-PDS ...	(International Workshop on) Performance Modelling, Evaluation, and Optimization of Parallel and Distributed Systems	— page 81
POP	Post Office Protocol	— page 12
POWERPC	<u>P</u> erformance <u>o</u> ptimization <u>w</u> ith <u>e</u> nhanced <u>R</u> ISC <u>P</u> erformance <u>C</u> hip	— page 31
PPC	PowerPC	— page 31
PSU	Power Supply Unit	— page 12
RAID	Redundant Array of Independent Disks	— page 28
RAM	Random Access Memory	— page 11
RDMA	Remote Direct Memory Access	— page 39
RFC.....	Request For Comments	— page 11
RISC.....	Reduced Instruction Set Computer	— page 82
RTT	Round Trip Time	— page 51
S.M.A.R.T.....	Self-Monitoring, Analysis, and Reporting Technology	— page 12
SDR.....	Single Data Rate (<i>InfiniBand network connection</i>)	— page 23
SIESTA	Spanish Initiative for Electronic Simulations with Thousands of Atoms	— page 24
SM	(InfiniBand) Subnet Manager	— page 40
SMP.....	Symmetric Multi-Processor	— page 23

SMS.....	Short Message Service <i>cellular phone text messaging</i> — page 20
SMTP.....	Simple Mail Transfer Protocol — page 12
SNMP.....	Simple Network Management Protocol — page 11
SSH.....	Secure SHell — page 10
TB.....	TeraByte (10^{12} byte $\approx 2^{40}$ byte) — page 23
TCP.....	Transmission Control Protocol — page 51
TDES.....	Triple DES — page 38
UDP.....	User Datagram Protocol — page 93
UPS.....	Uninterruptible Power Supply — page 12
VL.....	Virtual Lane — page 40
WLAN.....	Wireless LAN — page 12
XML.....	eXtensible Markup Language — page 33

1 Introduction

The semi-annual published Top500¹ list containing the five hundred fastest computers on earth fascinates computer interested people as well as the rest of the world every time anew. This topic is very attractive not only because of the very illustrative coverage in the media, especially the amazing comparisons with the first supercomputers like the one which was used for planning the moon landing, but also because of the abilities of recent ordinary computers or even with mobile phones which have about the same performance. This shall show that every person can have one of today's supercomputer some time in the future for himself and that it is only a matter of perspective what a supercomputer is.

In this work not the pure computing power is the main topic, it is the supervision of those computers. Because only with the help of monitoring it can be ascertained what is the state of the system and only with that information a system's administrator is able to keep it running properly so that a user can really benefit from the full computing power.

1.1 Cluster Computers

“When computing, there are three basic approaches to improving performance – use a better algorithm, use a faster computer, or divide the calculation among multiple computers.”[Slo05, p.4] Often the algorithms and their implementation are already optimised and cannot be made significantly faster – furthermore at a certain stage of the problem which needs to be calculated, the computing power of a fast computer is not enough or the price for it is too high.

In this situation, the use of the third approach is advisable. Having a minimum of two computers, is already a cluster because of its definition². But usually the term “cluster” in Information Technology (IT) refers to a large number of systems commonly at supercomputer size which means “The class of fastest and most powerful computers available.”[LJ93, p.275]

Still, having a bunch of computers is still not a computer in the sense of a super-computer-cluster. Referring to Sterling [Ste02], “[...] a cluster is any ensemble of independently operational elements integrated by some medium for coordinated and cooperative behaviour.” The most important issue regarding the definition is the need

¹The Top500 list is presented at the Supercomputer Conference twice a year since 1993.
<http://www.top500.org/lists/>

²*cluster*: a number of persons, animals, or things grouped together (refer to Webster's New World Dictionary of American English, Third College Edition [Neu88])

of something that makes the computers work together which means in this specific case at least two things: an interconnection network and some special software.

What kind of computers, network connections and cluster–software is used depends on the purpose of the cluster. Sloan [Slo05, p.11] distinguishes three types:

High-Performance Cluster This is the original type of cluster. At the very beginning when sets of computers were assembled to a cluster it was because of high-performance (HP) reasons. It was done with the intention to get more computing power than a single machine can provide, usually to solve a specific problem within an acceptable period of time. In literature the terms *supercomputer* and *cluster* usually refer to this type of cluster which is also the focus of this work.

High-Availability Cluster These types of clusters are made for scenarios where maximum reliability is necessary. It means that a service or application is available whenever it is needed, with only a minimum of downtime. High-Availability (HA) clusters can guarantee that because of their failover mechanism. It means that there is a set of computers which are doing their job and another set of spare computers which are running idle, just checking the working ones and waiting to take over if one of them fails. Respectively the “spare” computers are also working but they are able to take over the load of the other ones if necessary. Those clusters, because of their functionality, are also called failover clusters [Slo05, p.11] or Fault-Tolerant Clusters [Boo03]. They typically consist of only a few computers, often only of two machines and not of tens, hundreds or thousand like the High-Performance Clusters (HPC).

Load-Balancing Cluster This kind of cluster is very similar to the HPCs because they are also made for dividing the work among multiple computers, but with the difference that a Load-Balancing (LB) cluster is for providing a better real-time³ performance. For example a scenario where a web server, that cannot handle the traffic on its own or within an acceptable period of time any longer, is suitable to be replaced by a LB cluster.

Especially when thinking of a slightly different scenario, like processing a large amount of data in a short period of time from a physical simulation, this class of cluster can also be named High-Throughput Cluster (refer to [Luc04]).

The classification mentioned above is not precise. Depending on the problem that has to be solved, and especially the type of software and algorithms which are used, the cluster can be a combination of the different types.

In addition to a definition concerning the purpose of the cluster, the National Research Council of the National Academies [GSP05] further classifies supercomputer–clusters regarding their overall productivity:

³The term *real-time* means that a response to a query [the result of an operation] has to be given within a specified period of time, this can be a fraction of a second, but it does not have to be. (refer to [Dib02])

capability A capability cluster is used as a whole to solve a single, large problem – one that otherwise cannot be solved in a reasonable period of time [GSP05, p.24].

capacity The representative capacity system is one on which several and smaller computations are executed. The merit is a system that has good performance / cost value and can be used for a lot of domains.

1.2 Cluster Management

Working out what kind of cluster is needed, buying the hardware and building up the system is one thing – to keep it running another. It is necessary to have a strategy for managing all work related to the cluster computer. One of the first things that comes to one's mind thinking of *management* is probably the management of a company or the act of conducting or supervising something. In its general meaning, *to manage*⁴ something means “to handle or direct with a degree of skill”.⁵

In a more specific conception, management is every action inside an organisation with the focus to guarantee an effective and efficient way of operation.

Thus, management can be classified into five layers of integrated management referring to Hegering [HAN99] whereby every layer is based on the efficiency of the underlying ones:

5. enterprise management / service management (organisation of business processes, business services and policies)
4. management of applications (computer programs, distributed applications)
3. information management (all kinds of business data)
2. system management (server, workstation, printer, etc.)
1. network management (communication network, router, switches, etc.)

Integrated management describes the process of the seamless interaction of tools of every layer to cooperate and interact together. This is contrary to an isolated approach of management where a single tool is used for single problems especially of single management layers without interacting with one another [HA94]. Referring to this model, the overall integration is, of course, an ideal point of view. But nevertheless, the goal is to coordinate as much as possible within the system of management.

Based on the classification mentioned above and the limited focus of this work further considerations are dedicated to the two basic layers system and network management only. Obviously, the areas of enterprise and information management are primary non technical ones and therefore not interesting in terms of “Cluster Monitoring”. In

⁴manage: Italian *maneggiare*, from *mano* hand (refer to Webster's New World Dictionary of American English, Third College Edition [Neu88])

⁵Merriam-Webster Online Dictionary [man05]

addition to that the area of application management is touched only in few points so that a complete consideration of this is not necessary and would go beyond the scope of this work.

Furthermore, the classification of the five layers of integrated management is directed to the specific objects belonging to them. This is important in order to know *what* has to be managed. But it is at least as essential as this to know *how* all that can be managed. To achieve that, the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) has published the ISO/IEC 7498-4 standard “Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management Framework” [ISO89].⁶

In this document, the ISO has categorised the requirements for management functionality into five areas that are also known as FCAPS based on the starting letters of fault, configuration, accounting, performance and security:⁷

fault management A fault is the abnormal operation of a component that reveals as a particular event (e.g. an error). The management of it deals with its detection, isolation and correction.

configuration management This management part is responsible for collecting and providing information, as well as for the identification and control over components. It includes the initialisation and termination as well as the provision of continuous operation of the system.

accounting management Accounting is the recording and summarising of actions with the intention to analyse, verify and report them for being able to charge for the use of resources.

performance management The management of performance is the evaluation of the behaviour and the effectiveness of resources. It is used for gathering statistical information for tuning and sizing them, as well as for reporting reasons.

security management Principally this is the support of applications’ security policies. It has importance in the secure implementation of management tasks, the detection of security violations and maintaining security audits, as well as the creation, deletion, and control of security mechanisms.

Those five functional areas of management explicitly apply to each of the five layers of integrated management [HAN99] that are mentioned above. It means the classification in functional areas is orthogonal to the classification in layers.

Because of the limited focus of this work, only the intersection of system and network management with fault and performance management is considered (refer to Table 1.1 on the facing page). Apparently, the management of security [Wor05] and

⁶This is the fourth part of the well known ISO/IEC 7498-1 standard [ISO94] (refer to [Tan03]), in which the seven layer Open Systems Interconnection (OSI) reference model is defined, that is the abstract description for communications and computer network protocol design.

⁷An interpretation of this standard can be found in [Lan94, CS92, Lib00].

enterprise m.					
application m.					
information m.					
system m.					
network m.					
	accounting m.	configura- tion m.	fault m.	perfor- mance m.	security m.

Table 1.1: Layers of Integrated Management and Functional Areas of Management

configuration is important and big areas of interest themselves, but they are too extensive and the relevance compared to the other functional management areas is too small to be explained in detail in this work – nevertheless annotations regarding those aspects were given in a short way if possible. In addition to that, accounting management is also not the main focus of this work, because at a cluster system its usage is important not the billing for it. – An explanation of the exclusion of the management layers was mentioned above.

1.3 Summary

In this chapter, a short introduction to cluster computers was given. It was shown for what purpose those kinds of computers are needed. Moreover, a classification of cluster computers into one of the categories of *high-performance*, *high-availability* and *load-balancing* was presented, as well as an alternative approach of cluster computer classification based on *capability* and *capacity*.

Then, a general overview of the field of management was given. A five layer classification of management with the people in suits on top, down to the “technical” management of computer hardware (HW) was shown, with the further focus on system and network management. In addition to that, five functional areas of management that are adaptable to the management layers mentioned above were introduced, whereupon fault and performance management were chosen for detailed considerations in following chapters.

2 Cluster Monitoring

In Section 1.2 on page 3 an introduction to the management of clusters was given. It describes all the procedures that have to be done beginning with installation, over maintenance to the shut down of the system. There are decisions to make *how*, *when*, *where*, *what*, etc. to do with the cluster. To get a basis of information, necessary to make management decisions, it requires monitoring. Based on a useful business axiom “if you can’t measure it, you can’t manage it” [LH02] it means that it is essential to get an overview of the state of the cluster before it can be run in a useful and efficient way.

The verb *to monitor*¹ means “to watch and check on a person or thing” [Neu88], in a more technical definition it means “to check on or regulate the performance of a machine” [Neu88]. For this it is necessary to measure – to realise performance measurement. In this aspect *performance* is always related to what is intended. Based on a specific task, the performance of a system can be anything from *none*, it cannot perform at all, to *it has optimum performance*, in the way that it uses the existing resources as good as possible.

Finally, referring to Joyce et al. monitoring can be defined as the process of collection, interpretation and presentation of information concerning objects [JLSU87].

2.1 Monitoring as Part of Management

As mentioned in Section 1.2 on page 3 there are various fields of management that can be categorised in different ways. Furthermore, it was also emphasised in the preceding section that monitoring is the basis for management. Thus, monitoring is essential for every kind of management and every kind of management requires its own type of monitoring. Therefore management is not possible without proper monitoring. Based on this conclusion, it is permissible to substitute the word *management* by *monitoring* in Table 1.1 on page 5 and with its new meaning it also describes the kind of monitoring that is focused in this work.

There are two primary types of monitoring regarding their purpose [LH02, CWSC01] where each of both belongs to its correlative management area (refer to Section 1.2 on page 3):

real-time monitoring Also known as event or fault monitoring.² It watches every unintended, unexpected change of the systems state, this could be a check for a

¹monitor: past participle of the classical Latin “monere” which means “to warn” [Neu88]

²Although fault monitoring is the correlative of fault management, the term real-time monitoring is used because it describes its task in a better way.

subsystem's outage or the exceed of preset threshold values. The key premise is the permanent check of the system and the instant processing and dissemination of the information (refer to Section 2.4 on page 14 and Section 2.5 on page 19), which can be, for example, the immediate alerting of a responsible person in case of an unexpected behaviour.

It allows only *reactive* actions related to the systems state. Consequently, an action can be taken only after an event has occurred, which usually means that the unwanted behaviour has already appeared.

historical monitoring Another term is performance monitoring³, the result of its job is the automatic generation of (long-term) statistics, for example for *availability, utilisation and throughput*. First, a system performing historical monitoring collects the data and stores them. After this, the gathered data can be used to generate graphs that usually show the dependency between values, e.g., the system's performance over time, or for the detection of problems that occurred in the past.

It is the basis for predictions of the systems behaviour in the future and possible actions that can be taken to prevent system outages (*proactive* actions). For example for the prediction of future resource demands.

A classification in two categories, like mentioned above, does not mean that the two types of monitoring are incompatible to one another – indeed the gathered data of a real-time monitoring system can be used as the input for historical monitoring. But it has to be kept in mind that both systems have their own purpose and that a simple integration would not lead to a satisfactory solution, for example the storing of all real-time monitoring data without any concept would lead to a huge data grave instead of convincing historical monitoring statistics.

Finally the question is posted, where does monitoring end and where does management begin. Although both terms are sometimes used to describe the same thing and even appear interchangeable for some reasons, it is possible to make a distinction. Monitoring can be defined as the process of gaining information about an object and management as the process of making decisions regarding the object based on the monitoring information, as well as the control of the object. The control loop between monitoring, management and the object that is managed and monitored is shown in Figure 2.1 on the next page (refer to [MSS94]).

This distinction is important to define what type of action belongs to monitoring and what belongs to management. Regarding the definition mentioned above, additionally, there is the scope of direct processing of the monitoring data that can be seen either as a task of the monitoring system or of the management system. It depends on whether monitoring is defined as the pure gaining of information of an object or as a system that supplies the management system as good as possible.

³Though performance monitoring correlates to performance management, because of its better description of what is meant with it, the term historical monitoring is used further on.

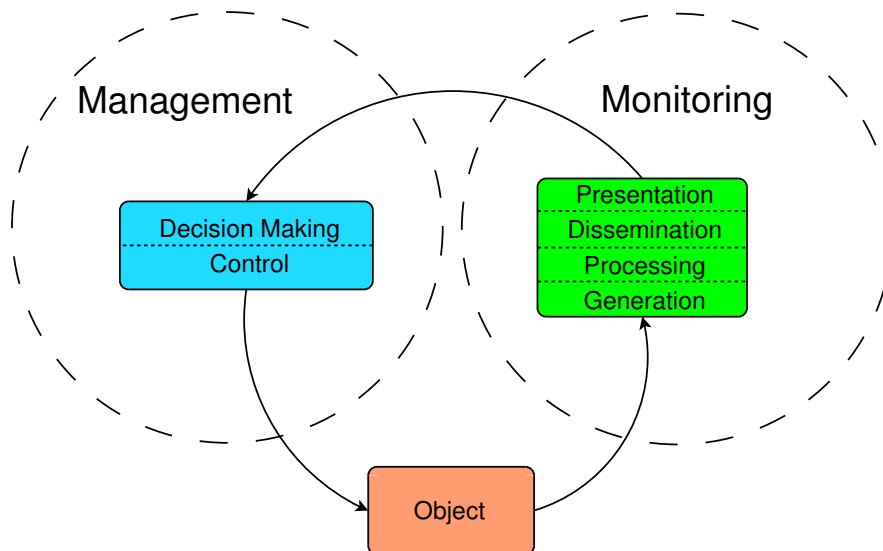


Figure 2.1: Correlation of Management and Monitoring

Basically, because of the approach to reduce the complexity of the anyway complex management system and the approach to reduce the load on the whole system while performing various tasks as early as possible in the processing chain, a monitoring system with some intelligent behaviour is described in this chapter.

2.2 A Monitoring Model

A generally functional model which can be used to describe activities of an efficient monitoring system, is presented in [MSS94] as the four areas of Generation, Processing, Dissemination and Presentation.

- *Generation*: Detection of events and generation of event and status reports.
- *Processing*: Providing of common functions for data processing such as validation, combination, correlation and filtering. The goal is to convert the raw and low-level monitoring data into appropriate data structures and a certain level of detail.
- *Dissemination*: The distribution of monitoring reports to people (users, administrators, managers) or software (management SW, cluster batch system, etc.) that require them is organised.
- *Presentation*: Displaying of gathered and processed monitoring information to the user in an appropriate form.

In the following sections, the parts of the monitoring model are described in detail with the focus on monitoring cluster computers.

2.3 Generation of Data

The beginning of the monitoring process is to generate the monitoring information. This can be done in several ways. Simply by performing several actions manually from time to time [Sel00], for example *ping* the machine, login via *SSH* to perform some commands like *df*, *ifconfig*, *top*, *ps* and others, or check the system's services by fetching a website, downloading a file via *FTP* and so on. Although this kind of "monitoring" is very common among administrators supervising a small set of computers, it is not as seldom as expected also for large computer sets. Nevertheless, for a cluster computer the better way is to use tools that generate monitoring data in an automated way. Therefore the approach of a centralised monitoring server is analysed, which is supported by specialised software modules for monitoring, or the status and event logs of a third party (software-)system.

2.3.1 Local and Remote Monitoring

Based on the location where monitoring is performed, it is differentiated between local and remote monitoring. The method of **local monitoring** means that the monitored object itself performs the necessary actions for getting its own status information and that it is responsible itself for all monitoring activities. This can be done by scripts or programs that are executed periodically for example with the help of the *cron* service, by a specialised, permanent running monitoring daemon, or by a dedicated hardware system like a special *service CPU*. The gained information can, but does not have to be sent (*pushed*) periodically, or if necessary, to a system that collects data from local monitoring systems to store and further process them.

The counterpart of local monitoring is **remote monitoring**. The term *remote* means that the monitoring supervision of the system is mandatory performed from the outside by a dedicated monitoring server which is driven by a kind of policy. This could be realised completely externally for example by passively analysing the network traffic or by actively checking the host's status. For this reason a local monitoring daemon can be installed too, but with the difference that it is dependent on the monitoring server and controlled by it. The communication between them can be organised by *pulling* (also: *polling*), which means that the server periodically demands information (*probe*) from the monitored object that thereon replies the requested information. If necessary, the monitored objects send an *alert* (also: *trap*) to the server in addition.

2.3.2 Communication Methods

In addition to the principles where monitoring is performed, the two principles have one thing in common: a kind of communication between the monitored object and the monitoring server. This can be done in various ways.

A very common practise is the use of the Simple Network Management Protocol

(SNMP)⁴ standard [MS01], that offers a standardised and flexible method mainly for network devices. It has not only been developed to meet monitoring but also management requirements.

Especially for network status concerns the use of Internet Control Message Protocol (ICMP) specified in RFC 792 [RFC81] is suitable [Hal00]. For example network connectivity and performance can be monitored by actively using *ping* messages or passively record ICMP error messages like “destination unreachable”.

Among other methods like remote execution of commands which was mentioned above, the communication protocol of a specialised monitoring client is usually realised in a very specific implementation. Not the interoperability of those implementations is the main topic, the focus typically lies on the lowest possible impact on the performance of the involved monitored components, and a minimum network load and the possibility to fit the communication method to the specific needs of dedicated monitoring systems best (refer to Section 2.3.4).

2.3.3 Overview about Monitoring Objects

At this point, the kind of real objects that are meant for *system and network* monitoring in combination with *fault and performance* monitoring are presented. With the term “objects” every kind of things that can be monitored or that are involved in the process of monitoring are described. See Table 2.1 on the next page for a brief overview.

Apart from the objects whose values can change, not mentioned in Table 2.1 on the following page are all the objects that are static, for example CPU ID, CPU type, HDD vendor, HDD size, RAM size, OS version, MAC address, etc., because the focus of monitoring is on the objects that are applicable for finding out if something is working as it should do or not. Although those object values are also retrievable and usable for the monitoring system, they are more interesting for configuration management (see Section 1.2 on page 4). Further on, the non-requesting of the values mentioned above via the monitoring system also avoids network traffic.

2.3.4 Performance and Scalability

The **performance** of a system describes the ability how well it carries out an action or pattern of behaviour [per05]. A statement regarding a system’s general performance is often a relative value, either compared to another system or the practically measured performance value compared with the theoretically expected one of the system. Only based on the results of the system’s comparison mentioned above, a statement like *poor*, *good*, *optimum* regarding its performance can be made. Obviously, the performance should always be the optimum, which usually means to use the full power of a system, because of the reason not to give away the expensively purchased performance, e.g., of a cluster computer.

⁴The SNMP standard consists of various Request for Comments (RFC), the details of the actual SNMP version 3 can be found in RFC 3410 to 3418 at the Internet Engineering Task Force (IETF) website – <http://www.ietf.org/rfc/>.

classification	objects	states to be monitored
specific hardware devices	wireless local area network (WLAN) access point (AP)	working?
	printer	paper and toner fill level
standard computers	computing centre	temperature
	rack	temperature, water throughput, fan
	network switch	working?, performance
	hard disk	free capacity, S.M.A.R.T. values (<i>temperature, defect-free?</i>)
	CPU	utilisation, temperature
	RAM	utilisation, defect-free?
	network controller, network connection	working?, performance (<i>speed, throughput, latency, bandwidth</i>), IP address, DNS name
	graphics card	temperature of the GPU
	fan (<i>mainboard, PSU, chassis</i>)	rotation
	power supply unit	supply voltage
software processes	Uninterruptible Power Supply (UPS)	charged?
	mainboard, chassis	temperature
	operating system	uptime, load, processes, I/O, log files (events, errors), user accounts, software licences
	services (<i>HTTP, FTP, DB, DNS, e-mail (POP, SMTP, IMAP), file server (NFS)</i>)	availability, correct execution, where appropriate: number of connections, response time, queue, etc.

Table 2.1: Monitoring Objects and their States

The problem is that it is anything but trivial to manage the system in the way that it exhibits optimum performance – as mentioned above the basis for this is monitoring. For example if one of the many computing nodes of a cluster computer goes down for some reason, the performance of it also does. To recognise such situations, a typical approach is to monitor the node directly. As mentioned above in Section 2.3.1 on page 10 there are various possibilities at what point of the system monitoring can be performed.

One thing they all have in common is that various resources have to be used which otherwise could have been used by the productive applications. Usually the highest impact is seen at the system's CPU and network interconnect. If no extra network interconnect and no additional monitoring hardware are installed, the resources of the productivity system have to be also used for monitoring tasks. It means that unfortunately the process of monitoring also consumes a fraction of the overall performance of the entire computing system.

Therefore the goal is to keep the influence of the monitoring actions on the system's performance as low as possible. Especially the impact of the monitoring activities on the system's CPU and the network interconnect has to be minimised. In order to reach that goal a careful selection has to be made of the objects that *have to be* monitored and the objects that *can be* monitored (refer to Table 2.1 on the preceding page). Sometimes the more likely disadvantageous proceeding is made by trying to monitor everything that is possible, because at first sight every object appears worth it.

Unfortunately this does not increase the chance of getting to know about an important event. In contrast, without appropriate filtering and processing this approach lowers it, because of the flooding of information the chance of getting to know about the really important ones gets lower. The lower the number of objects that have to be monitored the lower the performance impact. Furthermore the impact of the monitoring system itself regarding its architecture, monitoring principles and rules, implementation and especially the frequency of executing monitoring activities is also an issue that has to be paid attention to [SDA⁺00].

From this it follows that there is a trade-off between the performance loss because of the execution of monitoring functions and the risk of not getting to know if something is wrong with the cluster, that therefore also could lead to a significant performance loss.

Another issue that affects the performance of monitoring and therefore also of the cluster computer is the **scalability** of the monitoring system. The scalability describes the capability of a system being easily expandable or upgradeable on demand [sca05]. For a monitoring system it means that a small installation of a few monitored objects could work well but for a huge number of objects like on a large cluster computer the same monitoring approach could have that much influence on the productive system that it needs a large fraction of its performance for monitoring only. This is of course unacceptable, hence a monitoring system must be examined regarding its ability to work well with a large number of monitored objects. A big influence on this problem has the type of application that runs on the computer cluster and the period of time between the execution of two monitoring actions.

For example when a distributed application⁵ runs on a cluster computer and has to do a lot of communication work among each of its components, additionally there is a monitoring application that shares the CPU and the network connection with the distributed application, in a special case an uncoordinated execution of the two applications can lead to a significant slowdown of the productive application [PKP03]. Although the outlined scenario above is a very special one it shows that the configuration of the monitoring system is essential for the scalability of the system.

Another aspect of the scalability is the ability of the system that controls all monitoring activity to handle a large amount of monitoring data. Based on the monitoring data this system needs an appropriate network connection, an adequate data storage space

⁵ A distributed application, based on a distributed processing system (DPS), consists of several autonomous parts which interact in order to cooperate to achieve an overall goal by coordinating their activities and exchange information by means of communication systems. [SK87]

and reasonable computation power or a distribution of the monitoring functionality itself or by using a load-balancing cluster (refer to Section 1.1 on page 2).

2.4 Processing of Data

In the previous section the generation of monitoring information was discussed. In this section common processing activities that can be performed on this information are considered. Note that these processing functionalities are often integrated and are performed in different places and at various stages.

2.4.1 Data Validation and Storage

First of all the generated monitoring information (Section 2.3 on page 10) has to pass validation and plausibility tests to make sure the system has been monitored correctly. This may be performed on different levels. When the monitoring information is examined, it is tested for example whether the identification number (ID) is the expected one or if the time-stamp is valid. Invalid reports are discarded.

A different class of invalid values are those out of the defined range of a certain value and which appear obviously incorrect for a human, for example if the CPU load is more than 100 percent or the rotation of a fan is twice the value of its absolute maximum. The origins of such values, e.g., a wrong measurement itself, are various and the treatment of them depends on the analysis strategy. Those values can be generally ignored or they can be stored and analysed further, depending on if they occur only once, occasionally or frequently to extrapolate appropriate reactions towards this.

The monitoring data is stored in a database to have a current status of the system, because it is used to access it for further analysis of the data later on, especially for detecting component failures or for concerns of the management system. The data is stored separately for real-time or historical monitoring reasons (refer to Section 2.1 on page 7) or the real-time data can be converted for historical monitoring by summarising the data with a specific procedure to master its volume.

2.4.2 Combination of Monitoring Values

The combination describes the real analysis of the monitored data. Up to this point the generated, validated and stored monitoring information are completely uninterpreted. Not until the measured values are put into relation with the expected ones a statement about the system's status can be made.

A simple approach to monitoring is to find out if a system is healthy or not. But without the definition what the “right” status is, this question cannot be answered. Thus, it has to be defined what an “error” is.

Events

The measured monitoring information can be classified into various types of *events*.

- alright/okay (no problems)
- unknown (no classification possible – the monitoring value cannot be classified otherwise)
- warning
- critical (an error has occurred)

But the definition to which class an event belongs is the administrator's concern. Not before the administrator has defined what an expected value is and what an unexpected value or behaviour of an object is, a statement regarding an event's classification can be made. Thus, this is the prerequisite to make intelligent appearing statements like “*the temperature of the CPU is too high*”.

The conditions for the classification of the events can be different. The simplest way is that the monitored value is compared with one or more predefined ranges in which it fits best – resulting from this the classification is made. For example if the temperature of the CPU is below value x_1 it is an *okay event*, above value x_1 and below value y_1 it is a *warning event* and above value y_1 it is a *critical event*.

In a more complex situation the measured value is additionally compared with preceding values of the object or with values of another object to perform the event classification. For example if the CPU temperature is between value x_1 and y_1 (*warning*) for z hours the administrator can define a rule that classifies this as *critical*. Regarding the comparison of values, it has to be paid attention that also the missing of the present or preceding ones can lead to a reaction. This can be for example the classification as a *warning* event immediately after the missing of just one value or after some time if a few are missing, or for example it can be classified as a *critical* event if the missing of a value remains persistent, depending on the configuration.

But the previous situation can be expanded even more to model more complex situations. The reason for the further enhancements of the classification process is due to special requirements.

First, the user of the monitoring system could demand the monitoring of very special situations or combinations of events that are not scheduled in the normal monitoring rules.

Second, wrong measured values should be excluded by comparing for example values that are directly or indirectly related to each other, such as the comparison of the temperature of the CPUs, of the chassis and of the rack to eliminate situations like the following one. The temperature of the rack rises, but the CPU and chassis temperatures of all the computers in that rack stay almost the same. If only the rack's temperature would have been measured, a possible conclusion would have been that the cooling system has a problem and the classification would have been *critical* for instance. But knowing about all temperature implies the more possible conclusion that there is only

a problem with the measurement of the temperature of the rack, as the computer work well, therefore the classification would be *warning* only.

Third, it is possible to make more general and abstract statements of the system's state by correlate several values to get just one value that for example expresses that one computer is without any problems. If all measured objects of the computer indicate that there are no problems at the moment, or the computer is accessible via the network, the infrastructure (network switch, network connection), for instance, is alright as well. The main reason for value summarisation is to ease the work of the administrator by reducing the classification of events in either *warning* or *critical*, because usually a lot of effort is involved to handle these. Another possibility for a better classification is the performing of additional monitoring actions to confirm or rebut a measured value where qualified doubts about its validity exist. For instance, the additional measuring of the CPU's load can test if the possibly too high value of the CPU's temperature is caused by a lot of work the CPU is doing or because of a defective temperature sensor.

The monitoring system is able to perform this action because the monitoring module for a specific value is already available on the system which is monitored. The monitoring system can simply access this value if necessary – the execution of general purpose commands in this context is not necessary by the monitoring system. Moreover its execution also would not have been allowed, because of the separation of monitoring and management tasks regarding a specific object (refer to Figure 2.1 on page 9), the direct control of it is part of the management only, refer to Figure 2.2 on the facing page. This is also important for the field of *proactive management* which handles the execution of commands on an object. It can be the installation of additional software like another monitoring module (refer to Section 1.2 on page 4 – *areas of management*) or just the execution of a command that takes influence on the monitored object [CS92].

For instance a high load, almost 100 percent, on a specific computer that is not working on a cluster job may occur from time to time. The reason for this was figured out – it was caused by a program that sometimes misbehaves and because of this consumes almost all processing power for nothing. The newest version of this program is already installed and there is no replacement for this program, which means that only the effects of it can be treated. Thus, a possible solution would be to implement that if a high load on the specific computer occurs, an additional measuring verifies if it actually is due to the program that sometimes misbehaves, if yes the process of the program is terminated and it is restarted. The solution is suitable for this situation only and this kind of solutions should be used only rarely, because there is a high danger that something goes (automatically) wrong and it can cover up problems instead of solving them. To encounter this risk, at least a notification of the administrator should be made after the performing of the solution of the problem (refer to Section 2.5 – *Dissemination*, p. 19).

Another use of the summarisation of events is the avoidance of event flooding. For example if a network switch breaks down, only one event is generated and not one for every single computer that is affected by this situation [LH02]. In addition to that, in a situation where the measured value of an object wobbles slightly around a threshold value in a fast manner, for example at the stage to *warning*, the generation of one event for every exceeding of it is not desired.

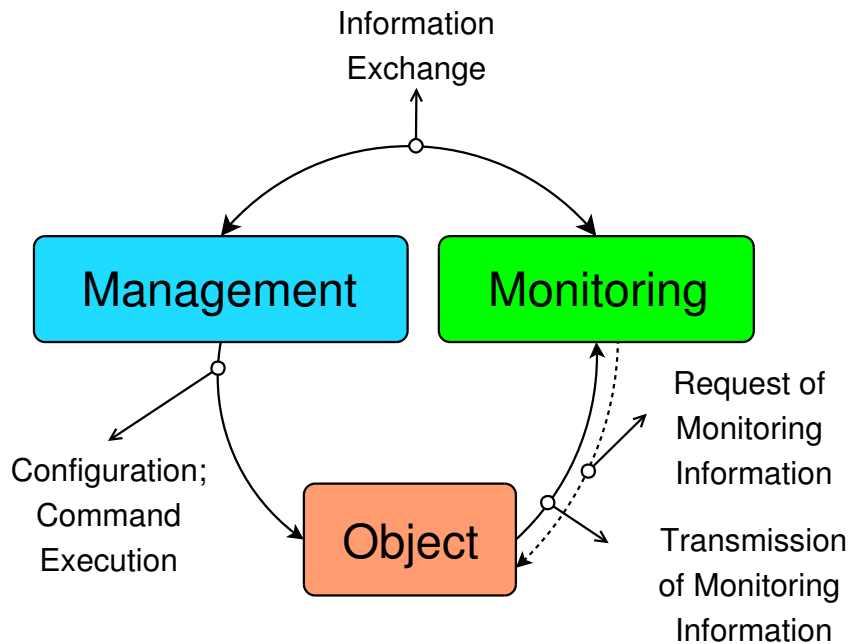


Figure 2.2: Object that is Controlled and Monitored

The configuration of the scenarios described above requires the foregoing modelling of them by the administrator of the monitoring system as mentioned before, as well as the support of the monitoring system for this. It means, that it has to have the possibility for example to record the representation of the network regarding the situation mentioned above where event flooding has to be avoided if just one component fails but many others are affected.

The classification and combination of the monitoring information is essential to keep a clear view of the system. The increasing level of abstraction associated with this approach prevents the users of such information from being overwhelmed by the considerable volume of information. The separation of events and their associated reactions is for reducing the complexity, so that it is not necessary to define appropriate reactions on every single monitored value – the aggregation of them allows the definition of a set of reactions for the event classes only. Any appropriate reaction to this is part of the dissemination of the monitoring data (refer to Section 2.5 on page 19).

Although the reactions are not directly on a single measured value but on an event class the information, of what the reason for the event was, is put through to the object or person which handles the reaction to it in the end.

Measurement Errors

The classification of the measured values in event classes is not without errors. Ideally there is a measured value that really is a specified event type (e.g. *warning*) and it will be classified accordingly. It is the same situation if a measured value does not belong

to a specified event type and is not classified as this, too. This seems trivial, but it is not. There are situations conceivable where this is not correct (see below). If the two situations mentioned above occur, everything is alright and it is the way how it should work (refer to **true positive** and **true negative** in Table 2.2).

Nevertheless, the following two situations are more important, because something has gone wrong if they occur. The **false positive** (also: type *I* error or α error) is the result of a classification which describes that the measured value is not of the specified event type, but it is wrongly classified as this type. Thus, a measured value would be classified as a *warning* although its true nature does not fulfil the requirements for the event class. Further on, a **false negative** (also: type *II* error or β error) classification describes that the measured value is of the specified event type, but it is not classified as this type. Thus, although for example a measured value should have been classified as a *critical* event it has not been.

		the true nature of the event	
		<i>it is the specified type</i>	<i>it is not the specified type</i>
classifi- cation of the event	<i>as the speci- fied type</i>	true positive	false positive
	<i>not as the specified type</i>	false negative	true negative

Table 2.2: Errors of Event Classification

The importance of the topic is to understand that a monitoring system itself can never be without any errors, so it cannot be perfect. There are always errors that can occur, mainly that a value is not classified as a special event although it is or vice versa. In consequence this can lead to events that are not handled although they should and then perhaps cause big trouble regarding the monitoring purpose. For example a bug in the software of the I/O server can lead to a misbehaviour of it, so that it writes data to the storage system which causes damage to the stored information and that needs to be stopped immediately. In the other error situation it bothers the administrator unnecessarily if a lot of false alarms occur.

The reasons for the wrong event classifications are mainly systematic errors like insufficient classification rules, wrong measurement interpretation, general errors in the software that processes the classification, etc. or random errors like in the measurement itself and in the measurement data transmission [Kan02].

2.4.3 Filtering and Analysis

The filtering of the information is primarily for their reduction on all levels of the monitoring system due to the amount of data that has to be generated, processed, disseminated and presented. The filtering on a stage as early as possible of the monitoring process reduces the work for the stages that follow. For example it reduces the CPU and network influence on the cluster computer best by ascertaining only the actually necessary data. Also the validation of information (refer to Section 2.4.1 on page 14) is

a kind of filtering that pursues the goal of information reduction, as well as the control of the dissemination of information (refer to Section 2.5), for example that the monitoring reports are only sent to the person who is interested in them. The criteria on which the filtering is performed, are defined as a part of the processing rules and are also based on the requirements that are defined by the administrator of the monitoring system.

Another important issue of the processing of the monitoring information is the analysis of them. The analysis can be the main purpose for example of a historical monitoring system (refer to Section 2.1 on page 7) that usually has specialised functions for this. But it can also be useful for a real-time monitoring system (refer to Section 2.1 on page 7) that can benefit, e.g., from the ability to determine the average or a mean average of particular status variables, forecasting faults in components and the possibility to get some statistics that allow a clear view of the system's state [MSS94]. Often, for instance it is interesting to have statistics such as the total CPU usage, idle and busy times, the amount of data sent, etc. Another use of the analysis can be the supervision of the monitoring system and its process itself. For example with the help of the administrator that classifies all false alarms on their occurrence, statements regarding the system's quality of the classification of events can be made.

2.5 Dissemination of Information

Monitoring reports that are the result of the monitoring information processing have to be forwarded to different users of such information. The destination of such reports may be human users, the management system, other monitoring objects or processing entities. It is based on the approach to disseminate only the really necessary information to avoid a big workload on the monitoring system and to ensure that the receiver of the information gets only those that are interesting.

The following reactions on events are conceivable, based on the classification of them (refer to Section 2.4.2 on page 15) and the principle of separating the event's classification by their reaction to it.

1. forwarding the information to other systems
 - event log database
 - further processing instances
 - the management system
 - presentation module (refer to Section 2.6 on page 21), change the object's status in the monitoring system if necessary – information available for pulling
2. inform user or administrator – by pushing information
 - a) e-mail

- b) cellular phone text message (SMS)
- c) instant messages (IM) or playing a sound file if the person is working at the computer
- d) phone call

A dissemination of the information to other systems always happens. Either the receiving objects and instances have rules to handle the event information or they simply discard or ignore them. But the main task of the dissemination module is the handling of the information that is sent directly to the human user. This is done only if the information is of such importance that it needs the attention of a person. Which information, respectively event classes are chosen for this is part of the configuration of the monitoring rules, as mentioned above. An exemplary situation could be that if a *warning* event occurs an e-mail is sent and if a *critical* event occurs an SMS or IM is sent.

Further on, the dissemination module is responsible for the enforcement of the rules that handle the *Who?* is informed, depending on the *When?*. It means, that only the responsible person or a group of persons (*Who?*) will be informed at a certain time (*When?*) to ensure that the person is really able to handle the information and that this person is in duty or stand-by duty and not on holidays so that he or she can really deal with it. It has to be ensured as well that at every time and for every event the right destination is addressed. This is part of an escalation procedure in the handling of the dissemination of monitoring reports.

The procedure of **escalating** a problem means that it is tried to counteract on a perceived discrepancy, for example that a problem gets bigger and bigger if not treated and that it is levelled up from *warning* to *critical* if it lasts to ensure the treatment of it for instance. But if it is a *critical* event already, there are two possibilities to ensure that treatment of it takes place. One is to define subclasses of an event class and to perform different actions based on the level of sub-classification. Another one is to handle the escalation of the treatment of monitoring events with the help of the dissemination procedure itself. In a reliable monitoring system it has to be ensured that there is a reaction to an event that has occurred.

This can be done by the receiver of the event notification by sending an answer to the monitoring system that the processing of the event is under way. The escalation of all monitoring information finally end at a person, because if every preceding classification and the therewith associated actions, like automated execution of programs that should fix something, fail, a person is the last instance that can handle a problem. Thus, the escalation of the treatment of a problem if it is disseminated to people is very important.

One possibility to do this is for example the rule that a person flags the problem he or she starts working on. This can be done by sending a reply message on the received e-mail, SMS, etc. or by marking the problem as *working on* at the front-end of the monitoring system. If the flagging does not happen after a period of time or the status does not change although the problem is flagged, its treatment is further escalated. Furthermore it is unaffected by the possibility to flag the problem as, e.g., *not solvable*

until further notice or even *broken* if there are conditions that cannot be fulfilled at the moment and that thereon no further escalation is carried out.

Finally, the escalation of the treatment of a problem means that another person which can be the member of the team or someone from outside (horizontal escalation) is informed, or a person that has more experience or the superior or even his or her superior is informed (vertical escalation), or that it is tried to get through to a specific person by means of different communication methods, for example e-mail first, text message (SMS) second and automated phone call as the last resort.

2.6 Presentation of Results

The final step in monitoring is the presentation of the information as the result of the preceding generation, collection, processing and dissemination of data.

The way in which this can be done depends on the physical device the administrator is using, for example a computer display, a mobile phone display or a voice interaction system. Additionally it also depends on the requirements of the user to choose the appropriate presentation form. This can be a textual representation in a system console, short message or e-mail, as well as a graphical representation with the help of a Graphical User Interface (GUI) or a specialised display system. Thus, the presentation module has to control the amount of monitoring data, the levels of abstraction of such information and the rate at which this information is presented.

The main focus of the presentation of information has to be on the *usability* of the system which means the ability of convenient and practical use of something [MAS⁺03, Nie04]. It means that the quantity of information, the detail of information and the time interval in which the information is presented depends on the specific requirements of the user and the presentation system and is essential for the success of the monitoring strategy. Only if the user of such information is not overwhelmed by irrelevant information a proper reaction to it is possible.

Thus, user-friendly techniques for instance at the presentation device computer monitor is desirable, for example the grouping of the information for a better overview, comprehensible event messages especially for e-mail or adjusted display format with weighted lists (e.g. tag clouds [MHS06]). The use of weighted lists can also solve a typical problem in information presentation, that is to find out which of the *critical* events is the most urgent one. It means, the system administrators “have to be able to tell at a glance which of the “red” issues is the “reddest” and having the most impact” [LH02, p. 519]. The condition on which a very urgent event is highlighted with a larger font or the top position of a list, has to be an additional value which could be the position in the dependency hierarchy, a manually predefined one or a dynamic one such as the number of events over a fixed period of time. Thus, events may be displayed in their *causal* rather than *temporal* order.

Nevertheless, especially for historical monitoring the use of two dimensional diagrams, with one axis representing a specific value and the other representing time is the most common method to show the changes over time.

2.7 Summary

In this chapter, the relation between management and monitoring has been shown. The distinction between real-time monitoring and historical monitoring as the correlative of fault management and performance management was made and the different requirements of both on a monitoring system were presented in detail.

Furthermore a model that describes the procedure of monitoring as the four areas of generation, processing, dissemination and presentation of data was introduced and explained in more detail.

In Section 2.3 “Generation of Data” the distinction between local monitoring and remote monitoring was made, an overview about objects and their states was given, in addition to that some remarks were made about building up a preferably high-performance and scalable system.

The Section 2.4 “Processing of Data” discusses various types of classification approaches to classify the measured values in event categories for further processing, as well as the appearance of measurement errors.

In the last two sections, first the dissemination of the monitoring results to the objects or people that use them with the help of an appropriate escalation procedure and second the presentation of them on various devices with the focus on their usability was discussed (refer to Section 2.5 on page 19 and Section 2.6 on the previous page).

Finally, this chapter has given a comprehensive overview about cluster monitoring in general and about topics on which it has to be paid attention to, in particular to be able to understand the requirements of such a system.

3 Chemnitz High-Performance Linux Cluster (CHiC)

3.1 Introduction to the CHiC

The Chemnitz High-Performance Linux Cluster (CHiC)¹ is a computer system, according to the definition in Section 1.1 on page 2 and it serves as a capacity system regarding the classification in Section 1.1 on page 3. It is the successor of the Chemnitz Linux Cluster (CLiC)², a 528 node, single-CPU, self-made Beowulf³ system incorporating Intel Pentium III 800 MHz CPUs and Fast Ethernet interconnection network. It is maintained by the university's computing centre and it was rated on position 126 ($R_{max} = 143.3 GFlops^4$, $R_{peak} = 424.3 GFlops$) of the November Top500 list [top00] in the year 2000.

The CHiC installed in November 2006 under the supervision of the Computer Architecture Group is a diskless 530 node system with dual-CPU (SMP architecture) and dual-core AMD Opteron 2218 (2600 MHz) central processing units, which was build by IBM and installed by Megware. Additional 8 input and output (I/O) nodes for the connection to the 60 TB storage system, 12 visualisation nodes with high-end graphics cards, two management and two login nodes are also part of the system. The interconnection network for computation is a 4xSDR (10 Gb raw bandwidth) InfiniBand⁵ (IB) network with Clos network topology [LNC98, Clo53] and an extra network for management according to the Intelligent Platform Management Interface (IPMI)⁶ standard is realised with Gigabit Ethernet technology.

The CHiC project group consists of 23 professorships and institutes at the Chemnitz University of Technology with the intention to support research in the fields of modelling and numeric simulation of problems, for example in quantum mechanics, as well as real-time rendering of complex virtual reality scenes. The following computational kernels, libraries and tools were identified to contribute to the majority of the system's load regarding a foregone survey: Basic Linear Algebra Subroutines (BLAS), double-precision General Matrix Multiply (DGEMM), ABINIT [GBC⁺02],

¹<http://www.tu-chemnitz.de/chic/>

²<http://www.tu-chemnitz.de/urz/clic/>

³<http://www.beowulf.org/overview/>

⁴GFlops = giga (10⁹) floating point operations per second

⁵<http://www.infinibandta.org/>

⁶<http://www.intel.com/design/servers/ipmi/>

Car-Parrinello Molecular Dynamics (CPMD)⁷, SIESTA⁸ and self-developed ones. The communication interface is almost only the Message Passing Interface (MPI).

3.2 Experiences from the CLiC System

The Chemnitz Linux Cluster (CLiC) is the predecessor of the Chemnitz High-Performance Linux Cluster (CHiC) as mentioned in Section 3.1. Experiences from the long-standing operation of the CLiC and therefore, resulting recommendations for the CHiC will be described as follows.

The current infrastructure of the CLiC monitoring is based on the Big Brother⁹ monitoring software – the following statements and conclusions are drawn from the use of the Big Brother Version 2003. The monitoring of specific hardware values on the computers with a Linux operating system is done by the `lm_sensors`¹⁰ software in combination with a Big Brother script that is executed by the system's *cron* daemon according to the preset interval. This is not necessary at the CHiC, because every node of the system has an Intelligent Platform Management Interface (IPMI) that takes over this functionality.

At the CLiC the monitoring of the system's fans was very error-prone. The status reports from the monitoring system often did not match the real state (refer to Section 2.4.2 on page 17). The reasons for this could not be clearly identified, but a faulty mainboard chip is the most likely explanation for this. Due to the filtering of the air within the air conditioning, no considerable dust has been deposited on the fans, this may also be the reason that only few of them actually broke down. Because of a rack-based air conditioning at the CHiC system and as a consequence of that an even more cleaner environment, a premature breakdown of fans is not expected at the CHiC. The future use of IPMI hardware monitoring, another monitoring software, and CPUs that turn off before overheating should prevent the appearance of this problem at the CHiC. The more so as a fan breakdown can be indirectly recognised by a temperature increase, which is a feature that was not implemented in the CLiC, but it is conceivable for an implementation in the CHiC.

Common practise in the monitoring of the CLiC was, that the responsible administrator looked for event e-mails every morning and from time to time during the day or the person looked at the central status website of the monitoring system to see what is the state of the system at the moment or what happened in the past. Other alerting mechanisms besides e-mail like mentioned in Section 2.5 on page 19 were not used.

Mechanisms that are able to complement the e-mail approach like SMS or automated phone calls may be viewed as useful, but only if they are set up carefully so that they do not annoy during working hours, as well as during stand-by duty. Also helpful for the configuration of the alerting as well as for the escalation procedure (refer to

⁷<http://www.cpmc.org>

⁸<http://www.uam.es/departamentos/ciencias/fismateriac/siesta/>

⁹<http://www.bb4.org/>

¹⁰<http://www.lm-sensors.org/>

Section 2.5 on page 20) is the interconnection with a groupware system [Got02, Mur00] or a similar system, which contains holidays, meetings, working hours, even the lunch break schedule if necessary, on which a decision can be made *Who?*, *When?* and *How?* a person is contacted (refer to Figure 3.1). This will help to prevent the configuration of those anyhow existing information at a second place which would originate only more work and the risk of a misconfiguration.






	Anne	Paul	Alex	...		available via e-mail
12 p.m. - 6 a.m.	stand-by duty	out of duty	holi-day	...		available via SMS
6 a.m. - 8 a.m.	working			...		available via SMS - only if urgent (escalation)
8 a.m. - 12 a.m.	working	working		...		
12 a.m. - 1 p.m.	lunchbreak			...		available via phone calls only
1 p.m. - 3 p.m.	working	meeting		...		
3 p.m. - 5 p.m.	stand-by duty	working		...		
5 p.m. - 12 p.m.		out of duty		...		not available for any communication

Figure 3.1: Exemplary Timetable as Basis for Alerting Methods

One problem with the CLiC monitoring system was the insufficient filtering of less important event messages and the tagging of really important ones. This led to the generation of up to several hundred e-mails if a central component broke down (refer to Section 2.4.2 on page 16), whereas usually only around five e-mails occurred during the day. Moreover, sometimes the administrator did not pay appropriate attention to a single possibly very important incoming e-mail, due to the large number of less important messages that too often contained only information that not necessarily require a person's attention.

Another thing that can be improved based on the experiences of the CLiC system is the amount of monitoring information that was sent. The problem is that too many messages weaken the attention of the administrator for the really important ones. It also consumes the administrator's time with things that not necessarily have to be done immediately and therefore a notification has not necessarily to be generated for this. In this situation the regular survey of the system's status for example several times a day or perhaps only now and then during the week, depending on the importance of the system that is monitored, may be enough for the administrator's check of minor important events.

Three main approaches need to be considered for this. One is that also minor important events and messages are sent via e-mail to the administrator and that in addition to that major important ones are sent also via another communication method like SMS. Second, only e-mails are used and major important messages are flagged with a custom mail header for example with the priority class "high" which can be interpreted adequately by the e-mail client. The third approach is to *push* only the really important information to the administrator and let the minor important ones be *pulled* by him or her. Which one of them, perhaps slightly modified or even another approach should be chosen is mainly an issue of the dissemination and escalation procedure (refer to Section 2.5 on page 19), as well as of the administration policy and the person's preferences.

One approach to prevent false event messages (refer to Section 2.4.2 on page 17) is the use of some kind of maintenance support functionality like the registration of the maintenance schedules. With this, the administrator would be able to see if an event is caused by something to which it has to be paid attention to or because of a planned outage for maintenance. Otherwise the administrator would not be bothered at all if no message is sent out because of the monitoring system's knowledge about the service plan. The installation of additional functionality whereby a person can advert his or her work on the elimination of a problem is not necessary at the CHiC system because the operation and maintenance is not accomplished by a huge staff but by a small group of people, therefore the installation of such a system would not justify the additional work compared with the expected benefit from this.

Disadvantages of the Big Brother monitoring system that was used at the CLiC system were, among others, the following ones. The system's abilities for historical monitoring (refer to Section 2.1 on page 7) was very limited and not very meaningful, moreover the information was not very helpful in trend observation. In addition to that the reporting of events, especially in e-mails was not very intuitive. The generated messages contained only very short and cryptic text that needed a person trained in reading them. This stands in contrast to the recommendations of an easy comprehensible and user friendly content (refer to Section 2.6 on page 21), especially the source of the information and the rule of processing that led to this message should be given – this counts all the more for a replacement administrator, who would gladly take on some help additionally provided by the system like the proposal of possible solutions regarding a specific event or if it would have executed a preceding test (refer to Section 2.4.2 on page 16) to present a perimeter of the problem. Nonetheless, the creation of rules for this is very complex and it has to be paid attention that a wrong configuration of it or an error in its execution can be misleading for the administrator.

An example for a misconfiguration of the monitoring system at the CLiC is, that for a specific computer a problem with the Network Time Protocol (NTP) service was indicated because it could not be verified if the NTP daemon runs or works correctly. But the reason for this was that the node was very busy with computations and the monitoring plugin that checks the NTP availability could not be executed in time. Thus, the problem indicated by the monitoring application was not the actual one. This could have been prevented by a more careful configuration, for example by interpreting only

more than one or two missing measured values as a problem if, apart from that, the other values are in a proper state. A disadvantage of the Big Brother monitoring system was the lack of a possibility to schedule the monitoring tasks within a specific period. It for example always started to check the connectivity (*ping*) of a computer exactly every five minutes for all computers which led to a high and short network burst instead of distributing this action over a certain period of time for a constant but low network load.

In addition to that, another disadvantage of Big Brother was the huge amount of files that had to be handled by the suite. Especially at the very beginning of the project this led to the effect that the creation of a lot of single log files caused the Linux system to run out of inodes.

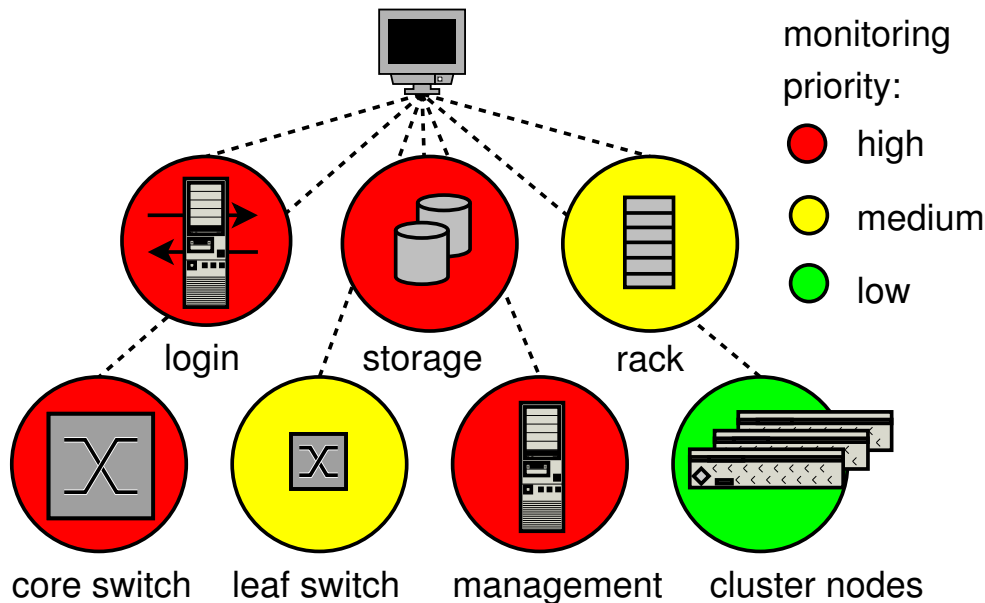


Figure 3.2: Monitoring of the Infrastructure with Corresponding Importance

In the following paragraph several considerations regarding the CHiC's monitoring approach will be shown.

First, the monitoring system has to be able to cooperate with miscellaneous components of the cluster system which means it has to be easily expandable on demand. For instance, an interaction of the batch system with the monitoring system is essential, because both benefit from the information of one another. In the situation where the batch system is updated by the monitoring system if something is wrong, the batch system can exclude a node from getting new work. Or if the batch system gets to know about irregularities over its application feedback mechanism, it can update the monitoring system so that it can process the information and induce appropriate actions. For example the Portable Batch System (PBS)¹¹ of the CLiC sometimes stopped on a node for some reason and therefore also the processing on it, but the monitoring application did not recognise this and instead indicated a proper state further on.

¹¹<http://www.openpbs.org/>

In addition to that the monitoring of the critical infrastructure of the cluster is very important, which includes the monitoring of the I/O servers, storage system, air conditioning, login servers, network switches and more. Although some infrastructure like the storage system, network switches and racks bring their own monitoring functionality, leveraging SNMP or even own mechanisms for alerting, mostly via e-mail, it is essential that those components are also connected to the overall monitoring system due to the integrated approach of management (refer to Section 1.2 on page 4) and therefore the monitoring system (refer to Section 2.1 on page 7). Because only if the breakdown of a disk in a RAID system or irregularities at a high-availability system (refer to Section 1.1 on page 2) are properly processed by an overall monitoring system, instead of the use of special purpose mechanisms and approaches, an efficient monitoring can be assured. In the end, the precise monitoring of every single compute node of a cluster is less important than the monitoring of the central infrastructure components, because a problem at this point may affect everything else on the cluster system, whereas a problem at one or a few compute nodes may only affect limited computation jobs or even nothing if the node is anyway idle at the occurrence of an error (refer to Figure 3.2 on the previous page).

Another aspect that has to be paid attention to on a monitoring system for the CHiC is the ability to present the status information in a meaningful and clear way, because on a system with so many nodes there is always something that is not in a proper state. Furthermore, there may be several nodes that are in a specific state and may remain in it for some time, for instance until the exchange hardware has arrived the node is indicated as *offline for maintenance* instead of *critical – host unreachable*. In Big Brother this led to the effect, that the overview website of the CLiC was very colourful, because Big Brother indicates the states of an object in several colours like green, yellow, red, purple, etc. but the sorting criterion was always the name of a monitored object and so the really interesting states were mixed with other ones and that was not useful to quickly find out what was wrong. A possibility to group and filter the information would simplify the work with the monitoring system in the way that it should be able to sort for example the most urgent events at the top of a table followed by the remaining events in ascending order.

Although the use of an extra network for management and monitoring reasons is a good idea, because it reduces the impact on the productive network connection, it has to be paid attention to the fact that the monitoring of the reachability of a system over the management network does not proof the reachability of it through the productive network. So additional tests have to be made, like the approach that was used at the CLiC: a node that has currently no work to do is temporarily converted into a monitoring assistant by using it for connectivity tests over the productive network.

Another possibility to improve the quality of the event messages is to introduce a separate rating system that handles the administrator's feedback regarding an event message with the goal to reduce the handling of unimportant or uninteresting messages for the administrator. Because the monitoring system cannot find this out for itself, it requires the administrator's feedback. This approach is very interesting, because it relieves the administrator from wasting time making changes in the configuration

files that may not be very intuitive, every time some little change has to be made. But the problem is, that it is not trivial to set up a system that is able to appraise the administrator's feedback to transform it into a format which is capable to be executed by the monitoring system.

The monitoring must be easily configurable, as seen at the CLiC system, a too complicated system is not suitable for the continuous change of the configuration that has to be performed at a system of this size.

Also observed at the CLiC was the effect, that a self-developed monitoring expansion module that performed several specific tests in a row could not be supervised regarding what it is doing at the moment and in which state it is in. In addition to that it checked too many objects too often, for example the CLiC system had trouble with the outage of the disks, the reason for this may have been the execution of extensive bad block test of the hard disks.

3.3 Summary

In this chapter, the Chemnitz High-Performance Linux Cluster (CHiC) and its predecessor the Chemnitz Linux Cluster (CLiC) were presented. Regarding those systems, information of the construction and of the purpose of the systems were given (refer to Section 3.1 on page 23).

Furthermore, experiences made at the CLiC system were discussed and it is considered how the operation of the CHiC could be improved based on these information. For this, advantages and disadvantages from the operation of the CLiC were analysed and recommendations were derived from this as well as from the conclusions of a generic monitoring approach (refer to Section 2.2 on page 9).

4 Evaluation of Monitoring Approaches

In this chapter the monitoring software that was chosen for the CHiC shall be considered as well as the reasons that led to this decision. Furthermore, a short description and comparison of other monitoring applications that could have been used also for the CHiC shall be given.

Based on this, a monitoring script is presented that adds new functionality, to the monitoring application. Further on, it was paid attention to the fact that this new piece of software fits into the desired monitoring strategy and infrastructure.

4.1 Selection of a Monitoring Application

In the area of monitoring software a lot of different applications are established. They all have different features, depending on what kind of monitoring they are focused on (refer to Section 1.2 on page 4 and Section 2.1 on page 7). There are products, that focus on the network which means that they can easily monitor the network infrastructure, e.g., switches, routers, etc., and they usually have a good SNMP support. Another kind of software is explicitly made for performing a specific task, for example monitoring a computer cluster or very specific hardware or software components.

There are also products that have a general focus on a possibly wide availability for example in heterogeneous environments for Unix, Linux, Windows, Mac OS, etc., as well as a support for very different platforms such as standard computers with IA-32/IA-64 or PowerPC (PPC), proprietary hardware or software (storage systems, access control systems, air conditioning, embedded devices, micro-controllers and so on). Further on, they all have different strengths and weaknesses in several areas, for example in maintenance, alerting mechanisms, use for *mission critical* infrastructure, etc. And finally a differentiation regarding proprietary software, freeware and open source software could be made.

During the course of this chapter, proprietary software products such as the monitoring components of large suites like HP OpenView¹ (NNM²) and IBM Tivoli³ (NetView⁴) are not considered further. The advantages of this kind of software which are

¹OpenView Monitoring: <http://openview.hp.com/solutions/nsm/>

²Network Node Manager: <http://openview.hp.com/products/nnm/>

³Tivoli Monitoring: <http://www.ibm.com/software/tivoli/products/monitor/>

⁴NetView: <http://www.ibm.com/software/tivoli/products/netview/>

primarily a rich function set and a seamless integration in the existing workflow or infrastructure of an institution do not compensate the disadvantages of them regarding the use for the CHiC. These disadvantages are first of all a high complexity of the whole system that results in expensive customising and a comparatively deep know-how that is necessary for operation and the development of special extensions, as well as the high purchase price and partially obligatory vendor support.

The further considerations shall be restricted on free software that is widely used for monitoring tasks, and in addition to that software that is used at the CLiC and the university's computing centre. The following section gives a short overview about software that could be suitable for the CHiC, regarding an as far as possible generic approach that allows the easy integration of all the monitoring objects into one solution without having multiple applications for similar tasks side by side.

If not mentioned otherwise, only a short introduction and significant advantages or disadvantages are given for every software to clarify the reasons that led to the decision for one or another monitoring software for the CHiC. The selection of the software can be made regardless of existing applications at the university's computing centre and CLiC, which are Big Brother 2003, Nagios 2.6 and Network Node Manager of HP OpenView 6.2, because the CHiC is a self-contained system which is going to be arranged preferably independent from other systems.

Big Brother The monitoring software Big Brother⁵ was used for the CLiC system and it is still in use for monitoring the infrastructure of the university's computing centre. The software is not state of the art any longer. Some design aspects, like the way of using shell scripts means a lot of maintenance work [Sel00]. For more aspects regarding the use of Big Brother refer to Section 3.2 on page 24.

Another aspect that keeps this software from being deployed at the CHiC is the ambiguous licence situation of it for non-commercial users and that there is no ongoing development. Still, this software was analysed because it would have been a good idea to use software that is already used in the institution but because of its disadvantages and that there is no need to use the same software as the computing centre as described above, the application was discarded at an early stage of the evaluation of monitoring software.

Big Sister A very close alternative would have been Big Sister⁶. It is still being developed, open source and compatible to Big Brother [CWSC01], but it also means that the weaknesses of the design, for example the inability to efficiently parallelise the checks, are included too. In addition to that, the existence of only a few plugins and the small dissemination of it in combination with a comparatively small community, led to the decision to discard Big Sister for further analysis at this point.

⁵<http://www.bb4.org/>

⁶<http://www.bigsister.ch/>

OpenNMS The open source network management system OpenNMS⁷ is an application that has its main focus on the management of networks, which can be seen at the excellent SNMP support. The first impression of this program is quite good, but there are some disadvantages [O'D02, Bal05]. It is the most complex software of the ones presented here, the design is based on Java and XML which is among other reasons causal for the difficulties to get the system running. Remote and passive monitoring capabilities are relatively new features, and in addition to that, the possibility for an easy use of plugins for one's specific needs is based on the Nagios plugin infrastructure.

To sum up, OpenNMS is a very powerful program for managing and monitoring a network of components, if supporting SNMP. The use of additional software that can monitor specific values is only supported by an indirection via Nagios plugins. After all, OpenNMS will not be analysed further regarding a possible use at the CHiC.

Ganglia An application that has been explicitly developed for the use in cluster computers is Ganglia⁸. The software architecture consists of a daemon which is installed on each computer that shall be monitored. The daemon gains the desired information and sends status reports to another daemon on a server which collects and stores the information gained by the client daemons.

The application has some very good features, for example a very lightweight data exchange protocol between the daemons [SKMC03] and an easy installation and configuration. But there are some disadvantages, for instance the documentation is very short. A resulting problem from this is, that neither there nor in other documentations⁹ for Ganglia an option for a notification of the administrator in case of a problem can be found – alas this is an important component for an integrated monitoring system with real-time monitoring (refer to Section 2.1 on page 7). Another disadvantage is that there is no possibility for modelling the infrastructure in-depth and therefore also no option for modelling dependencies between the components. There is an automatic discovery feature that is quite functional, but it has one vital handicap – it does not recognise computers that are already down right from the beginning because it has no information about the infrastructure in advance as mentioned above.

Another aspect which is a two-edged sword is the procedure that every monitoring client briefs its current state via multicast in the network at default settings. On the one hand, this has the advantage that the status of the whole network can be seen at every node and not just on the monitoring server. On the other hand, this behaviour only scale up to a certain limit of cluster nodes – as the developers of Ganglia confess themselves (refer to [MCC04]). Furthermore, for a cluster computer environment this feature is definitely not necessary.

⁷<http://www.opennms.org/>

⁸<http://ganglia.sourceforge.net/>

⁹<http://www.ibm.com/collaboration/wiki/display/WikiPtype/ganglia>

Although the application was among the last choices, not only because of its wide use in other clusters all over the world, it was not chosen for the CHiC because of the disadvantages mentioned above.

Nagios The software Nagios¹⁰ is basically just a framework which have to be extended by various kinds of features, plugins and extensions, to set up a solution that fits one's needs best. It is the last one that should be mentioned here and to anticipate the decision, it is the one that was chosen for the use at the CHiC. It is an application that also has some negative aspects, for example a somewhat higher network load compared to other software and the design issue that it starts a single process for every task that it has to accomplish, leading to a higher load on the monitoring server. Nevertheless, it is the one that showed the most solid performance in various important disciplines.

First of all, the Nagios open source software is only a monitoring framework which is delivered with various extensions that makes it a very generic application that can monitor notionally everything. For example the Nagios Remote Plugins Executor (NRPE) add-on allows, the execution of software on a remote machine and beyond network borders. In addition to that the Nagios Service Check Acceptor (NSCA) add-on allows passive checks that work in a trap-based style. On top of that the flexibility that comes with the plugin concept makes Nagios capable of monitoring network devices with and without SNMP, as well as standard computer clients and special hardware and software. For standard requirements, plugins are included, for everything else, the large and active community provides a lot of additional plugins that fulfil most needs. In the end, because of the open plugin application programming interface (API) and the good and comprehensive documentation [Gal06] it is possible to add necessary functionality.

Furthermore, compared to other software the notification and alerting functionality is very good as well as the graphical report and visualisation functions which are feasible for a basic recognition of trends and the historical performance of an entity – it can be even improved by additional tools such as Multi Router Traffic Grapher (MRTG).

Facts about Nagios that are important for this work are given above, a further extensive introduction of the detailed functioning, features, installation, etc. shall be refrained from at this point – for this refer to [Gal06], [Fri02] and [Har03]. Moreover, a general, only matter-of-fact comparison of Ganglia, Nagios and various other applications that does not contain a rating of the software, but which compares only specific aspects and features, can be found in [GWB⁺04].

The monitoring applications shown above are only a selection from plenty of software that is similar to those, but which was not examined in-depth, for various reasons. For example they are not developed further, the software is a one-man project only,

¹⁰<http://www.nagios.org/>

the latest release is too old, there is no community for it, important functions are missing, the information on the website appears outdated, respectively a documentation is missing. In alphabetic order these are: Angel Network Monitoring¹¹, CluMon¹², GroundWork¹³, Lemon¹⁴, Midas¹⁵, Mon¹⁶, Munin¹⁷, Performance Co-Pilot¹⁸, PIKT¹⁹, Spong²⁰, Supermon²¹ and Zenoss²².

At the end and as glanced at quickly above, Nagios was chosen for the CHiC based on the selection of various monitoring applications and the considerations that were made regarding them. Due to its powerful framework design the implementation of various functions that extend the basic monitoring abilities can be carried out easily. Which kinds of specialised plugins and functions could be realised, will be discussed next.

One feature of a monitoring application that has to be paid attention to, are the abilities for *historical monitoring* (refer to Section 2.1 on page 8). Also known as trend or performance monitoring this function could be necessary to watch the *trend* of the system's capacity utilisation – usually this is done to detect bottlenecks in advance and to gain information for the expansion of a system. For the CHiC, no upgrades are planned and a capacity overload will be prevented with the help of a batch system. Thus, regarding this topic, an extensive monitoring of the system to gain those data is not mandatory. For example, information regarding the system's usage can also be gained via the load of the batch system or indirectly via the analysis of anyway available *real-time monitoring data* (refer to Section 2.1 on page 7) – an active and because of its nature performance reducing monitoring, only because of this, would be a waste of resources.

Another reason for historical monitoring, the accounting of the system's usage, is not the main focus at the CHiC, because at the first expansion stage only research inside the university shall be performed. If the CHiC should some day be included into a grid infrastructure, monitoring because of accounting reasons does not have to be done at the stage of the system's monitoring software, because for example the Globus Toolkit since version 4.0 does contain the SweGrid Accounting System (SGAS) which is responsible for this.

Furthermore, at a late stage of expansion the system could be extended to support the users in analysing the possibly good performance of their applications. For the support and also for the training of the users, information, regarding the enhancement of their

¹¹<http://www.paganini.net/index.cgi/angel/>

¹²<http://clumon.ncsa.uiuc.edu/>

¹³<http://groundworkopensource.com/>

¹⁴<http://lemon.web.cern.ch/>

¹⁵<http://midas-nms.sourceforge.net/>

¹⁶<http://www.kernel.org/software/mon/>

¹⁷<http://munin.projects.linpro.no/>

¹⁸<http://oss.sgi.com/projects/pcp/>

¹⁹<http://pikt.org/>

²⁰<http://spong.sourceforge.net/>

²¹<http://supermon.sourceforge.net/>

²²<http://www.zenoss.com/>

programs that run on the cluster, for example data regarding the cache misses and the time that was spent with waiting, could be gained by additional monitoring modules. Although, this is a very useful feature, it is not the primary concern of the monitoring of the cluster. If this option is required, specialised tools (e.g. Paradyn²³) that are usually started via the batch system can be used for this, but these are not the main focus of a generic monitoring system.

Nevertheless, additional monitoring time and effort can be spent during times when a cluster node is idle. In combination with the batch system that knows about these situations, some tests could be performed. For example at the CLiC system, hard drive checks were performed, but they are not necessary at the CHiC, because the compute nodes are diskless ones. A useful check on an idle computer could be the test of the system's memory as well as for example the check of the behaviour of specific components. This could be for instance the throughput of the network connections, primarily the InfiniBand (refer to Section 4.3 on page 38) but also the Gigabit Ethernet management connection is conceivable. If there are a minimum of two idle nodes, fitness tests of involved network hardware could be made which means that the speed and the error rate could be measured. First of all, physical damages from the installation and those that could arise from the maintenance can be discovered, as well as the slow deterioration of performance caused for example by the ageing of the components which could be discovered by the analysis of the monitoring data over time.

Finally, a main goal of the efforts regarding monitoring is the creation of a powerful system with which the users can solve their problems as good as possible. This can be done by making and keeping the system as fast as possible, which does work only if everything is in good condition – this is the issue of the monitoring system. The worst performance, a single component can have, is if it does not work at all, so the downtime of it must be as short as possible as well as the downtime of the whole cluster, if a major incident has happened. Although, this is not very comfortable, the consequences are not as significant as on a high-availability system, for example in a commercial environment. Nevertheless, the downtime of for example the whole cluster should be minimised, because the depreciation of it is distinctive and can be calculated as follows.

The pure purchase price of the cluster was 2,640,000 *Euro*. If an operation period of 5 *years* is assumed,

$$\frac{2,640,000 \text{ Euro}}{5 \text{ years} \cdot 365 \text{ days} \cdot 24 \text{ hours}} \approx 60 \text{ Euro/hour}$$

in terms of figures one hour mathematically costs about 60 *Euro*. In addition to that, also singular costs for the reconstruction of the room ($\approx 1,800,000 \text{ Euro}$), running expenses for the maintenance staff (one administrator for one year: $\approx 45,000 \text{ Euro}$), electricity²⁴ (maximum power consumption: $\approx 200 \text{ kW}$) and air conditioning (additional 200 *kW*) have to be considered,

²³<http://www.paradyn.org/>

²⁴energy price: roughly estimated 0.10 Euro/kWh

$$\frac{1,800,000 \text{ Euro}}{5y \cdot 365d \cdot 24h} + \frac{45,000 \text{ Euro}}{365d \cdot 24h} + 2 \cdot 200 \text{ kW} \cdot 0.10 \text{ Euro/kWh}$$

$$\approx 41.10 \text{ Euro/hour} + 5.14 \text{ Euro/hour} + 40.00 \text{ Euro/hour} \approx 86 \text{ Euro/hour}$$

so that the additional costs are about 86 *Euro/hour*.

Thus, the total expenses are 60 *Euro/hour* + 86 *Euro/hour* = 146 *Euro/hour*. That means in total, every day, the cluster cannot be used for scientific calculations, costs in terms of figures 146 *Euro/h* · 24 *h* = 3504 *Euro/day*²⁵, so that the outage of the whole cluster or just a few nodes should be as short as possible with the help of the appropriate monitoring strategy.

4.2 Nagios and the Plugin Topology

As the monitoring platform for the cluster, Nagios was chosen (refer to Section 4.1 on page 34), hence a short introduction into the topology of Nagios 2.6 shall be given for an easier understanding of the decisions regarding the developed software.

The simplest way of monitoring is, if the monitoring server actively checks the monitoring client directly via a network connection – for this, the server sends a request message to the client which sends an answer and finally the server processes the answer. Hereupon the server performs an adequate reaction to it which can be also caused by a missing answer, depending on the configuration of the monitoring application (refer to Section 2.5 on page 19). With this approach of monitoring, for example, it can be checked if the client is up (*check_ping*) or if a certain service is working (*check_ssh*, *check_mail*, etc.). The advantage is that nothing at the configuration of the client has to be changed and if the administrator is interested only in monitoring values that can be gained via the network, the setup of such an approach can be performed very easily.

If the requirements regarding what has to be known about a client are more demanding, for example the CPU temperature or the system's load, the approach from above has to be expanded. For this, on the specific client that has to be monitored, an additional service has to be set up which can gain the necessary information. It works in the way that on the monitored computer the Nagios Remote Plugins Executor (NRPE) is installed, which is responsible for the communication with the monitoring server similar to the description above as well as for gaining the information by executing plugins (*check_temperature*, *check_load*). Hence, the NRPE add-on, that was tested and utilised in version 2.5.2, works as an agent in between the request of the monitoring server and the location from where the monitoring information is required. Because it is possible for NRPE to execute any kind of plugin, even another NRPE instance, this software can be used not only as a local, but also as a remote gateway which can be even multi-level. It means, in a situation where a device that should be monitored could not be accessed directly by the monitoring server, for example because the client is in the scope of another network, NRPE could be used as gateway if it is installed on a computer that is accessible from both networks, so that the monitoring server can check the

²⁵3504 Euro are around \$4555 (1 Euro ≈ 1.30 USD, Jan. 2007)

clients health status by performing a *check_ping* command via NRPE. The monitoring server would send a request for executing a check, if a specific computer is alive, to the computer, on which the NRPE service is running, that executes the *check_ping* command by proxy for the monitoring server and sends the result back to it.

The third main possibility of the server for gaining monitoring information is to “perform” passive checks. It means, the server is configured in the way that it is waiting for a status report from a specific device or for a specific value instead of initiating an active check itself. If the information of a passive check, which is also called a trap, reaches the server it is processed like a response to an active check. Depending on the server’s configuration, the absence of a message within a specific period of time either means for the server that everything is alright or that it is not. For Nagios, the add-on NSCA (Nagios Service Check Acceptor) exists which runs as a daemon on the monitoring server. It receives the incoming passive values and passes them on to the core of the Nagios application which further processes them. In order that no one can send possibly faked information to the server, the connection between the NSCA daemon and the NSCA client can be weakly (XOR) or strongly encrypted (e.g. DES, TDES, AES, Twofish, Serpent) – because of the additional load that every encryption generates it should be carefully considered which level of security is necessary in a specific situation. Sending monitoring status information to the server in the Nagios context means to pass the information in a specified format to the `send_nsca` client on the computer that is monitored which delivers them to the NSCA daemon on the monitoring server. During this work, version 2.6 of NSCA was tested and utilised.

Last but not least a few words regarding the configuration of Nagios – some examples can be seen in Appendix B on page 73. The text-based files are self-explaining as far as possible, the configuration of the hosts and services is straightforward, but some attention has to be paid to the definition of the check commands, depending on the intention to use a check command directly or via NRPE a different argument passing has to be considered (refer to Section B.2 and Section B.3 on page 76). Furthermore, for the passive monitoring with NSCA it is crucial that the value names that are passed on to Nagios are the same as the service names for what Nagios is expecting information, otherwise Nagios will reject them without notice. Because of the missing response regarding the nature of the design it is hard to detect misbehaviours concerning this matter.

4.3 The InfiniBand Interconnection Network

Due to the usage of InfiniBand at the CHiC system and that an enhancement of its monitoring approach is presented in Section 4.4.1 on page 40 which is based on this technology, some basic information about InfiniBand are given below.

4.3.1 Introduction to Design and Features

As the main interconnection network the InfiniBand Architecture (IBA) is used. Although, it was developed as a point-to-point bidirectional high-speed interconnect for inter-server and server-I/O communication in standard computing centres, in the majority of cases it is nowadays used in High-Performance Computing (HPC) systems.

The design goal of the InfiniBand Trade Association²⁶ (IBTA) was a system that offers a possibly high “reliability, availability, performance, and scalability necessary for present and future server systems” [Pfi01]. The result was a standard [Inf04] that supports symmetric full-duplex connection from 2.5 Gb/s gross (single data rate). Very common in InfiniBand installations is a $4 \times 2.5 \text{ Gb/s} = 10 \text{ Gb/s}$ raw bandwidth connection which results in a usable data rate of a maximum of 8 Gb/s.

Important features of InfiniBand are Remote Direct Memory Access (RDMA), message send/receive within the user space and the support of standard bus systems like PCI-X or PCIe [Hoe05, CWP03]. The main reason for the use at the CHiC was the comparably low price regarding the provided bandwidth and latency.

4.3.2 Constitution of the Port Counters

The InfiniBand network interface, that is called Host Channel Adapter (HCA), offers a set of values that can be monitored. In general they are called *port counters* but depending on the characteristic that should be accentuated it can also be referred to them as *performance counters*, *health counters* or *error counters*.

A selection of the most interesting port counters that are also important for the specific monitoring of the HCA regarding the InfiniBand standard [Inf04] are presented in Table 4.1 on the following page. They are collected separately for every single port, of which a Host Channel Adapter can have multiple ones.

In addition to the counters in Table 4.1 on the next page, the InfiniBand HCA also offers the values PortXmitData (transmitted data in byte), PortRcvData (received data in byte), PortXmitPkts (transmitted packets) and PortRcvPkts (received packets), but because of the limited benefit for monitoring the system’s health, they are not considered further.

The knowledge of the significance of each counter is based on the experiences with the network, therefore no generally valid statements of the importance of each value can be made. So, the detailed setting of threshold values and the appropriate reaction of the monitoring system on their exceeding must be customised during the operation of the system. Hence, no preferred differentiations will be made at this point regarding the attention to each counter.

²⁶<http://www.infinibandta.org/>

counters	description (number of ~)
SymbolError	minor link errors detected on physical lanes
LinkErrorRecovery	successfully completed link error recoveries
LinkDowned	failed link error recoveries (link down)
RcvErrors	received packets containing an error
RcvRemotePhysErrors	packets that are received with a bad packet end delimiter
RcvSwRelayErrors	received packets that were discarded because they could not be forwarded by the switch
XmtDiscards	outbound packets discarded because the port is down or congested
XmtConstraintErrors	packets not transmitted from the switch's physical port
RcvConstraintErrors	packets received on the switch port that are discarded
LinkIntegrityErrors	times that the count of local physical errors exceeded the specified threshold
ExcessiveBufferOver-runErrors	consecutive (receive) buffer overrun errors
VL15Dropped	incoming management packets dropped due to resource limitations

Table 4.1: InfiniBand HCA port counters

4.4 Design and Implementation of a Port Counter Monitoring Plugin

In the following section, the design and implementation of a possibility to monitor the performance counters of the InfiniBand interface shall be considered, for this the self-developed monitoring script `check_iberr` will be described (refer to Appendix A on page 67). It is capable of checking the health status of an InfiniBand network adapter.

4.4.1 Preliminary Considerations

At the CHiC system the monitoring of the port counters (refer to Table 4.1) can be made with the subnet manager (SM) of the Voltaire InfiniBand core switches that offers, among other information and abilities, an interface so that the administrator can supervise them. However, an integration into a monitoring system like Nagios is not provided. Thus, the monitoring of the port counters with this alternative would have meant to write a tool that filters and converts the data of the Voltaire specific status file provided by the core switches to be able to process them in the monitoring application. The disadvantage of this approach is that a solution that is specific to the Voltaire switches can be used only with these. Hence, the decision was made to develop a vendor independent possibility to monitor the performance counters so that the information can be processed by the monitoring system Nagios.

Some vital advantages that were decisive for this are the potential to use this approach

also in environments with mixed vendors and in situations where multiple isolated InfiniBand network scopes have to be monitored. Furthermore, the use of the locally executed version of the plugin via NRPE (see below) allows to monitor the port counters of the InfiniBand network interface and, in spite of a malfunction of the InfiniBand network, troubleshooting can be made with the help of another network connection. At the CHiC system this is a Gigabit Ethernet connection that can be used in this case to perform analyses of the InfiniBand state despite its outage.

Furthermore, the reason for developing the `check_iberr` script was the interest in the health status of the network connections. The knowledge about the detailed state of a network connection can be an important information in several fields. For example it can be useful for error diagnostics or the verification if everything is alright with the network, so that as reason for a possible misbehaviour of some software the network connection can be excluded. This knowledge is also important for finding the reasons for performance issues, as well as for isolating the possible reason for a problem with the network. Among others also for the identification of possibly abnormal behaviour of the network regarding specific issues that can be seen only in the temporal trend, e.g., abnormal behaviour of a component that was caused by occasional errors at the network interface.

The `check_iberr` plugin was written in the programming language Perl, because if necessary Nagios can be used with an integrated Perl interpreter that has a better performance than the standard Perl interpreter that otherwise would have to be executed every time for the accomplishment of each script. Furthermore, Perl is the preferred script language for plugins by the Nagios community²⁷, shown by the fact that a lot of them are written in Perl and that tutorials and libraries exist for this programming language. Plugins can also be written in other script languages, as far as their execution is possible on the Nagios server or on the device that is monitored if NRPE is used.

Nevertheless, binary programs can also be used as a plugin for Nagios. Actually, the difference that makes an ordinary script or binary program a Nagios plugin is just the way how it expresses its return codes and values. Basically, Nagios expects return codes that stand for the status types (0 = “OK”, 1 = “WARNING”, 2 = “CRITICAL”, 3 = “UNKNOWN”) and optional some textual information that describe the status codes closer, as well as additional information that are stored for generating detailed reports, e.g., for historical monitoring (refer to 2.1 on page 8). If some program provides the desired monitoring information but it does return them in another format, a Nagios plugin would consist only of a wrapper that converts them as needed.

The name of the script `check_iberr` comes from the different names that express the characteristics of the InfiniBand port counters (refer to Section 4.3.2 on page 39). The names of the plugins in Nagios regardless if they are scripts or binaries are usually composed of `check` + *underscore* + NAME, whereas the NAME typically describes what exactly the plugin is doing. In this case, the name `check_iberr` describes that a Nagios plugin that checks the InfiniBand (IB) error counters, the ending `.pl` to mark it as a Perl script can be optionally added.

²⁷<http://www.nagiosexchange.org/>

4.4.2 The `check_iberr` Script

The realisation of the port status monitoring `check_iberr` plugin for Nagios is based on the OpenFabrics Enterprise Distribution (OFED) in version 1.1 of the OpenFabrics Alliance²⁸, therefore this software collection has to be available on the systems that shall be monitored. The plugin has to be executed with additional rights²⁹ due to the dependency on information that can be gained only with extended rights. Apart from the installation of the NSCA client on the computer that runs the plugin, no other pre-conditions are necessary³⁰.

First, only the essential parameters shall be considered. The execution of `./check_iberr.pl -H name` is the most basic variant. Whereas, the parameter `-H` is the only one that is mandatory, which expects the *name* of the computer that is monitored. It has to be the exact name of the computer as it was defined in the Nagios configuration, so that the monitoring server can correctly associate the results to it (refer to Section 4.2 on page 38). All other parameters are optional, respectively they have default values, so that they not have to be specified until they are needed. Anyhow, the parameter `-G` for the global unique identifier (GUID) of the port or `-l` for the local identifier (LID) of the monitored InfiniBand interface is usually specified, as well as the parameter `-m` for the name of the Nagios monitoring server. The GUID of an InfiniBand network port is comparable with the MAC address of an Ethernet network port, whereas the InfiniBand LID, which is a kind of a dynamic identifier that is only valid inside a specific network scope, does not have a straight counterpart at an Ethernet network, but the LID is soonest comparable with a private IP address. More script parameters like for the output of debug messages, the specification of paths for the OFED or NSCA directory, etc. can be discovered with the `-h` help option (also refer to Appendix A on page 67).

The application works as follows. It expects the GUID or LID of the InfiniBand port that should be monitored, otherwise *localhost* is assumed. If a GUID or the option *localhost* is given, the corresponding LID is looked up with the *ibaddr* program from the OFED software collection. The next step is to perform the check of the port counters (refer to Table 4.1 on page 40) with the help of the LID. For this, the *ibcheckerrs* script that is also part of the OFED software collection is used. After this, its response is processed and the results are sent out to the monitoring server with the *send_nsca* program. The standard option is to send out only the values that exceed the corresponding thresholds, but optionally also the status of all port counters can be sent out (`-u` update option). If desired, the counters of the InfiniBand port are reseted (`-r` reset option) afterwards. The last activity of the plugin is to choose its appropriate return value, which is either “OK” (non of the port counters exceed a threshold value), “WARNING” (one or

²⁸<http://www.openfabrics.org/>

²⁹The execution of an application with root permissions is a known security concern, that is why the limitation of the rights is advisable. This can be done among other possibilities via *sudo*, *setuid* or *SELinux*.

³⁰For detailed instructions of the installation of the script and the necessary configurations on the monitoring server, as well as on the computer that executes the script, refer to the documentation that is included in the plugin.

more values exceeded a threshold), “CRITICAL” (the exceeding of minimum one value is higher than defined with the `-c factor` option) or “UNKNOWN” (something unexpected has happened or an error occurred during execution, e.g., a wrong LID or GUID was given). In addition to that, applicable extra information regarding the return values are printed out for further processing in Nagios.

For a detailed insight how the `check_iberr` plugin works, refer to the source code in Appendix A on page 67.

As it can be seen from the explanation above, there are two ways in which the `check_iberr` plugin can be used for monitoring the port counters of an InfiniBand HCA. The first is that it reads only the port counters of the host on which it is executed. This is the alternative for the use of the plugin via the Nagios Remote Plugins Executor (NRPE) as described in Section 4.2 on page 37, it is recommended if the port counters shall be checked without using the InfiniBand network. The second alternative is, starting from one computer, to check the port counters of different hosts via the InfiniBand network. If it has to be assured that the check of the counters is performed locally only, the plugin has to be executed neither with the GUID (`-G`) nor with the LID (`-l`) option so that the default value *localhost* is assumed. Otherwise, using either the GUID or LID port identifier, it is automatically detected if the counters for the given identifier can be checked locally, because it belongs to the local InfiniBand interface from which the information can be obtained. If the detection shows that the port counters have to be obtained remotely via the InfiniBand network connection, they are requested from the remote host corresponding to the given identifier (GUID or LID). The differences of the two alternatives of checking the port counter information locally or remotely, regarding the impact on the network performance, as well as on the host’s computing performance are discussed in Section 5.2 on page 47 and Section 5.3 on page 54.

Compared to the standard approach of Nagios for gaining monitoring information, the use of the Nagios Service Check Acceptor (NSCA) add-on has several advantages.

The general procedure of checking a value with Nagios plugins is, that the monitoring server requests the check of a specific value regarding its internal schedule and that a plugin is executed thereupon. This plugin checks, regardless if it is executed locally or remotely, usually one value and returns the result as mentioned above to the monitoring server. If there are multiple values to check, one plugin each has to be executed. This approach is advisable if all values are different, because different plugins have to be used for this anyway. Unlike the port counters (refer to Table 4.1 on page 40) of the InfiniBand network interface which can be checked all in the same way.

The execution of one plugin for every value that has to be checked would mean a significant expense. Therefore, the monitoring of the InfiniBand port status shall be performed in a more efficient way. Thus, the approach was chosen to use NSCA (refer to Section 4.2 on page 38) to reduce the plugin’s communication and execution effort for gaining the monitoring information. The advantage of it is, that only the values that exceed a predefined threshold are reported to the monitoring server. However, if a specific value is alright for a long time, no reports are sent to the monitoring server, which hereupon cannot be sure if the absence of reports is due to an alright status of a value or because a problem has occurred in the monitoring workflow. To face this

issue, the `check_iberr` script has the update option (`-u`) that reports the status of all values of a specific InfiniBand port to the monitoring system.

Finally, the `check_iberr` plugin that was presented above is able to monitor the InfiniBand port counters in different efficient ways. However, there are a few minor things that could be done to improve it further. One thing is to refine the usability of the script. For example, to use the `-G` and `-l` option for passing a GUID and a LID at the same time does not make sense. Either both identifiers refer to the same port, than they do not both have to be passed on, or they refer to different ports and the script has to opt for one, which would not be conformable with a consistent program behaviour and shall be avoided. Hence, a possible usability improvement is if the script would offer only one option for passing an identifier and be able to detect whether it is a GUID or LID.

Another thing that could be improved in the `check_iberr` plugin is its robustness of the program flow against unexpected behaviour of external, depending programs and functions. But all these are minor topics which do not effect the general functionality of the script if installed and used as documented. However, those improvements are not crucial, and have to be put off until a later development version due to time restrictions for this work.

4.5 Summary

In this chapter, considerations regarding a monitoring approach for the CHiC were made.

First, starting from a plenty of different monitoring software that exists, the number of applications to be considered further was limited to a subset of them that could be used for the CHiC. According to this, several products were excluded because they did not fit minimum requirements, so that the software Big Brother, Big Sister, OpenNMS, Ganglia and Nagios were chosen for a more detailed comparison (refer to Section 4.1 on page 32). Based on this, the Nagios monitoring framework was chosen as the one that fits best for the monitoring of the CHiC system.

Furthermore, several aspects and requirements were considered that are relevant for the monitoring of the cluster, for example the need for minimising the outage of the system. Calculations regarding this showed that every day which the cluster cannot be used as planned the amount of around 3500 Euro gets lost in terms of figures. Hence, this showed that it is important to minimise the time that the CHiC is out of service and that it is crucial to monitor the system to face this issue.

With reference to the monitoring software Nagios that was suggested for the CHiC, an introduction was given to the application itself and the corresponding plugin topology which is important for the monitoring topics (refer to Section 4.2 on page 37). In addition to that, a short introduction to the InfiniBand interconnection network was given, in particular to the port counters of the InfiniBand HCA (refer to Section 4.3 on page 38).

Finally, the design and implementation of the `check_iberr` monitoring plugin

that checks the health counters of an InfiniBand network interface was presented (refer to Section 4.4.2 on page 42). Thereby, the focus was on a solution that is as efficient as possible in checking the values so that the influence on the computation tasks of a computer is preferably low.

Moreover, the `check_iberr` plugin was posted on the Nagios plugin developer mailing list (*nagiosplug-devel*) whereupon it was challenged releasing the plugin on the official central repository³¹ for plugins, which is now the place where possible further improvements will be released.

³¹<http://www.nagiosexchange.org/>

5 Evaluation of the Performance Impact of Monitoring Activities

5.1 Introduction to the Test Configuration

In this chapter the Nagios plugin `check_iberr` for InfiniBand port status checks (refer to Table 4.1 on page 40), developed during the course of this work, and a set of various standard Nagios plugins (refer to Table 5.1 on page 51) will be tested regarding their impact on the network performance as well as the performance of the applications that run on a specific compute node.

The test configuration consists of four dual-CPU, dual-core, 64 bit Intel Xeon 5130 at 2 GHz computer systems with a total of 2 GB RAM under Scientific Linux 4.4 with kernel 2.6.9-42.0.3.ELsmp. The interconnection consists of two separate Gigabit Ethernet (GbE) ports and one 4xSDR (10 Gb) InfiniBand port.

If not described otherwise the computers *c6-3* and *c6-4* have been used for the performance measurements. The computers *c6-1* and *c6-2* have been used for triggered the interrupt data (*c6-1* to *c6-3* and *c6-2* to *c6-4*).

5.2 Impact regarding the Execution of Applications

This section shows how monitoring of a computer can influence the applications that run on it. Referring to this, the application ABINIT¹ in version 5.2.3 was used, which is a software package developed to find charge density, electronic structure and the total energy of systems made of nuclei and electrons within Density Functional Theory (DFT) [GBC⁺02]. It was chosen as a reference application because it is one of the software packages which are supposed to be used at the CHiC (refer to Section 3.1 on page 23).

The ABINIT software, in particular the parallel version *abinip*, was used with the help of the MVAPICH2² MPI implementation in version 0.9.8 and OpenIB³ InfiniBand in version 1.1 as the network interconnect. As the input data for ABINIT, a working set was used that was originally created for benchmarking the submissions for the solicitation of the CHiC and which results in eight jobs, because they fit well on two computers

¹<http://www.abinit.org/>

²<http://nowlab.cse.ohio-state.edu/projects/mpi-iba/>

³<http://www.openfabrics.org/>

with two CPUs and two cores each. Another issue for choosing this benchmark is the execution time of approximately 90 seconds which is long enough to compensate minor irregularities during the measurements and short enough for the test series to be manageable regarding the large number of them that have to be executed for gaining the measured values.

Moreover, ABINIT uses 100 percent of the CPU resources on all cores, so it is guaranteed that the system is always on heavy load and that also minimum influences on the system can be measured indirectly via the execution time of ABINIT.

5.2.1 Influence on Abinit due to Local and Remote `check_iberr` Script Execution

The influence on ABINIT caused by the execution of the `check_iberr` script as a local script via the Nagios NRPE add-on, as well as the execution of the same script on a remote computer is presented in Figure 5.1 on the facing page. Its intention was to clarify the significance of the slowdown of a typical application that runs on the cluster regarding a specific monitoring value and its monitoring interval. In addition to that the difference between two methods of gaining the same information was determined. The reason for this was to find out whether it is better to gain the performance counters for the InfiniBand network connection on a specific host either directly via the same InfiniBand network connection that is also used by ABINIT or via an additional out-of-band (OOB) communication over the GbE management network.

Referring to this, Figure 5.1 on the next page shows the dependency between the execution time of ABINIT (in seconds) on the y-axis and the delay between the execution of two `check_iberr` scripts in a row (in seconds) on the x-axis. The delay between two script executions is displayed on the abscissa because during the measurements the delay was the best way to guarantee a constant triggering rate on the two systems to be measured during the variable execution time of ABINIT. The labelling of the second y-axis on the right side of the diagram shows the issue mentioned above as the slowdown of the ABINIT execution (in seconds) regarding its execution time without any disturbances. An uninterrupted run of ABINIT takes about 90.9 seconds – this value was used as the basis for the calculation of the slowdown of the application. Furthermore, every measuring point consists of the average of at least six independent measurements, which means a minimum of six full ABINIT runs at any given script execution delay value. The delay of $1e^{-6} s = 1\mu s = 0.000001s$ is roughly the same as if there is no delay, which means that the script executing process cannot be made faster than this.

Thus, the analysis of Figure 5.1 on the facing page shows a slowdown of the ABINIT execution if the `check_iberr` script is executed locally via NRPE over GbE of about 40% but only for very small delays which is synonymous for a high monitoring frequency. If the delay between two script executions becomes larger, the slowdown of the ABINIT execution becomes smaller. At a delay of one second the slowdown is less than 4% and at a delay of ten seconds less than 0.5%. It means that the influence of the script execution on ABINIT is already that low at this untypically frequent monitoring

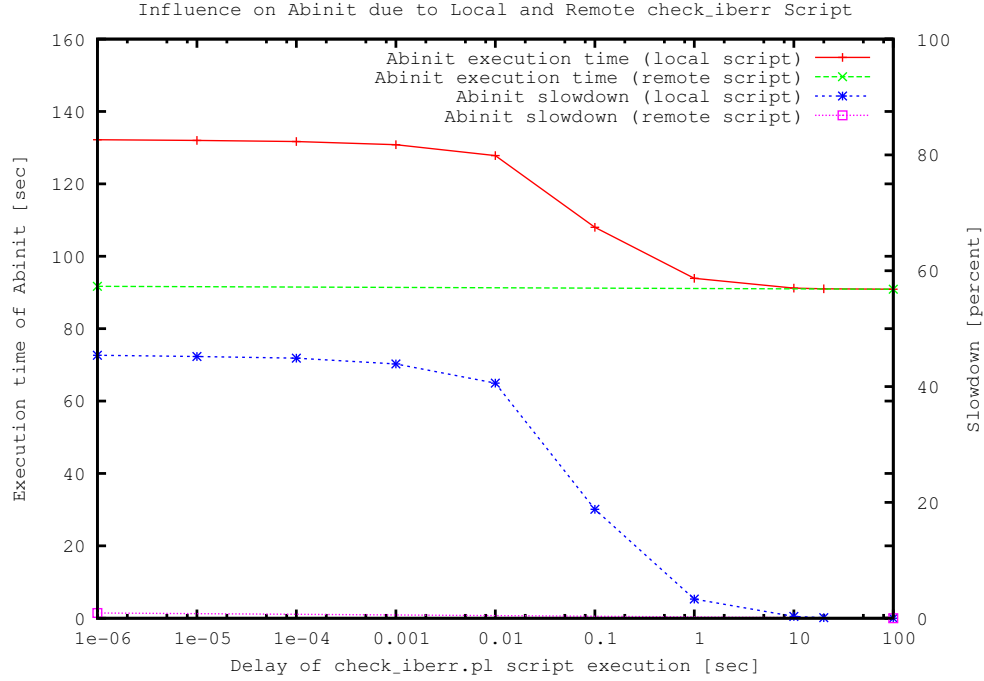


Figure 5.1: Influence on Abinit due to Local and Remote check_iberr Script Execution

interval that the expected slowdown of intervals of three or five minutes (180 to 300 seconds) will be even lower. A more detailed consideration regarding this aspect will be made explaining Figure 5.2 on the next page. The cause of the general slowdown is the CPU consumption of the NRPE plugin and the check_iberr script. A disturbance of the communication between the ABINIT jobs can be eliminated by using an extra Gigabit Ethernet network, so that the communication of ABINIT via InfiniBand network is exclusive.

Regarding the impact on ABINIT caused by the remote monitoring of the performance counters via a remote execution of the check_iberr script it can be stated that, even at a very low delay rate of $1\mu s$ between two executions, the slowdown of ABINIT was less than 1%. The reason for the very low slowdown close to the borderline of measuring errors is that the CPU load caused by the InfiniBand kernel module `ib_mad1` which is responsible for answering those queries is very low compared to the load caused by the scenario mentioned above. Obviously the impact on the ABINIT job communication via InfiniBand is not significant, probably because of the implementation of remote communication on the InfiniBand network regarding the query of the performance counters via Management Datagram (MAD) packets as described in [Inf04].

Finally, the method of querying the performance counters of an InfiniBand network card via the remote execution of the check_iberr script has a much lower slowdown on a specific ABINIT calculation with very small monitoring intervals than the local execution of it and should be preferred because of this aspect.

5.2.2 Influence on Abinit due to Local and Remote Nagios Plugins

In Figure 5.2 the execution time of ABINIT depending on the local execution of various Nagios plugins via Gigabit Ethernet network, as well as via InfiniBand network is shown. The conditions are similar to Figure 5.1 on the previous page, which means that the labelling of the axes is the same except that the abscissa displays the delay between two executions of the whole set of Nagios plugins instead of only the `check_iberr` script.

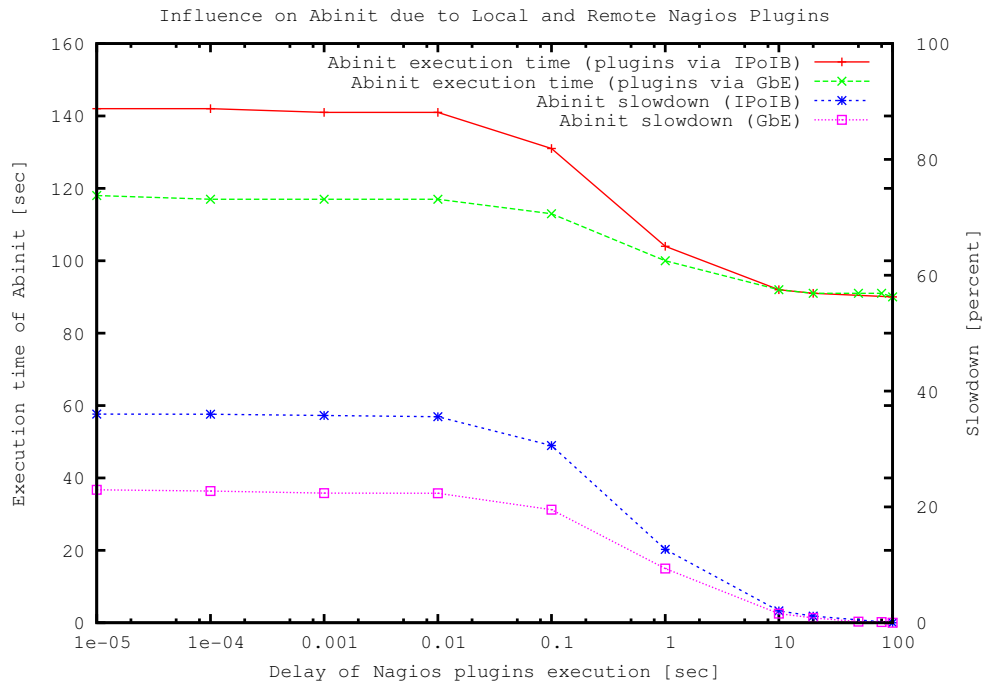


Figure 5.2: Influence on Abinit due to Local and Remote Nagios Plugins

The main purpose of this measurement is to find out the impact of a typical monitoring situation on the performance of the reference application ABINIT. The purpose is to determine whether the execution of the Nagios plugins via GbE or via IPoIB is the faster alternative. The preliminary considerations are that either the communication via the Internet Protocol over InfiniBand (IPoIB) could have less impact on ABINIT, because the InfiniBand network (10 Gb) is faster than GbE (1 Gb) or it could have more impact on ABINIT, because the network connection has to be shared unlike the communication via Gigabit Ethernet.

Table 5.1 on the facing page shows the list of plugins that were exemplarily chosen as a possible set of values for monitoring. During the use of this set of plugins for test purposes, they were executed sequentially as they appear in Table 5.1. This set of plugins, that are selected from the Nagios plugin collection⁴ in version 1.4.4, covers

⁴<http://nagiosplug.sourceforge.net/>

many values which an administrator can be interested in. They reflect a profile of what can be monitored, but they do not lay claim to be a ready to use selection. Usually the number of plugins that are used to monitor a specific machine would be smaller than the eleven plugins that are presented here, especially at a cluster computer where high performance is desired. Because every additional check module influences the performance of the system in a negative way, only modules that are really necessary should be used.

plugin name	via	description
check_load	NRPE	checks the CPU load
check_mem	NRPE	checks the memory usage
check_procs	NRPE	(1) checks the general number of processes (2) checks if the MPI daemon is running (3) checks if the <i>ipoib</i> module is loaded
check_sensors	NRPE	reads the health status (CPU temperature, fan speed, etc.) with <i>lm_sensors</i>
check_log	NRPE	checks if something has changed in <i>/var/log/messages</i>
check_ntp	network	checks if the NTP server is running and the possible time difference
check_ping	network	checks the average round trip time (RTT) and packet loss
check_ssh	network	checks if the SSH daemon is answering
check_tcp	network	checks if a daemon is listening on a specific TCP port, in this case port 80 for a web server

Table 5.1: Nagios Plugins

Figure 5.2 on the preceding page shows that the execution of monitoring applications slows down the execution of ABINIT all the more the number of disturbances increases which is equivalent to a decrease of the delay between two executions. The maximum slowdown is 23% for plugins via GbE and 36% for plugins via IPoIB, but the slowdown decreases as the delay between two sets of Nagios plugins executions increases, for example for a delay of 10 seconds the slowdown for GbE is 1.58% and for IPoIB it is 2.05%.

Furthermore, the chart shows that the execution of the plugins via the IPoIB network connection slows down ABINIT more than the execution of the same plugins via the Gigabit Ethernet network connection. This difference cannot be explained due to the plugins execution, because exactly the same set was executed via both network connections and therefore they also had the same impact on the system's CPU.

Thus, the less impact of the plugins via GbE is probably caused by the plugin communication via an extra network, so that the communication of ABINIT via another one is not affected. The other way around, the greater impact of the plugins via IPoIB is more likely because of the network connection that has to be shared among the communication for the plugins and the communication for ABINIT (refer to Figure 5.4 on page 55). This is interesting in reference to the higher bandwidth and lower latency of

the InfiniBand network. Obviously, the advantage from this is not enough to get a better performance than with the GbE communication via two separated networks. Another reason for worse performance via IPoIB network could be a potentially higher CPU consumption for processing the IPoIB communication on a specific machine which is discussed in Figure 5.3 on the next page.

Finally, the impact of the plugins regarding an application that runs on a computer can be calculated as follows. For the consideration of a usual monitoring interval from three to five minutes the run time of ABINIT of about 90 seconds is too short. This run time is good for measurement reasons as described above in Section 5.2 on page 47, but for reflections regarding practical monitoring intervals, extrapolations have to be made. Based on a slowdown of 1.58% at a delay of ten seconds via the GbE interface (refer to Figure 5.2 on page 50) which corresponds to nine sets of monitoring checks during a 90.9 *seconds* ABINIT run, the slowdown for one set of checks every 90.9 *seconds* would be: $(1.58\% / 9) = 0.175\%$. Assuming a monitoring interval of 5 *minutes* ($= 300$ *seconds*) which would mean one check every 300 *seconds* instead of one check every 90.9 *seconds*, this would reduce the influence on the application by a factor of $(300 \text{ seconds} / 90.9 \text{ seconds} = 3.30033) \approx 3.3$. Thus, for an ABINIT that runs very long, the performance loss would be $0.175\% / 3.3 \approx 0.053\%$.

To sum up, it can be stated that for a monitoring interval of five minutes for the collection of eleven plugins via Gigabit Ethernet a slowdown of just around 0.053% is expected – by using InfiniBand and a basis of 2.05% the expected slowdown would be around 0.069%. Even if a very short monitoring interval of one minute and InfiniBand is used, the maximum expected performance loss would be $0.069\% \cdot 5 = 0.345\%$. Due to the fact that usually less than eleven plugins per computer are used the expected slowdown is actually less than this, nonetheless a higher impact on the system's performance could be seen if other, uncommon types of plugins are used that for example cause a high CPU or network load.

5.2.3 Influence of Local and Remote Nagios Plugins via IPoIB and GbE on Four Local Abinis Jobs

Figure 5.3 on the facing page shows the analysis of the influence of the Nagios plugins (refer to Table 5.1 on the previous page) on the execution of the sequential version of ABINIT, called *abinis*. In contrast to *abinip* which was used for gaining the measurement data of Figure 5.1 on page 49 and Figure 5.2 on page 50, *abinis* uses, because of its sequential nature, only one CPU core for its calculations and therefore it does not perform any communication with other processes on the same machine or on a remote one. A similar input file as before was used for *abinis*, but with the difference that the complexity of it was reduced by the factor of eight, so that a single run also took about 90 seconds.

In this test, only one machine was used for getting the measuring results and one for triggering the plugins (refer to Section 5.1 on page 47). On the computer for getting the measuring data, *abinis* was started four times almost in parallel. The duration of

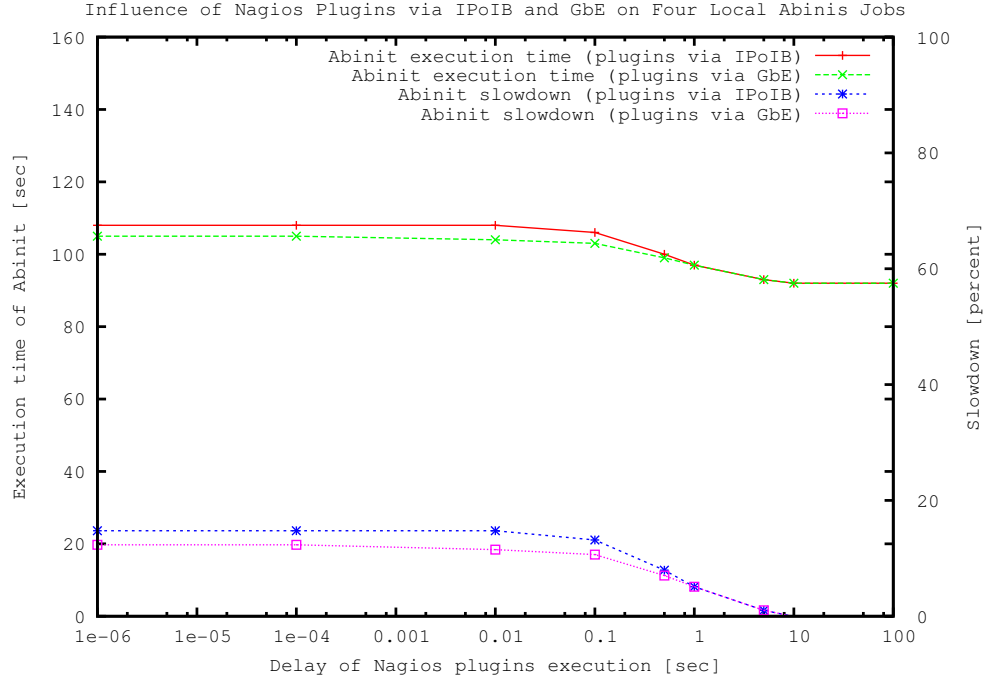


Figure 5.3: Influence of Local and Remote Nagios Plugins via IPoIB and GbE on Four Local Abinis Jobs

every single run was recorded and the average of all four runs was taken as a measuring point. This was usually repeated six times and the average of them was taken as the final measuring point for a specific Nagios plugin delay.

In addition to Figure 5.2 on page 50 the difference in CPU consumption caused by the communication via Gigabit Ethernet, as well as via InfiniBand is examined. The comparison of the progression of the GbE and IPoIB curve in Figure 5.3 shows that the impact on *abinis* is higher for a situation in which the communication of the plugins goes via InfiniBand network. Thus, because other influences like disturbances of the inter-process communication (IPC) can be excluded due to the test setup, the impact has to be traced back to a higher CPU load that is caused by communication via IPoIB. The most likely reason for this are differences in the network driver modules.

The maximum relative difference in the slowdown of *abinis* is about 17% ($(1 - (12.32\% / 14.84\%)) \cdot 100 = 16.98\% \approx 17\%$ for $x = 0.01$ seconds, $y_{GbE} = 12.32\%$, $y_{IPoIB} = 14.84\%$), compared with the results from Figure 5.2 on page 50 where plugin executions via IPoIB performed about 37% worse than via GbE ($(1 - (22.356\% / 35.58\%)) \cdot 100 = 37.1669\% \approx 37\%$ for $x = 0.01$ seconds, $y_{GbE} = 22.356\%$, $y_{IPoIB} = 35.58\%$). Since the relative slowdown of IPoIB is always higher than the GbE one in both figures, for the detailed calculation above, the point $x = 0.01$ seconds was chosen because this is where the maximum difference between the two is and therefore more accurate predictions can be made.

Thus, a 17% worse performance of IPoIB relative to GbE regarding only the CPU impact shows that the worse performance of IPoIB relative to GbE regarding an ABINIT

run with process communication via InfiniBand network of 37% is caused partially by the higher CPU load of the IPoIB network communication. It means, that only a little more than half of it is caused through the InfiniBand network connection that has to be shared among ABINIT and Nagios plugins (refer to Figure 5.2 on page 50).

5.3 Impact regarding the Network Performance

In the following section, figures are presented that show the influence of monitoring activities on the network performance. For this, the network benchmark application *netgauge*⁵ in version 1.0a1 was used ([HLR]), analogous to ABINIT with the help of MVAPICH2 MPI over InfiniBand. The test configuration as described in Section 5.1 on page 47 stays the same, which means the computers *c6-3* and *c6-4* were used for benchmarking only and the computers *c6-1* and *c6-2* for triggering the interrupt data.

The functionality of *netgauge* is that it begins with a packet size of one byte and in the process of measuring the packet size increases exponentially by doubling the preceding value to get the actual one, until the desired maximum packet size is reached – for example $2^{23} = 8388608 \text{ byte} \approx 8.4 \text{ MB}$. For every size, a packet is sent to the second computer which echoes it as soon as it has been received and when it reaches the first computer again, the total runtime is taken – after this, the next measuring of a packet with the same size can start. This is repeated several times, from around 500 to 8000 times, depending on the accuracy that is needed. For every total measurement of a specific packet size, among other values the median for it is calculated by *netgauge*. The median, which is the value for what the sum of the absolute deviations from it is minimal, is used instead of the arithmetic mean because of its resistance to outliers.

5.3.1 Network Performance with and without Remote and Local Execution of Nagios Plugins via IPoIB and GbE

In this section, by means of Figure 5.4 on the facing page, the impact on the InfiniBand network performance regarding the set of Nagios plugins as described in Table 5.1 on page 51 via GbE network, as well as via IPoIB network connection is discussed. In comparison to Figure 5.2 on page 50 and Figure 5.3 on the previous page the delay between two executions of the whole set of plugins was made zero, which means that the triggering computers tried to generate as much influence as possible. The labelling on the x-axis is the packet size in byte with a logarithmic scale and the labelling of the y-axis, which is in a logarithmic scale too, is the throughput of the InfiniBand network connection in Megabit per second (Mb/s).

The unhindered chart shows an almost linear increasing network throughput depending on the packet size at the beginning, which starts to slightly flatten past a packet size of 2^{11} byte and intensifies with further growing packet sizes. As from a size of 2^{20} byte the chart does show only marginal growth until a maximum network throughput of

⁵<http://www.unixer.de/research/netgauge/>

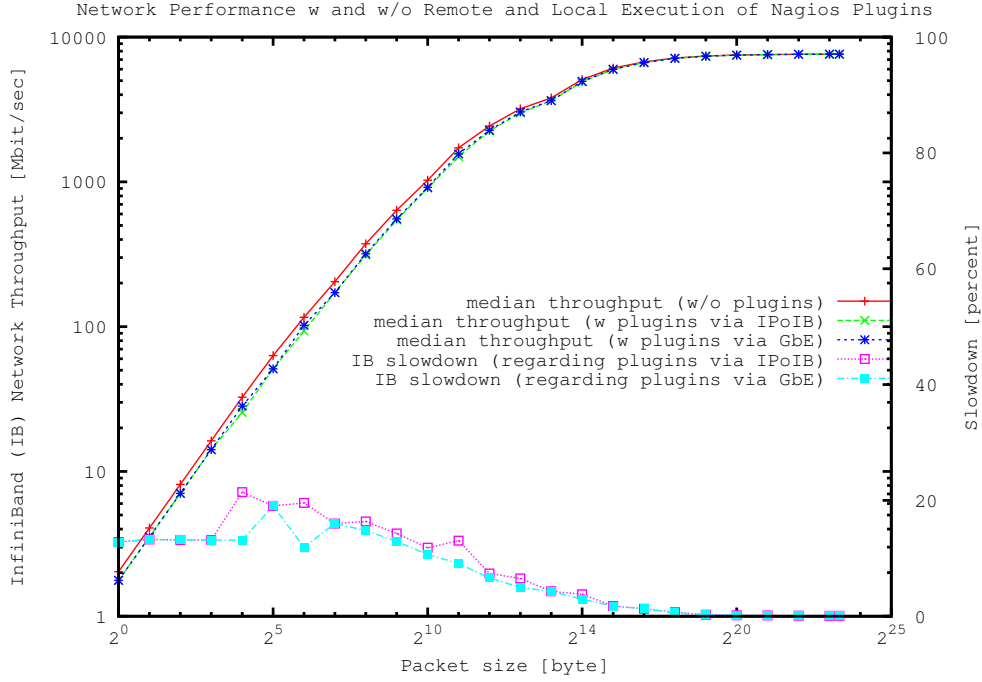


Figure 5.4: Network Performance with and without Remote and Local Execution of Nagios Plugins via IPoIB and GbE

around 7600 Mb/s is reached. The charts for the measurements of the Nagios plugins influence via Gigabit Ethernet or InfiniBand network are very similar to the unhindered one, whereas the performance is always lower in comparison to the unhindered chart. Referring to this, only a small difference in those three charts can be seen, although the labelling is already in a logarithmic scale, hence the slowdown in percent was plotted on the second y-axis due to plugin execution via GbE and IPoIB.

First, it shows that the slowdown on the InfiniBand network is always higher than the one on the GbE network. This can be explained with the fact that the plugin communication via IPoIB influences the InfiniBand performance and as a consequence of that the network performance measurement too. It also fits with the conclusion that was made at Figure 5.2 on page 50 and Figure 5.3 on page 53.

Second, the value, which shows that the IPoIB slowdown is higher compared to the GbE slowdown, is almost zero or it differs not more than 2% on most measuring points. In particular for very small and very large packet sizes – for example at $x = 2^8$ byte, $y_{GbE} = 312.36$ byte and $y_{IPoIB} = 318.15$ byte it is $(1 - (312.36 \text{ byte} / 318.15 \text{ byte})) \cdot 100 \approx 1.82\%$. Complementary to this, three significant differences can be identified. At $x = 2^4$, $x = 2^6$ and $x = 2^{11}$ byte the throughput slowdown of IPoIB differs explicitly from the GbE slowdown. A possible explanation for this are conflicts of the measured packet size with the packet sizes of specific Nagios plugins. In Figure 5.7 on page 62 the packet sizes for the test set of Nagios plugins which were used during the measurements can be seen. The peak of packets smaller than 150 byte is significant and possibly the reason for the IPoIB slowdown at $x = 16$ byte and $x = 64$ byte. The peak between

the Nagios plugins packet size of 1126 byte and 1200 byte could be causal for the InfiniBand slowdown at the network benchmark packet size of $x = 2048 \text{ byte}$, which is expected to be the packet size that is affected next for data of 1126 byte to 1200 byte.

Since at all stages of packet size measuring the same set of Nagios plugins was executed, it seems that plugin packets of a specific size have a significant impact on the InfiniBand network performance if their size is just at the size of the packets that are tested at this moment by *netgauge*.

As a reason for this, the influence of the different treatment of different packet sizes of InfiniBand is assumed. It seems that for the IB packet processing, a set of queues is used, each for a specific range of packet sizes. Hence, for the network benchmark of a specific packet size the corresponding InfiniBand processing queue is used only and very extensively, whereas the other queues are almost empty, except for a few packets for the Nagios plugins. Thus, if Nagios packets in addition to the plenty of benchmark packets of a specific size are in the queue, the possible reason for the slowdown as described above can be explained as a consequence of the load on this specific InfiniBand processing queue.

5.3.2 Network Performance with and without Execution of the `check_iberr` Script

The test setup for obtaining the charts of Figure 5.5 on the next page was used to find out the influence of the `check_iberr` script in relation to the InfiniBand network performance via GbE, as well as via IPoIB. It is very similar to the test setup of Figure 5.4 on the preceding page which means that the `check_iberr` script is executed over and over again without any delays instead of the execution of the known set of Nagios plugins. Everything else stays the same, which means the assignment of the roles to the computers and the labelling of the axes has not changed, as well as no delay between two script executions means that the triggering computers try their best to generate as much load as possible. Furthermore, the general conclusions regarding the first three charts that show the throughput depending on the packet size, without the presentation of the slowdown, are still applicable (refer to Section 5.3.1 on page 54).

The first difference in Figure 5.5 on the facing page compared to Figure 5.4 on the previous page is the percentage slowdown of the script due to execution via the two different networks. In this figure, the InfiniBand network slowdown regarding the execution of the `check_iberr` script via IPoIB is not always higher than the network slowdown that is caused by script execution via GbE. In fact, the two charts of IPoIB and GbE have very much the same progression and do differ only marginally in two measuring points each.

Although, some differences could also be explained by possible conflicts of InfiniBand queues (refer to Figure 5.4 on the preceding page), the generally more aligned progression of the two slowdown charts points to another relevant explanation for this. Note that both slowdown charts in Figure 5.5 are constantly higher than the one in Figure 5.4. Regarding the InfiniBand slowdown charts, for example the percentage

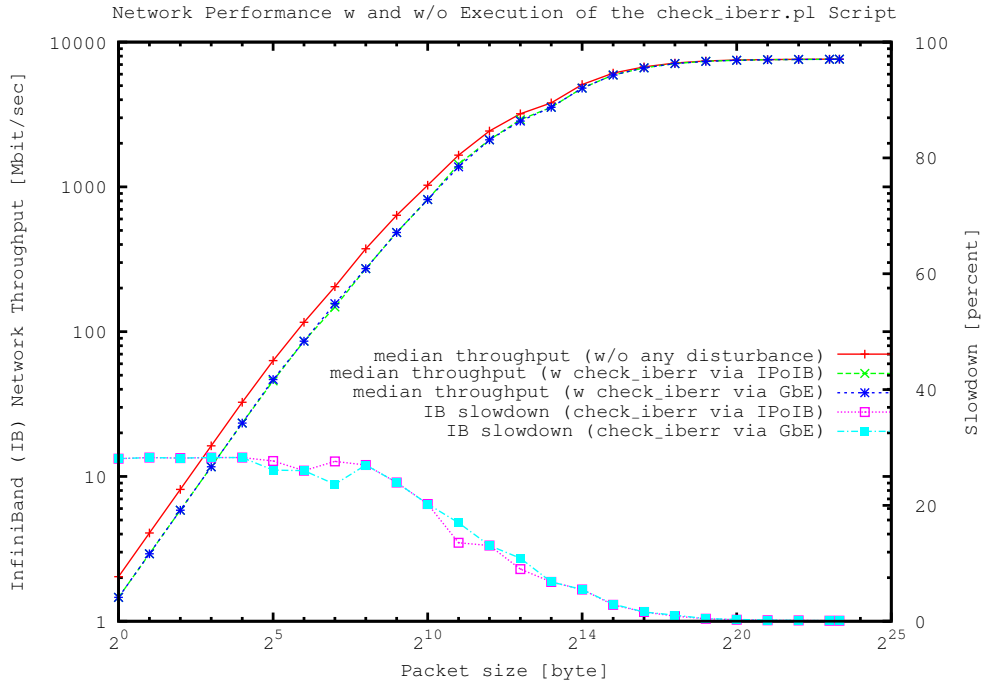


Figure 5.5: Network Performance with and without Execution of the `check_iberr` Script

slowdown for a packet size of $x = 2^0$ byte to $x = 2^3$ byte is just around 13% in Figure 5.4 and with around 28% significantly higher in Figure 5.5. In addition to that at the measuring point of $x = 2^8$ where the percentage slowdown in both figures begins its continuous decrease, the slowdown is just about 16% in Figure 5.4, but about 27% in Figure 5.5.

As a reason for this, the very different kinds of monitoring information that were gained on the one hand by the set of Nagios plugins and on the other hand by the `check_iberr` script can be taken into account. Whilst the set of Nagios plugins are checking various kinds of information that basically result in load for the CPU and memory, as well as some load on the network for the essential communication of the plugins, the `check_iberr` script primarily checks only one kind of information that mostly stresses the InfiniBand Host Channel Adapter (HCA) itself. The HCA, which is equivalent to the Network Interface Card (NIC) on other network types, is the target of the `check_iberr` script to gain the performance counter information. It means that the script checks over and over again those values and therewith interferes with the InfiniBand HCA. Because of the high frequency of the checks and the direct influence on the InfiniBand network card, the overall performance of the IB network connection is substantially lower than shown in Figure 5.4 on page 55.

5.3.3 Network Performance with and without Execution of Nagios Plugins Depending on the Delay of their Execution

In Figure 5.6 the influence on the InfiniBand network throughput regarding the packet size and the frequency of the plugin calls as a third dimension is presented. In addition to Figure 5.4 on page 55 and Figure 5.5 on the preceding page which showed the impact charts for a lot of disturbances, this figure shall demonstrate the influence on the network regarding smaller frequencies of interferences.

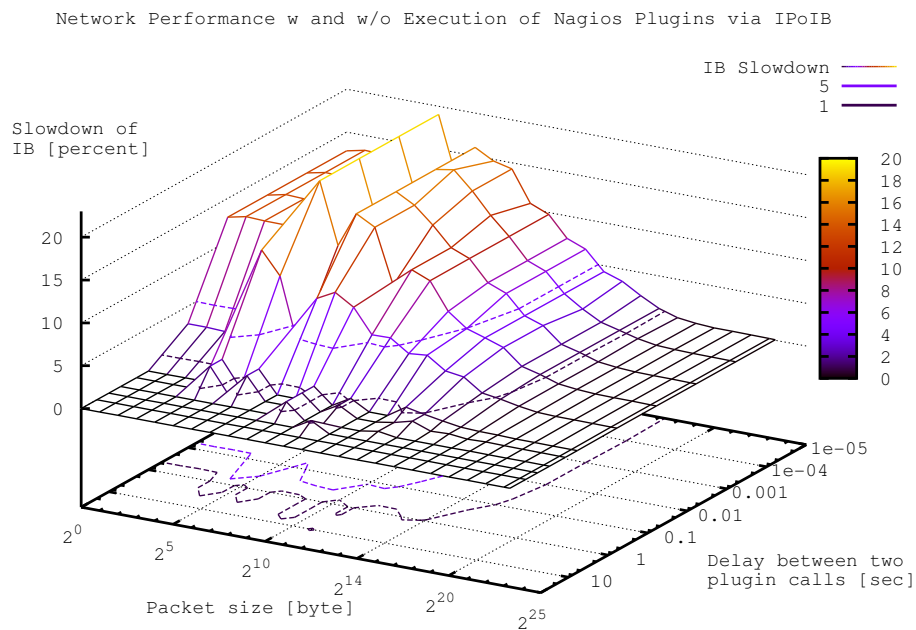


Figure 5.6: Network Performance with and without Execution of Nagios Plugins Depending on the Delay of their Execution

The position of the axis is a little different as in the two-dimensional plots, which means that in this three-dimensional figure the x-axis shows the packet size in byte, the y-axis the delay between two calls of the whole set of Nagios plugins in seconds and third, the z-axis shows the slowdown in percent of the InfiniBand network regarding the plugin execution via IPoIB based on the throughput of an untroubled network. The execution of the plugins via IPoIB was chosen because in the preceding figures the influence for IPoIB was always higher or equal to the GbE alternative and therefore conclusions from this test setup can also be adapted for the Gigabit Ethernet later on.

This figure is based on Figure 5.4 on page 55, which can be seen at plugin delays of 0.01 seconds and below on the y-axis where the slowdown chart is almost the same as the IPoIB one in Figure 5.4. Both preceding figures (5.4 and 5.5) show the maximum performance impact that can be expected because the measurement setup tried

to generate as much influence as possible. This impact can be easily recognised at the slowdown for small packet sizes, but for a realistic monitoring scenario the charts for larger delays have to be paid attention to.

Figure 5.6 on the facing page shows that an influence of less than 1% is reached with a delay of 10 seconds at most. It means that if the time between two total executions of the set of Nagios plugins is larger than 10 seconds, the influence is marginal and for a delay of 20 seconds the influence on the InfiniBand network performance cannot be detected anymore as it can be seen in the chart. Thus, for considerations concerning realistic monitoring intervals of 1 minute to 5 minutes, as made for Figure 5.2 regarding the impact on the execution time of ABINIT, no measurable influence of the network throughput is expected.

5.4 Quantitative CPU and Network Load Analysis

One aspect that has not yet been considered is the quantitative load of the CPU and the network connection of the monitoring server and its corresponding clients. For this, the known set of Nagios plugins (refer to Table 5.1 on page 51) and the `check_iberr` script are examined regarding their effects on the monitoring server, as well as regarding their effects on the clients that are monitored.

For this, the application *iptraf*⁶ was used because it has features to determine detailed network statistics regarding a specific interface. To exclude parasitic influences regarding the measurements, the configuration as described in Section 5.1 on page 47 was used. In particular the use of two network interfaces was important whereby every usual traffic like SSH, DHCP messages, and others used the first network interface, so that the second network interface could be used for untroubled measurements. For measuring the CPU load, the applications *top* and *ps* were used.

5.4.1 Influence of Nagios Plugins on Clients and the Monitoring Server

First, the quantitative influence of the set of Nagios plugins (refer to Table 5.1 on page 51) on the client computers that were monitored and second the total impact on the monitoring server was examined. The singular execution of this set of plugins generated a total of 143 packets with 29883 byte of data. It means that statistically every packet has a size of $29883 \text{ byte} / 143 \text{ packets} \approx 209 \text{ byte/packet}$ and that every packet is responsible for $143 \text{ packets} / 11 \text{ plugins} = 13 \text{ packets/plugin}$ in average.

The next test setup was based on the execution of the whole set of Nagios plugins 1000 times in a row without any delays. This took 210 seconds and generated a network traffic of around 29.8 MB which is a traffic rate of round 141 kB/sec, whereas half of it was traffic that went *in* and the other half was traffic that went *out*.

⁶<http://iptraf.seul.org/>

The execution of 1000 times 11 plugins within 210 seconds equals to $1000 \cdot 11 \text{ plugins} / 210 \text{ seconds} \approx 52.4 \text{ plugins/second}$ which means that $52.4 \text{ plugins/second} / 11 \text{ plugins} \approx 4.76 \text{ computer/second}$ can be monitored. For the CHiC system with around 550 computers to be monitored the time of $550 \text{ computer} / 4.76 \text{ computer/second} \approx 115.5 \text{ second}$ is necessary.

Thus, less than two minutes are needed to check eleven values from each computer of the CHiC. Therefore, a monitoring frequency of three to five minutes can be used without any problems. Also a very low frequency of just one minute is feasible if the plugins were executed in parallel as it is supported by Nagios and not sequentially as it was performed in the benchmark script or if fewer values for monitoring are used (refer to Section 5.2.2 on page 50). Furthermore, not the amount of data, a monitoring server can send out in parallel, was examined, only the fastest possible interaction of two computers was analysed as it was also done in the test setups in Section 5.1 on page 47 et sqq. Hence, this has no effect on the considerations because the parallel execution of the plugins on more than one computer at once tends to be faster than the strict sequential execution – an approximately realistic load on a monitoring server will be discussed below (refer to Section 5.4.3 on the facing page).

5.4.2 Influence of the `check_iberr` Script on Clients and the Monitoring Server

As the next step, the quantitative influence of the `check_iberr` script on client computers and the monitoring server shall be examined. The singular local execution of the script via the Nagios Remote Plugins Executor (NRPE) generates a total of 18 packets with 3965 byte of data.

The execution of this script 1000 times in a row without any delays took 64 seconds. Thus, a speed of $1000 \text{ executions} / 64 \text{ seconds} \approx 15.6 \text{ executions/second}$ can be reached, which means that around 550 computers of the CHiC can be monitored at least once within $550 \text{ computer} / 15.6 \text{ computer/second} \approx 32 \text{ seconds}$. Hence, the performance of the script is sufficient for realistic checking frequencies.

Above, the `check_iberr` script was executed in standard mode which reports the status of the performance counters of the InfiniBand network adapter only if they exceed a specific threshold. Usually, nothing is reported, because most of the performance counters are error counters and in a stable operation of the system they occur only occasionally. But to make sure that the monitoring of the performance counters works well, there is an update option which forces the script to read and report all counters irrespective of the exceeding of a certain threshold, as well as an option for resetting them to assure a well defined state if desired (refer to 4.4.1 on page 40).

This results in a total of 51 packets with 14931 byte of data which are generated by one singular local execution of the script with the update option via the Nagios Remote Plugins Executor (NRPE). The execution of the script 550 times in a row without any delays, which is equivalent for one update check for each of the CHiC computers, took 9 minutes and 10 seconds ($= 550 \text{ seconds}$). This seems to be a

long time, because monitoring intervals of 10 minutes or more appear possible. In a realistic scenario the update check would be made only once an hour or less frequently. Hence, to fit into the 5 minutes monitoring pattern, based on hourly intervals ($1 \text{ hour} = 60 \text{ minutes} / 5 \text{ minutes} = 12$) it has to be calculated with $550 \text{ seconds} / 12 \approx 45.8 \text{ seconds}$. Thus, in addition to the time consumption of 32 seconds of the standard `check_iberr` execution to check every CHiC computer at least once and the 45.8 seconds that are proportionally necessary for an update execution of the `check_iberr` script, to sum up an interval of 77.6 seconds is sufficient for a supposed monitoring interval of three to five minutes. Further on, the consideration made above regarding the limitations of the test setup and among others the sequential execution are still valid. The load on the monitoring server in a realistic situation will be discussed below.

5.4.3 Exemplary Monitoring Server Test with Nagios Plugins and the `check_iberr` Script

In the preceding sections some tests were performed that showed the influence of Nagios plugins and the `check_iberr` script regarding their influence on impact that have to be monitored as well as on a computer that works as a monitoring server. Those tests were good for discussing the quantitative influence in a controlled situation considering a specific aspect. In the following section the behaviour of the computer that works as the monitoring server in a realistic situation shall be examined.

For this, a computer with AMD Athlon at 950 MHz, 0.5 GB RAM under Scientific Linux 4.3 with kernel 2.6.9-34EL was used. On this computer, Nagios was installed. Further on, the four computers described in Section 5.1 on page 47 were used as the ones that should be monitored. To generate a preferably high load on the monitoring server, the known set of Nagios plugins (refer Table 5.1 on page 51) and the `check_iberr` script with the update option that generates more load as the standard version (refer to Section 5.4.2 on the facing page) were used and a monitoring interval of 10 seconds was set.

The impact on the CPU load of the monitoring machine varied from zero percent up to about five percent. During a 20 minutes = 1200 seconds interval the Nagios process consumed 26 seconds of CPU time. It means, that the average load was $(26 \text{ seconds} / 1200 \text{ seconds}) \cdot 100 = 2.1\bar{6}\% \approx 2.2\%$. Thus, for the 550 computers of the CHiC a load of $550 \cdot (2.1\bar{6}\% / 4 \text{ computers}) \approx 297.9\%$ is expected.

For a realistic monitoring interval of 5 minutes = 300 seconds instead of 10 seconds as in the measurements above, the load would be $300 \text{ seconds} / 10 \text{ seconds} = 30$ times lower and therefore only around $297.9\% / 30 = 9.93\%$. Although this is not a lot, the load would be even less, because the computer that is actually used as monitoring server in the CHiC is a much more powerful one which is faster and has more cores (refer to Section 3.1 on page 23).

Other processes like the NRPE or NSCA add-ons which assist Nagios are not neces-

sary to be examined in detail regarding their influence on the CPU, because during the test run as mentioned above they consumed less than a tenth of a second of CPU time.

The network load varied a lot, but it was never higher than 56 kB/s (*in* and *out* traffic), so that for the CHiC system it should be around $550 \text{ computer} \cdot (56 \text{ kB/s} / 4 \text{ computer}) = 7700 \text{ kB/s} = 7.7 \text{ MB/s}$ which is approximately $(100 / 125 \text{ MB/s}) \cdot 7.7 \text{ MB/s} = 6.16\%$ of a Gigabit Ethernet ($1 \text{ Gb/s} = 1000 \text{ Mb/s} / 8 = 125 \text{ MB/s}$) connection. Thus, there is enough reserve capacity for the operation of more monitoring plugins or plugins that generate a lot more traffic.

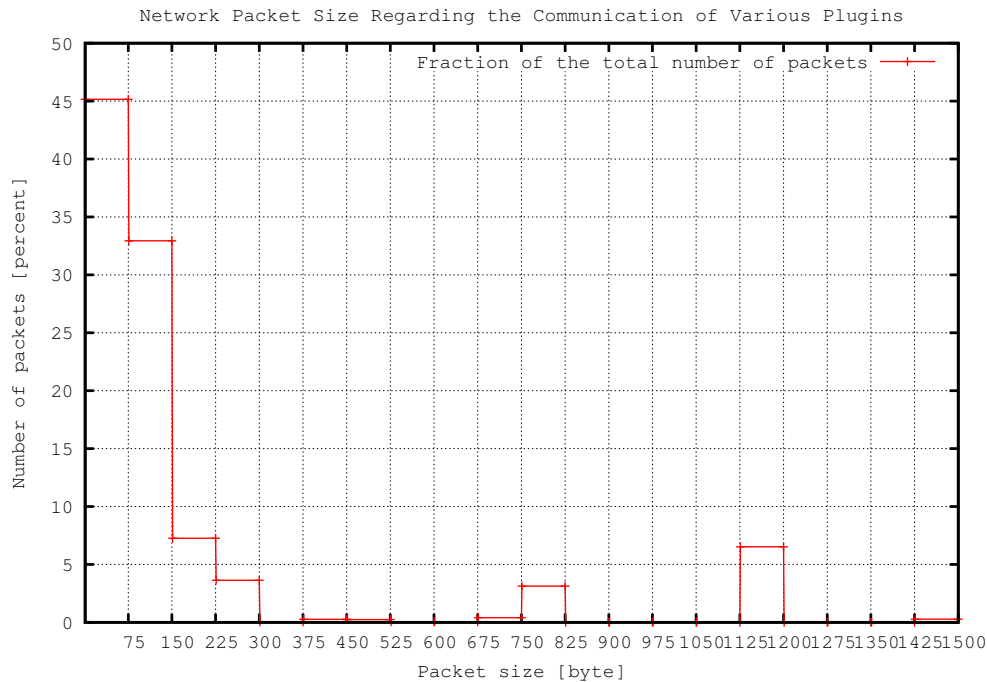


Figure 5.7: Network Packet Size Regarding the Communication of Various Nagios Plugins and the `check_iberr` Script

In addition to this, Figure 5.7 shows the spreading of the network packet sizes as they are caused by the execution of the known set of Nagios plugins and the `check_iberr` script with update option. It shows the number of packets in percent on the y-axis and the packet size in byte on the x-axis, whereas the classification in parts of 75 byte is due to the measuring with *iptraf*.

The first statement that can be made is, that about 45% of all packets are smaller than 75 byte and about 32% of all packets are between 76 and 150 byte, therefore more than three quarters ($45\% + 32\% = 78\%$) of all packets are smaller than 150 byte. Counting also packets between 151 byte and 300 byte, which are $7.3\% + 3.6\% = 11\%$ in total, almost 90% of all packets are smaller than 300 byte. Thus, most of the packets that are caused by monitoring activities are small ones and only packets between 751 byte and 825 byte with 3.1%, as well as packets between 1126 byte and 1200 byte with 6.5% show up as an exception.

Those peaks are very interesting, a further analysis of them showed that the peak between 751 byte and 825 byte is caused by the update packets that are generated by the `check_iberr` script with the update option. The peak between 1126 byte and 1200 byte is caused by the `check_tcp` plugin which is used multiple times for different purposes within the set of Nagios plugins (refer to Table 5.1 on page 51).

The influence of the relatively uneven spreading of the packet sizes can also be seen at the network throughput measurements of Figure 5.4 on page 55.

USER	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
nagios	28992	3496	2276	S	0.0	0.7	0:26.00	nagios
nagios	3496	684	572	S	0.0	0.1	0:00.08	nsca
nagios	4592	1440	1208	S	0.0	0.3	0:00.01	nrpe
worm	64572	4608	1796	R	1.0	0.2	0:00.02	check_iberr.pl

The table above shows a snapshot of the memory usage of the involved applications that was made with `top`. Its columns show in order: user name, virtual memory image (kB)⁷, resident memory size (kB), shared memory size (kB), process status (S = sleeping, R = running), CPU usage, memory usage, CPU time (min:sec.hundredth), command name.

The commands *nagios*, *nsca* and *nrpe* were executed on the monitoring server. The usage of resident memory for example of *nagios* is low with just 3496 kB ≈ 3.5 MB and it is not expected to be more than $550 \text{ computer} \cdot (3.5 \text{ MB} / 4 \text{ computer}) \approx 480 \text{ MB}$, even if a worst case scenario of a linear progression is assumed.

The *check_iberr.pl* command was recorded during its local execution via NRPE on a computer that was monitored with it. The resident memory size with round 4.6 MB is also low and it does not need to be paid a lot of attention to it, in particular because not more than one of these scripts is executed at a time on a specific monitoring client.

The execution of this script on the monitoring server as the remote version that directly monitors a specific computer without NRPE, shows the same behaviour regarding resident memory size. Considerations regarding the execution frequency of this script on one computer were made above (refer to Section 5.4.2 on page 60) and it was shown that it could be executed fast enough to monitor a CHiC-sized number of computers.

5.5 Summary

In this chapter the impact of the monitoring activities on a specific host and the network was analysed. The execution of a (monitoring) program on a computer always uses a fraction of the available resources. In the field of supercomputing often applications are executed that use all the CPU or network resources they can get. Thus, if a monitoring

⁷It has to be paid attention to the fact that the documentation of *top* misleadingly describes the unit as "kb" which would mean "kilobit", although the presented values are in fact in "kilobyte" (kB).

program also needs to be executed by the CPU, the main application is necessarily slowed down. This can be for example 0.1% or even 10% of the runtime depending on the application and on the monitoring strategy.

How much slowdown is expected for the CHiC was analysed in this chapter. For this, a sample application that is also to be used on it (ABINIT), as well as a set of values that are worth to be monitored was chosen (refer to Table 5.1 on page 51) and the effects caused by them were evaluated. In addition to that, the self-developed script `check_iberr` for monitoring the performance counters of the InfiniBand network interfaces was also embraced into the test setup (refer to Section 4.4.1 on page 40).

First, the analysis of the measurements regarding both, the set of Nagios plugins and the `check_iberr` script, showed that the slowdown of ABINIT is negligible for realistic monitoring intervals of one minute or more. The slowdown expected for an interval of one minute is definitely below 1% and for a five minutes interval it is even below 0.1% of the run time (refer to Section 5.2 on page 47 and Section 5.2.2 on page 52).

Furthermore, the behaviour of the local or remote execution of the `check_iberr` script was examined. The result is that with the remote version of the script that gains the InfiniBand performance counters via InfiniBand management packets has no measurable influence on the monitored computer. This is done by the relocation of the script execution to the monitoring server. With this the anyway small influence of the script on a monitored computer can be decreased further, so that no impact on applications that run on the monitored computer is expected regarding the utilisation of this approach (refer to Figure 5.1 on page 49).

Second, the analysis of the measurements of the InfiniBand network throughput, also regarding the set of Nagios plugins and the `check_iberr` script, showed that the slowdown is negligible too, for realistic monitoring intervals of one minute or more. Already at an interval of 10 seconds or more, the slowdown is less than 1% (refer to Figure 5.6 on page 58).

Last, an analysis, if the server on which Nagios runs can handle the entire monitoring of the CHiC was made. It showed that the CPU load and the network traffic which is expected for monitoring various values for around 550 computers is less than 10% of the CPU load and also less than 10% of a Gigabit Ethernet network connection and therefore manageable for a monitoring server with Nagios running on it (refer to Section 5.4.3 on page 61).

6 Conclusion and Outlook

This thesis dealt with the monitoring of large-scale cluster computers. There is often confusion what exactly the term *monitoring* stands for, favoured by the fact that this term is used in very different areas. Hence, for the monitoring of a cluster computer, a classification was made.

The definition of monitoring is not possible without defining the term *management*, thus a classification of it was presented. This contained a two-dimensional arrangement, that on the one hand describes five layers from enterprise and application management to information, system, and network management (refer to Section 1.2 on page 3), as well as on the other hand the FCAPS classification of the ISO/IEC 7498-4 standard that stands for fault, configuration, accounting, performance, and security management (refer to Section 1.2 on page 4).

The actual scope of monitoring a cluster computer was elaborated by adopting the management fields as described above for the action of monitoring. This is admissible because the relation of management and monitoring describes a control cycle regarding a specific entity (refer to Figure 2.1 on page 9). Hence, this thesis describes the process of monitoring as the sequence of generation, processing, dissemination and presentation of the information. The benefit from these classifications is the ability to describe which specific monitoring task belongs where in the monitoring scope so that appropriate actions can be performed at a computer system without getting in conflict with other duties and responsibilities.

Furthermore, the situation of the existing Chemnitz Linux Cluster regarding the monitoring was examined and suggestions for the improvement of the operation of it, as well as for succeeding systems, were presented. For example a lot of time and effort for maintenance can be saved by properly configured processing and dissemination of the monitoring data (refer to Section 3.2 on page 24).

As the result of considerations regarding the applicability of various free monitoring solutions at the Chemnitz High-Performance Cluster, the software Nagios was chosen as the one that fits the needs of the cluster system best (refer to 4.1 on page 34). On a cluster computer various entities have to be monitored, for example the status of the central infrastructure such as core switches or storage systems (refer to Figure 3.2 on page 27) or the health status of the racks. For most of these needs software is available that can perform such tasks. But there are other requirements that cannot be fulfilled with existing free monitoring software, such as the check of the counters of the InfiniBand network ports. Hence, during the course of this work, a plugin was developed that is able to read and process the different status information of the InfiniBand ports, unaffected by the vendor of the hardware. This software was exemplarily implemented

for the Nagios monitoring framework so that this plugin can be used as an integrated component that seamlessly fits with the monitoring approach.

In general, the monitoring of some entity does always influence the actual operation of it. Thus, the impact of the monitoring activities has to be as small as possible. Especially for a large number of computers, such as a cluster system, the scalability of the monitoring approach is very important. For this, various measurements were performed, regarding the influence of the self-developed plugin `check_iberr` and a set of representative monitoring values on the computation and the network performance of a specific system. Concerning this matter, it was evaluated how significant the expected impact is. First, a typical monitoring interval of 5 minutes was taken as a basis, but the results of this work showed that even for a very frequent interval of 1 minute the influence was always below 1% for every type of measurement. Whereas some of them showed even significantly less impact than this, for example, the remotely executed version of the self-developed `check_iberr` script had no measurable influence on the execution time of ABINIT (refer to Section 5.2.1 on page 48). Furthermore, this thesis has demonstrated, that the proposed monitoring solution would scale also on a large number of nodes, based on extrapolations from a test set with several monitoring values (refer to Section 5.4.3 on page 61).

Additionally, during the course of this work it was discovered that the InfiniBand network interface probably processes packets of different sizes in different queues, which would mean that the processing of several packets of one size would be less efficient than the processing of several packets of different sizes (refer to Section 5.3.1 on page 54). Considerations regarding this aspect should be accomplished in the future. For example a comprehensive test series could be developed that proves how far this behaviour of the InfiniBand network interface could be leveraged.

A Source Code Listing of the check_iberr Perl Script

```
1  #!/usr/bin/perl -Tw
2
3  use POSIX;
4  use strict;
5  use Getopt::Long;
6  use vars qw($opt_V $opt_h $opt_b $opt_r $opt_u $opt_m $opt_H $opt_G $opt_l
7              $opt_p $opt_c $opt_d $opt_n $opt_g $opt_f $PROGNAME);
8
9  my (%ERRORS) = ("UNKNOWN" => 3, "OK" => 0, 'WARNING' => 1, "CRITICAL" => 2);
10 $PROGNAME = "check_iberr";
11
12 sub print_revision ($$);
13 sub usage;
14 sub support();
15 sub print_help ();
16 sub print_usage ();
17
18 # empties the environment variables due to security reasons
19 $ENV{'PATH'}='';
20 $ENV{'BASH_ENV'}='';
21 $ENV{'ENV'}='';
22
23 # reads the input options of the script
24 Getopt::Long::Configure('bundling');
25 GetOptions ( "V" => \$opt_V, "version"           => \$opt_V,
26             "h" => \$opt_h, "help"              => \$opt_h,
27             "b" => \$opt_b, "bug"                => \$opt_b,
28             "r" => \$opt_r, "reset"              => \$opt_r,
29             "u" => \$opt_u, "update"             => \$opt_u,
30             "m=s" => \$opt_m, "monitoringhost=s" => \$opt_m,
31             "H=s" => \$opt_H, "hostname=s"       => \$opt_H,
32             "G=s" => \$opt_G, "portguid=s"       => \$opt_G,
33             "l=s" => \$opt_l, "lid=s"            => \$opt_l,
34             "p=s" => \$opt_p, "portnumber=s"     => \$opt_p,
35             "c=s" => \$opt_c, "critical=s"       => \$opt_c,
36             "d=s" => \$opt_d, "ofeddir=s"        => \$opt_d,
37             "n=s" => \$opt_n, "sendnscabindir=s" => \$opt_n,
38             "g=s" => \$opt_g, "sendnscacfgdir=s" => \$opt_g,
39             "f=s" => \$opt_f, "thresholdfile=s"  => \$opt_f );
40
41 # verifies the correctness of the input options
42 if ($opt_V) {
43     print_revision($PROGNAME, 'Revision: 0.4 $');
44     exit $ERRORS{'OK'};
45 }
46
47 if ($opt_h) {
48     print_help(); exit $ERRORS{'OK'};
49 }
50
51 if (($opt_r) && ($opt_b)) {
52     print "resetting performance (error) counters\n";
```

```

53 }
54
55 if (($opt_u) && ($opt_b)) {
56     print "updating all performance (error) counters\n";
57 }
58
59 ($opt_m) || usage("Warning: monitoring host not specified\n");
60 my $monhost = $1 if ($opt_m =~ /^([-.A-Za-z0-9]+)/);
61 ($monhost) || usage("Invalid address: $opt_m\n");
62
63 ($opt_H) || usage("Warning: host IP address not specified\n");
64 my $hostname = $1 if ($opt_H =~ /^([-.A-Za-z0-9]+)/);
65 ($hostname) || usage("Invalid host IP address: $opt_H\n");
66
67 $opt_G || ($opt_G = 'localhost');
68 my $portguid = $1 if ($opt_G =~ /^([-.A-Za-z0-9]+)/);
69 ($portguid) || usage("Invalid portguid address: $opt_G\n");
70
71 ($opt_l) || ($opt_l = 'none');
72 my $portlid = $1 if ($opt_l =~ /^([-.A-Za-z0-9]+)/);
73 ($portlid) || usage("Invalid LID address: $opt_l\n");
74
75 ($opt_c) || ($opt_c = 10);
76 my $critical = $1 if ($opt_c =~ /^[0-9]{1,5}|66000/);
77 ($critical) || usage("Invalid critical threshold factor\n");
78
79 ($opt_p) || ($opt_p = 1);
80 my $portnr = $1 if ($opt_p =~ /^[0-9]{1,2}|100+/);
81 ($portnr) || usage("Invalid port number (usually: 1): $opt_p\n");
82
83 ($opt_d) || ($opt_d = '/usr/ofed/bin');
84 my $sofed = $1 if ($opt_d =~ /^([-.A-Za-z0-9\/]+)/);
85 ($sofed) || usage("Invalid OFED directory (usually: /usr/ofed/bin): $opt_d\n");
86
87 ($opt_n) || ($opt_n = '/usr/bin');
88 my $sendnscabindir = $1 if ($opt_n =~ /^([-.A-Za-z0-9\/]+)/);
89 ($sendnscabindir) || usage("Invalid directory
90                             (usually: /usr/bin): $sendnscabindir\n");
91
92 ($opt_g) || ($opt_g = '/etc/nsca');
93 my $sendnscacfgdir = $1 if ($opt_g =~ /^([-.A-Za-z0-9\/]+)/);
94 ($sendnscacfgdir) || usage("Invalid directory
95                             (usually: /etc/nsca): $sendnscacfgdir\n");
96
97 my $thresholdfile='';
98 if ($opt_f) { $thresholdfile = $1 if ($opt_f =~ /^([-.A-Za-z0-9\/_]+)/);
99 }
100
101 my $lidstr='';
102 if ($portguid eq "localhost") {
103     if ($portlid eq "none") {
104         $lidstr = '$sofed/ibaddr';
105     } else { $lidstr = '$sofed/ibaddr $portlid'; }
106 } else { $lidstr = '$sofed/ibaddr -G $portguid';
107 }
108
109 my @test=split(/ /,$lidstr);
110 if ($opt_b) {print "LIDstr: $lidstr";}
111
112 # - checks if the script was executed with sufficient rights
113 # - get the LID, based on the given GID
114 if ("GID" ne $test[0] ) {
115     if ($opt_b) {
116         print "test[1]: $test[1] - Error: This script was possibly
117                 not started with superuser rights.\n";
118     }

```

```

119     alarm(2); # alarm is set to 2 seconds
120     $lidstr = '/usr/bin/sudo $ofed/ibaddr -G $portguid';
121     alarm(0); # cancel the alarm if everything is alright
122     if ($opt_b) {
123         print "LIDstr: $lidstr";
124     }
125     @test=split(/ /,$lidstr);
126     if ($test[4] eq "resolve") {
127         print "The LID that was passed does not exist.\n";
128         exit $ERRORS{'UNKNOWN'};
129     }
130     if ($test[5] eq "path_query") {
131         print "The GUID that was passed does not exist.\n";
132         exit $ERRORS{'UNKNOWN'};
133     }
134     if ("GID" ne $test[0]){
135         if ($opt_b) {
136             print "test[1]: $test[1] - Error: This script needs
137                 superuser rights (must be started as ROOT or SUDO
138                 must be configured).\n";
139         }
140         print "Script could not be executed without errors: Possibly
141             missing rights (no ROOT, no SUDO) or output format of
142             parsed tools (ibaddr) has changed.\n";
143         exit $ERRORS{'UNKNOWN'};
144     }
145     my $noroot = 1;
146 }
147
148 my $sublidstr = $1 if ($test[4] =~ /([-x0-9a-zA-Z]*)/);
149 if ($opt_b) { print "sublidstr: $sublidstr (test[4]: $test[4]\n"; }
150
151 # the performance counters are read
152 my @result='';
153 if ($opt_f) {
154     @result = '$ofed/ibcheckerrs -v -t $thresholdfile $sublidstr $portnr';
155 } else {
156     @result = '$ofed/ibcheckerrs -v $sublidstr $portnr';
157 }
158
159 my $anzresult = scalar(@result)-1;
160 my $i=0;
161 my @temp;
162 my @nsca_send;
163 my $thstr='';
164 my $thint=0;
165 my $thcrit=0;
166 my $value=0;
167 my $crit_val_ocured=0;
168 my (%th)=( 'RcvErrors', "4", 'RcvRemotePhysErrors', "5",
169             'XmtConstraintErrors', "8", 'RcvConstraintErrors', "9",
170             'SymbolErrors', "1", 'LinkRecovers', "2", 'LinkDowned', "3",
171             'RcvSwRelayErrors', "6", 'XmtDiscards', "7", 'VL15Dropped', "12",
172             'LinkIntegrityErrors', "10", 'ExcBufOverrunErrors', "11" );
173
174 # checks if the output of the performance check has the expected format
175 @temp=split(/ /,$result[$anzresult]);
176 if ("check" ne $temp[1]) {
177     # if not: try another way to check the performance counters
178     if ($opt_f) {
179         @result='/usr/bin/sudo $ofed/ibcheckerrs_patched -v -t
180             $thresholdfile $sublidstr $portnr';
181     } else {
182         @result='/usr/bin/sudo $ofed/ibcheckerrs_patched -v $sublidstr $portnr';
183     }
184     $anzresult = scalar(@result)-1;

```

```

185     @temp=split(/ /,$result[$anzresult]);
186     if ("check" ne $temp[1]) {
187         print "Script could not be executed without errors: Possibly the
188             output format of parsed tools (ibcheckerrs) has changed.\n";
189         exit $ERRORS{'UNKNOWN'};
190     }
191 }
192
193 # - if a minimum of one value exceeds a threshold, this will be reported
194 #   by sending NSCA reports to the monitoring server (one per value)
195 if (($anzresult >= 1) || ($opt_u)){
196     if ($opt_b) {
197         if (!open( WRITEME, "| $sendnscabindir/send_nasca $monhost -c
198             $sendnscacfgdir/send_nasca.cfg")) {
199             print "Script could not be executed without errors: SEND_NSCA
200                 could not be executed (executable missing?).\n";
201             exit $ERRORS{'UNKNOWN'};
202         }
203     } else {
204         if (!open( WRITEME, "| $sendnscabindir/send_nasca $monhost -c
205             $sendnscacfgdir/send_nasca.cfg 1>/dev/null")) {
206             print "Script could not be executed without errors: SEND_NSCA
207                 could not be executed (executable missing?).\n";
208             exit $ERRORS{'UNKNOWN'};
209         }
210     }
211     for ($i=0; $i<$anzresult; $i++){
212         @temp=split(/ /,$result[$i]);
213         $thstr = $temp[6];
214         chop($thstr); chop($thstr);
215         $thint = int($thstr);
216         $thcrit= $thint * $critical;
217         $value=int($temp[4]);
218         if ( int($temp[4]) < $thcrit ) {
219             print WRITEME "$hostname\tIB_$temp[2]\t1\tThreshold exceeded:
220                 $temp[4] $temp[5] $temp[6]\n\n";
221             if ($opt_b) {
222                 print "warning"; print " --- temp4: --$temp[4]--;
223                     value: --$value--; thcrit: --$thcrit--\n";
224             }
225         } else { print WRITEME "$hostname\tIB_$temp[2]\t2\tThreshold
226             exceeded: $temp[4] (Critical: $thcrit)\n\n";
227             if ($opt_b) {print "critical"; print " --- temp4: --$temp[4]--;
228                 value: --$value--; thcrit: --$thcrit--\n";
229             }
230             $crit_val_ocured=$crit_val_ocured+1;
231         }
232         delete( $th{$temp[2]} );
233     }
234
235     my @errnames = keys( %th );
236     for ($i=0;$i<scalar(@errnames);$i++) {
237         print WRITEME "$hostname\tIB_$errnames[$i]\t0\tOK:
238             value below threshold\n\n";
239         if ($opt_b) {
240             print "OK: "; print " --- errnames
241                 --$errnames[$i]--; i: --$i--;\n";
242         }
243     }
244     if ($opt_b) {
245         print "Errornames without exceeding a threshold:
246             @errnames; Total: "; print scalar(@errnames);
247     }
248     close(WRITEME);
249 }
250

```

```

251 # if the reset option is set: the performance counters are reseted
252 my $errcode=0;
253 if ($opt_r) {
254     $errcode = system("$ofed/perfquery -R $sublidstr $portnr");
255     if ($errcode != 0) {
256         $errcode=0;
257         $errcode = system("/usr/bin/sudo $ofed/perfquery -R $sublidstr $portnr");
258         if ($errcode != 0) {
259             if ($opt_b) { print "\nSomething went wrong! Errcode: $errcode\n"; }
260             exit $ERRORS{'UNKNOWN'};
261         }
262     }
263     if ($opt_b) { print "\nerrcode: $errcode (should be '0')\n"; }
264 }
265
266
267
268 if ($opt_b) {
269     print "Following error counters had values above
270           threshold ($anzresult total): \n";
271     for ($i=0; $i<$anzresult+1; $i++){
272         print "$result[$i]";
273     }
274 }
275
276 # - if critical errors or warnings occurred or everything was alright
277 #   different return values are passed to Nagios
278 if ($crit_val_ocured>0) {
279     print "$crit_val_ocured value(s) exceeded critical threshold.\n";
280     exit $ERRORS{'CRITICAL'};
281 }
282
283 if ($anzresult>0) {
284     print "$anzresult value(s) exceeded warning threshold.\n";
285     exit $ERRORS{'WARNING'};
286 }
287
288 if ($opt_r) {
289     print "Resetting of all performance (error) counters successful.\n";
290 } else {
291     print "everything alright\n";
292 }
293
294 exit $ERRORS{'OK'};
295
296 # a few subroutines are defined as follows:
297 sub print_usage () {
298     print "Usage: $PROGNAME [-r] [-u] -H <hostname> [-m <monitoringhost>]
299           [-G <portguid>] [-l <lid>] [-p <portnumber>] [-c <crit>]
300           [-d <dir-ofed>] [-n<dir-send_nsca>] [-g <dir-send_nsca.cfg>]
301           [-f <thresholdfile>] \n";
302 }
303
304 sub print_help () {
305     print_revision($PROGNAME,'$Revision: 0.4 $');
306     print "Copyright (c) 2007 Stefan Worm
307
308 This plugin reports if errors at ports of an InfiniBand interface have occurred.
309
310 ";
311     print_usage();
312     print "
313
314 -b, --bug
315     prints debug messages to STDOUT
316 -r, --reset

```

```

317     reset all performance (error) counters
318 -u, --update
319     update all performance (error) values
320 -H, --hostname=STRING
321     Name of the host in which the IB should be checked
322     (exactly the same as defined in Nagios)
323 -m, --monitoringhost=IP-Address
324     IP address of the monitoring host
325     (To where the results of this script should be sent to?)
326 -G, --portguid=HEX
327     portguid number of the IB device to be checked (Default: localhost)
328 -l, --lid=HEX
329     lid number of the IB device to be checked
330 -p, --portnumber=INTEGER
331     portnumber of the IB device to be checked (DEFAULT: 1)
332 -c, --critical=INTEGER
333     factor of the exceeding of the warning-threshold when
334     a CRITICAL status will result (DEFAULT: 10)
335 -d, --dirofed=full-directory-path
336     Full directory path for the ofed-tools (DEFAULT: /usr/ofed/bin)
337 -n, --sendnscabindir=full-directory-path
338     Full directory path for the send_nsca binary (DEFAULT: /usr/bin)
339 -g, --sendnscacfgdir=full-directory-path
340     Full directory path for the send_nsca.cfg config file (DEFAULT: /etc/nsca)
341 -f, --filename=threshold-file
342     Custom thresholds file with full path
343     (DEFAULT THRESHOLD: between 10 or 100 depending on the value)
344
345 ";
346     support();
347 }
348 sub print_revision ($$) {
349     my $commandName = shift;
350     my $pluginRevision = shift;
351     $pluginRevision =~ s/^\$Revision: //;
352     $pluginRevision =~ s/ \$\$*$/;
353     print "$commandName (nagios-plugins 1.4.4) $pluginRevision\n";
354     print "This nagios plugin come with ABSOLUTELY NO WARRANTY. You may
355     redistribute copies of the plugin under the terms of the GNU
356     General Public License. For more information about these
357     matters, see the file named COPYING.\n";
358 }
359
360 sub support () {
361     my $support='Send email to the author if you have questions\n
362     regarding use of this software. ';
363     $support =~ s/\/\@\/g;
364     $support =~ s/\/\n\/n\/g;
365     print $support;
366 }
367
368 sub usage {
369     my $format=shift;
370     printf($format,@_);
371     exit $ERRORS{'UNKNOWN'};
372 }

```

Listing A.1: check_iberr.pl

B Monitoring Server and Client Configuration

B.1 Definition of Hosts and Services on the Monitoring Server

On the Nagios monitoring server the following configurations regarding the hosts and services that should be monitored have to be performed to set up an exemplary monitoring scenario, to understand the situation under which the analyses of this work were made.

The following Nagios configuration example creates a situation in which the host named `c5-2` should be monitored regarding its InfiniBand (IB) error counters via Nagios Remote Plugins Executor (NRPE). This is appropriate if the Nagios monitoring server is not connected via InfiniBand with the host that should be monitored. However, if it is connected, the `check_iberr.pl` script can also be executed directly on the Nagios monitoring host and the error counters check could be made directly via IB-network which performs much better than the version via NRPE.

The configuration causes the error counters at the monitoring clients to be checked every 5 minutes and only if the state of an error counter has changed (threshold exceeded) this will be (passively) reported to the Nagios server. Furthermore, every 59 minutes the status of all values will be passively updated and reported to the Nagios server, as well as every 24 hours all error counters will be reseted.

```
1 define host{
2     use                                generic-host                ; - Name of host template to use
3                                     ; (Nagios-Standard-Template)
4     host_name                          c5-2
5     alias                              Compute-5-2
6     address                            192.168.1.52
7     check_command                      check-host-alive
8     parents                            jack
9     max_check_attempts                 10
10    check_period                       24x7
11    notification_interval              120
12    notification_period                24x7
13    notification_options               d,r
14    contact_groups                     admins
15 }
16
17 define service{
18     use                                generic-service            ; - Name of service template to use
19                                     ; (Nagios-Standard-Template)
20     host_name                          c5-2
21     service_description                iberr_nsca_trigger
```

```

22     is_volatile                0
23     check_period               24x7
24     max_check_attempts        4
25     normal_check_interval     5
26     retry_check_interval      1
27     contact_groups            admins
28     notification_options      w,u,c,r
29     notification_interval     960
30     notification_period       24x7
31     check_command             ibcheckerr !0x0002c9010ad27db1 ;InfiniBand PortGUID
32 }
33
34 define service{
35     use                        generic-service
36     host_name                  c5-2
37     service_description       iberr_nsca_update_trigger
38     is_volatile                0
39     check_period               24x7
40     max_check_attempts        4
41     normal_check_interval     59 ;min
42     retry_check_interval      1
43     contact_groups            admins
44     notification_options      w,u,c,r
45     notification_interval     960
46     notification_period       24x7
47     check_command             ibcheckerr_update !0x0002c9010ad27db1
48 }
49
50 define service{
51     use                        generic-service
52     host_name                  c5-2
53     service_description       iberr_nsca_reset_trigger
54     is_volatile                0
55     check_period               24x7
56     max_check_attempts        4
57     normal_check_interval     1440 ;in minutes (1440 min. equals 1 day)
58     retry_check_interval      1
59     contact_groups            admins
60     notification_options      w,u,c,r
61     notification_interval     1960
62     notification_period       24x7
63     check_command             ibcheckerr_reset !0x0002c9010ad27db1
64 }
65
66 define service{
67     name                       generic-iberrors-service ;the 'name' of this template
68     active_checks_enabled      0 ; Active service checks are enabled
69     passive_checks_enabled     1 ; Passive service checks are enabled/accepted
70     parallelize_check          1 ; Active service checks should be parallelized
71                               ; (disabling this can lead to major performance
72                               ; problems)
73     obsess_over_service        1 ; We should obsess over this service
74                               ; (if necessary)
75     check_freshness            1 ; Default is NOT to check service 'freshness'
76     freshness_threshold        3600 ; seconds
77     notifications_enabled      1 ; Service notifications are enabled
78     event_handler_enabled      1 ; Service event handler is enabled
79     flap_detection_enabled     1 ; Flap detection is enabled
80     failure_prediction_enabled 1 ; Failure prediction is enabled
81     process_perf_data          1 ; Process performance data
82     retain_status_information   1 ; Retain status information across program
83                               ; restarts
84     retain_nonstatus_information 1 ; Retain non-status information across program
85                               ; restarts
86     is_volatile                0
87     check_period               24x7

```

```

88     max_check_attempts          4
89     normal_check_interval       5
90     retry_check_interval        1
91     contact_groups               admins
92     notification_options         w,u,c,r
93     notification_interval        960
94     notification_period          24x7
95     servicegroups                iberrors
96     check_command                check_dummy_iberrors!!"The status of this passive
97                                 value is not up to date any longer
98                                 - something could be wrong"
99                                 ; - if the freshness_threshold is exceeded,
100                                ;   this command is executed (it returns the
101                                ;   status of 'Warning')
102     register                     0   ; - DON'T REGISTER THIS DEFINITION - ITS NOT
103                                ;   A REAL SERVICE, JUST A TEMPLATE!
104 }
105
106 define service{
107     use                          generic-iberrors-service ; Name of service template to use
108     host_name                    c5-2
109     service_description          IB_SymbolErrors
110 }
111 define service{
112     use                          generic-iberrors-service
113     host_name                    c5-2
114     service_description          IB_LinkRecovers
115 }
116 define service{
117     use                          generic-iberrors-service
118     host_name                    c5-2
119     service_description          IB_LinkDowned
120 }
121 define service{
122     use                          generic-iberrors-service
123     host_name                    c5-2
124     service_description          IB_RcvErrors
125 }
126 define service{
127     use                          generic-iberrors-service
128     host_name                    c5-2
129     service_description          IB_RcvRemotePhysErrors
130 }
131 define service{
132     use                          generic-iberrors-service
133     host_name                    c5-2
134     service_description          IB_RcvSwRelayErrors
135 }
136 define service{
137     use                          generic-iberrors-service
138     host_name                    c5-2
139     service_description          IB_XmtDiscards
140 }
141 define service{
142     use                          generic-iberrors-service
143     host_name                    c5-2
144     service_description          IB_XmtConstraintErrors
145 }
146 define service{
147     use                          generic-iberrors-service
148     host_name                    c5-2
149     service_description          IB_RcvConstraintErrors
150 }
151 define service{
152     use                          generic-iberrors-service
153     host_name                    c5-2

```

```

154     service_description    IB_LinkIntegrityErrors
155 }
156 define service{
157     use                     generic-iberrors-service
158     host_name               c5-2
159     service_description    IB_ExcBufOverrunErrors
160 }
161 define service{
162     use                     generic-iberrors-service
163     host_name               c5-2
164     service_description    IB_VL15Dropped
165 }

```

Listing B.1: Configuration of the Hosts and Services on the Nagios Server

B.2 Definition of the Check Commands on the Monitoring Server for Direct Execution

If the `check_iberr.pl` script should be executed locally and a direct performance counter check of the InfiniBand network device should be performed, the following definitions that have to be made on the monitoring server, can be used as a blueprint.

```

1 define command{
2     command_name    ibcheckerr
3     command_line    /usr/CHECK-DIR/check_iberr.pl -m 192.168.1.98 -H $ARG1$ -G $ARG1$
4 }

6 define command{
7     command_name    ibcheckerr_update
8     command_line    /usr/CHECK-DIR/check_iberr.pl -m 192.168.1.98 -H $ARG1$ -G $ARG1$ -u
9 }

11 define command{
12     command_name    ibcheckerr_reset
13     command_line    /usr/CHECK-DIR/check_iberr.pl -m 192.168.1.98 -H $ARG1$ -G $ARG1$ -r
14 }

```

Listing B.2: Nagios Server Direct Command Execution Configuration

B.3 Definition of the Check Commands on the Monitoring Server for Execution via NRPE

If the `check_iberr.pl` script should be executed remotely via NRPE the following definitions, that have to be made on the monitoring server, can be used as a blueprint as well.

```

1 define command{
2     command_name    ibcheckerr
3     command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_iberr 2
4                     -a 192.168.1.98 $HOSTNAME$ $ARG1$
5 }

```

```
6 define command{
7   command_name    ibcheckerr_update
8   command_line     $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_iberr_update 2
                        -a 192.168.1.98 $HOSTNAMES$ $ARG1$
9 }

11 define command{
12   command_name    ibcheckerr_reset
13   command_line     $USER1$/check_nrpe -H $HOSTADDRESS$ -c check_iberr_reset 2
                        -a 192.168.1.98 $HOSTNAMES$ $ARG1$
14 }
```

Listing B.3: Nagios Server Command Execution Configuration via NRPE

B.4 Definitions on the Monitoring Client

The configuration of the Nagios Remote Plugin Executor (NRPE) on the host that should be monitored is exemplary presented for the option that the `check_iberr.pl` script should be executed locally on this host. Of course, the path usually has to be changed regarding where the `check_iberr.pl` script is located on the specific host.

If one or more of the several options of the `check_iberr.pl` script are used, they should be added at this point. The program parameters that are expected below, are the ones that were passed by the command definition on the monitoring server to the NRPE (refer to Section B.3 on the facing page).

```
1  command[ check_iberr ]      = /usr/CHECK-DIR/check_iberr.pl 2
                                -m $ARG1$ -H $ARG2$ -G $ARG3$
2  command[ check_iberr_update ] = /usr/CHECK-DIR/check_iberr.pl 2
                                -m $ARG1$ -H $ARG2$ -G $ARG3$ -u
3  command[ check_iberr_reset ] = /usr/CHECK-DIR/check_iberr.pl 2
                                -m $ARG1$ -H $ARG2$ -G $ARG3$ -r
```

Listing B.4: NRPE Monitoring Client Configuration

Bibliography

- [Bal05] Tarus Balog: *Enterprise-Wide Network Management with OpenNMS*. O'Reilly SysAdmin, 2005.
URL <http://www.oreillynet.com/pub/a/sysadmin/2005/09/08/opennms.html> [p. 33]
- [Boo03] Charles Bookman: *Linux Clustering: Building and Maintaining Linux Clusters*. New Riders Publishing, Indianapolis, 2003. ISBN 978-1-57870-274-9. [p. 2]
- [Clo53] Charles Clos: *A study of non-blocking switching networks*. *Bell System Technical Journal*, Volume 32 (Number 2), March 1953:pp. 406–424. [p. 23]
- [CS92] Dah Ming Chiu and Ram Sudama: *Network Monitoring Explained: Design and Application*. Ellis Horwood, Chichester, UK, 1992. ISBN 0-13-614710-0. [p. 4, 16]
- [CWP03] B. Chandrasekaran, Pete Wyckoff and Dhabaleswar K. Panda: *MIBA: A Micro-Benchmark Suite for Evaluating InfiniBand Architecture Implementations*. *Lecture Notes in Computer Science - Computer Performance*, Volume 2497, Springer, Berlin / Heidelberg, 2003:pp. 29–46. ISSN 302-9743. (ISBN 978-3-540-40814-7).
URL <http://dx.doi.org/10.1007/b12028> [p. 39]
- [CWSC01] Stephen Chan, Cary Whitney, Iwona Sakreja and Shane Canon: *Monitoring Tools for Larger Sites*. In: *login: The Magazine of USENIX & SAGE*, Volume 26, Number 5, August 2001.
URL <http://www.usenix.org/publications/login/2001-08/pdfs/chan.pdf> [p. 7, 32]
- [Dib02] Peter C. Dibble: *Real-Time Java Platform Programming*. Prentice Hall, Palo Alto, CA, 2002. ISBN 978-0-13-028261-3. [p. 2]
- [Fri02] Aileen Frisch: *Essential System Administration*. O'Reilly, Sebastopol, CA, 2002. ISBN 0-596-00343-9. [p. 34]
- [Gal06] Ethan Galstad: *Nagios Version 2.x Documentation*, 2006.
URL http://nagios.sourceforge.net/docs/2_0/ [p. 34]

- [GBC⁺02] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, Ph. Ghosez, J.-Y. Raty and D.C. Allan: *First-principles computation of material properties : the ABINIT software project*. *Computational Materials Science* 25, 478–492, 2002.
URL [http://dx.doi.org/10.1016/S0927-0256\(02\)00325-7](http://dx.doi.org/10.1016/S0927-0256(02)00325-7) [p. 23, 47]
- [Got02] Ellen Gottesdiener: *Requirements by Collaboration: Workshops for Defining Needs*. Addison Wesley Professional, 2002. ISBN 978-0-201-78606-4. [p. 25]
- [GSP05] Susan L. Graham, Marc Snir and Cynthia A. Patterson, editors: *Getting up to speed: The future of supercomputing*. The National Academies Press of the National Research Council, Washington, D.C., 2005. ISBN 0-309-09502-6. [p. 2, 3]
- [GWB⁺04] Michael Gerndt, Roland Wismüller, Zoltán Balaton, Gábor Gombás, Péter Kacsuk, Zsolt Németh, Norbert Podhorszki, Hong-Linh Truong, Thomas Fahringer, Marian Bubak, Erwin Laure and Thomas Margalef: *Performance Tools for the Grid: State of the Art and Future*. Working Group on Automatic Performance Analysis: Real Tools (APART), 2004.
URL <http://urza.lpds.sztaki.hu/~zsnemeth/apart/repository/gridtools.pdf> [p. 34]
- [HA94] Heinz-Gerd Hegering and Sebastian Abeck: *Integrated network and system management*. Addison-Wesley, Wokingham, UK, 1994. ISBN 0-201-59377-7. [p. 3]
- [Hal00] Eric A. Hall: *Internet Core Protocols*. O'Reilly, Sebastopol, CA, 2000. ISBN 1-56592-572-6. [p. 11]
- [HAN99] Heinz-Gerd Hegering, Sebastian Abeck and Bernhard Neumair: *Integrated Management of Networked Systems: Concepts, Architectures, and Their Operational Application*. Morgan Kaufmann, 1999. ISBN 1-55860-571-1. [p. 3, 4]
- [Har03] Richard C. Harlan: *Network Management with Nagios*. *Linux Journal*, (Number 111), 2003. ISSN 1075-3583.
URL <http://portal.acm.org/citation.cfm?id=860378> [p. 34]
- [HLR] Torsten Hoefler, André Lichei and Wolfgang Rehm: *Low-Overhead LogGP Parameter Assessment for Modern Interconnection Networks*. TU Chemnitz. presented in Long Beach, CA, USA, Mar. 2007, Accepted for publication at the 6th International Workshop on Performance Modelling,

- Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS) 2007 in conjunction with IEEE International Parallel & Distributed Processing Symposium (IPDPS) 2007. [p. 54]
- [Hoe05] Torsten Hoefler: *Evaluation of publicly available Barrier-Algorithms and Improvement of the Barrier-Operation for large-scale Cluster-Systems with special Attention on InfiniBand Networks*. Diploma Thesis, Technical University of Chemnitz, Faculty of Computer Science, Germany, 2005.
URL <http://archiv.tu-chemnitz.de/pub/2005/0073/data/diploma.pdf> [p. 39]
- [Inf04] InfiniBand Trade Association (IBTA): *InfiniBand Architecture Specification Volume 1, Release 1.2*. 2004. [p. 39, 49]
- [ISO89] ISO/IEC (JTC 1) 7498-4: *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 4: Management framework*, 1989.
URL [http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s014258_ISO_IEC_7498-4_1989(E).zip). [p. 4]
- [ISO94] ISO/IEC (JTC 1) 7498-1: *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*, 1994.
URL [http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994\(E\).zip](http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip). [p. 4]
- [JLSU87] Jeffrey Joyce, Greg Lomow, Konrad Slind and Brian Unger: *Monitoring distributed systems*. *ACM Transactions on Computer Systems (TOCS)*, Volume 5 (Number 2), ACM Press, New York, 1987:pp. 121–150. ISSN 0734-2071.
URL <http://doi.acm.org/10.1145/13677.22723> [p. 7]
- [Kan02] Stephan H. Kan: *Metrics and Models in Software Quality Engineering, Second Edition*. Addison Wesley Professional, 2002. ISBN 978-0-201-72915-3. [p. 18]
- [Lan94] Alwyn Langsford: *OSI Management Model and Standards*. In Morris Sloman, editor, *Network and distributed systems management*, Addison-Wesley, Wokingham, UK. 1994. ISBN 0-201-62745-0, pp. 69–93. [p. 4]
- [LH02] Thomas A. Limoncelli and Christine Hogan: *The Practice of System and Network Administration*. Addison-Wesley – Pearson Education, 2002. ISBN 0-201-70271-1. [p. 7, 16, 21]
- [Lib00] Hastings Maboshe Libati: *Network Traffic Analysis and Security Monitoring to Detect Intrusions*. Dissertation, Friedrich-Schiller-University Jena, School for Mathematics and Computer Science, Germany, 2000. [p. 4]

- [LJ93] Rubin H. Landau and Paul J. Fink Jr.: *A Scientist's and Engeneer's Guide to Workstations and Supercomputers: coping with Unix, RISC, Vectors, and programming*. John Wiley & Sons, New York, 1993. ISBN 0-471-53271-1. [p. 1]
- [LNC98] Soung C. Liew, Ming-Hung Ng and Cathy W. Chan: *Blocking and non-blocking multirate Clos switching networks*. *IEEE/ACM Transactions on Networking (TON)*, Volume 6 (Number 3), IEEE Press, Piscataway, NJ, 1998:pp. 307–318. ISSN 1063-6692.
URL <http://dx.doi.org/10.1109/90.700894> [p. 23]
- [Luc04] Robert W. Lucke: *Building Clustered Linux Systems*. Prentice Hall, Upper Saddle River, NJ, 2004. ISBN 978-0-13-144853-7. [p. 2]
- [man05] "manage": *Merriam-Webster Online Dictionary*, 2005. (05 Oct. 2006).
URL <http://www.merriam-webster.com> [p. 3]
- [MAS⁺03] James McGovern, Scott W. Ambler, Michael E. Stevens, James Linn, Vikas Sharan and Elias K. Jo: *A Practical Guide to Enterprise Architecture*. Prentice Hall, Upper Saddle River, NJ, 2003. ISBN 978-0-13-141275-0. [p. 21]
- [MCC04] Matthew L. Massie, Brent N. Chun and David E. Culler: *The Ganglia Distributed Monitoring System: Design, Implementation, and Experience*. *IN: Parallel Computing*, Volume 30, Issue 7, 2004:pp. 817–840. ISSN 0167-8191. (<http://ganglia.info/papers/science.pdf>).
URL <http://dx.doi.org/10.1016/j.parco.2004.04.001> [p. 33]
- [MHS06] Yusef Hassan Montero and Victor Herrero-Solana: *Improving Tag-Clouds as Visual Information Retrieval Interfaces*. To appear in: International Conference on Multidisciplinary Information Sciences and Technologies (InSciT), Mérida, Spain, 2006.
URL http://www.nosolousabilidad.com/hassan/improving_tagclouds.pdf [p. 21]
- [MS01] Douglas R. Mauro and Kevin J. Schmidt: *Essential SNMP*. O'Reilly, Sebastopol, CA, 2001. ISBN 0-596-00020-0. [p. 11]
- [MSS94] Masoud Mansouri-Samani and Morris Sloman: *Monitoring Distributed Systems*. In Morris Sloman, editor, *Network and distributed systems management*, Addison-Wesley, Wokingham, UK. 1994. ISBN 0-201-62745-0, pp. 303–347. [p. 8, 9, 19]
- [Mur00] Richard Murch: *Project Management: Best Practices for IT Professionals*. Prentice Hall, Upper Saddle River, NJ, 2000. ISBN 978-0-13-021914-5. [p. 25]

- [Neu88] Victoria Neufeldt, editor: *Webster's New World Dictionary of American English, Third College Edition*. Webster's New World Dictionaries - A Division of Simon & Schuster, Inc., New York, 1988. ISBN 0-13-947169-3. [p. 1, 3, 7]
- [Nie04] Jakob Nielsen: *Usability engineering*. Kaufmann, Amsterdam, 2004. ISBN 0-12-518406-9. [p. 21]
- [O'D02] Shane O'Donnell: *Network Management with OpenNMS*. O'Reilly ONLamp.com, 2002.
URL <http://www.onlamp.com/pub/a/onlamp/2002/04/18/opennms.html> [p. 33]
- [per05] "perform": *Merriam-Webster Online Dictionary*, 2005. (23 Oct. 2006).
URL <http://www.merriam-webster.com> [p. 11]
- [Pfi01] Gregory F. Pfister: *Aspects of the InfiniBand Architecture*. IN: *IEEE International Conference on Cluster Computing, Proceedings*, 2001:pp. 369–371. ISSN 0272-5428. (ISBN: 0-7695-1390-5). [p. 39]
- [PKP03] Fabrizio Petrini, Darren J. Kerbyson and Scott Pakin: *The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q*. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, Washington, DC, USA, 2003. ISBN 1-58113-695-1, p. 55. [p. 13]
- [RFC81] RFC 792: *Internet Control Message Protocol (ICMP)*, 1981.
URL <http://www.ietf.org/rfc/rfc792.txt> [p. 11]
- [sca05] "scalable": *Merriam-Webster Online Dictionary*, 2005. (23 Oct. 2006).
URL <http://www.merriam-webster.com> [p. 13]
- [SDA⁺00] Anthony Skjellum, Rossen Dimitrov, Srihari Angulari, David Lifka, George Coulouris, Putchong Uthayopas, Stephen L. Scott and Rasit Eskioglu: *Systems Administration*. In Mark Baker, editor, *Cluster Computing White Paper*, chapter 6. 2000.
URL <http://arxiv.org/pdf/cs.DC/0004014> [p. 13]
- [Sel00] John Sellens: *System and Network Monitoring*. In: *login: The Magazine of USENIX & SAGE*, Volume 25, Number 3, June 2000.
URL <http://www.usenix.org/publications/login/2000-6/features/monitoring.html> [p. 10, 32]
- [SK87] Morris Sloman and Jeff Kramer: *Distributed Systems and Computer Networks*. Prentice Hall, 1987. ISBN 0-13-215864-7. [p. 13]

- [SKMC03] Federico D. Sacerdoti, Mason J. Katz, Matthew L. Massie and David E. Culler: *Wide Area Cluster Monitoring with Ganglia*. IN: *IEEE International Conference on Cluster Computing, Proceedings*, 2003:pp. 289–298. (<http://ganglia.info/papers/Sacerdoti03Monitoring.pdf>).
URL <http://dx.doi.org/10.1109/CLUSTER.2003.1253327> [p. 33]
- [Slo05] Joseph D. Sloan: *High Performance Linux Clusters with OSCAR, Rocks, openMosix, and MPI*. O'Reilly, Sebastopol, CA, 2005. ISBN 0-596-00570-9. [p. 1, 2]
- [Ste02] Thomas Sterling, editor: *Beowulf Cluster Computing with Linux*. The Massachusetts Institute of Technology Press, 2002. ISBN 0-262-69274-0. [p. 1]
- [Tan03] Andrew S. Tanenbaum: *Computer Networks, Fourth Edition*. Prentice Hall, Upper Saddle River, NJ, 2003. ISBN 978-0-13-066102-9. [p. 4]
- [top00] *The TOP500 list of the 500 most powerful commercially available computer systems*. website, November 2000.
URL <http://www.top500.org/list/2000/11/200/> [p. 23]
- [Wor05] Stefan Worm: *Administration of Access Rights in Web Applications*. Student Research Paper, Technical University of Chemnitz, Faculty of Computer Science, Germany, 2005.
URL <http://archiv.tu-chemnitz.de/pub/2005/0143/> [p. 4]

Index

Symbols

α error, *see* false positive event classification

β error, *see* false negative event classification

A

Abinit, 23, 47

 abinip, 47, 52

 abinis, 52

 DFT, 47

 MPI, 47

 MVAPICH2, 47, 54

 OpenIB, 47

absolute deviation, *see* measurement

access point (AP), 11

accounting management, 4

active monitoring checks, *see* types of monitoring

add-on, *see* Nagios add-on

administration policy, 26

administrator, 1, 10, 15, 16

Advanced Encryption Standard (AES),
 see encryption algorithm

Advanced Micro Devices, *Inc.* (AMD),
 23

agent, *see* Nagios Remote Plugins Ex-
 ecutor (NRPE)

air conditioning, 28, 31

alert, *see* monitoring communication

AMD CPU

 Athlon, 61

 Opteron, 23

Angel Network Monitoring, *see* moni-
 toring software (rejected)

application management, *see* integrated
 management

Application Programming Interface (API),
 34

arithmetic mean, *see* measurement

Association for Computing Machinery
 (ACM), 83

B

bandwidth, *see* network bandwidth

Basic Linear Algebra Subroutines (BLAS),
 23

batch system, 9, 27, 35

benchmark

 netgauge, 54

Beowulf, 23

Big Brother, *see* monitoring software

Big Sister, *see* monitoring software

bug, 18

C

capacity utilisation, 35

Car-Parrinello Molecular Dynamics (CPMD),
 24

cellular phone text message, 20

chassis, 11, 15

check_iberr script source code, 67

check_iberr.pl, 42, 56, 63, 73

Chemnitz High-Performance Linux Clus-
 ter (CHiC), 23

Chemnitz Linux Cluster (CLiC), 23

Chemnitz University of Technology¹, 23

CHiC components

12x visualisation nodes, 23

2x login nodes, 23

2x management nodes, 23

530x compute nodes, 23

8x I/O nodes, 23

Clos network, 23

CluMon, *see* monitoring software (rejected)

cluster

capability, 3, 5

capacity, 3, 5, 23

failover, 2

Fault-Tolerant, 2

High-Availability (HA), 2

High-Performance (HP), 2, 23, 51

High-Throughput, 2

Load-Balancing (LB), 2

commercial monitoring software

OpenView, *see* HP OpenView

Tivoli, *see* IBM Tivoli

community, *see* Nagios

computer error, *see* bug

computing centre, 11, 32, 39

configuration management, 4

CPU, 11, 13, 15, 19, 51

cron, *see* Unix/Linux programs

D

daemon, 10, 38

Data Encryption Algorithm (DEA), *see* encryption algorithm

Data Encryption Standard (DES), *see* encryption algorithm

DataBase (DB), 11

defective temperature sensor, 16

Density Functional Theory (DFT), *see* Abinit

depreciation (of the cluster), 36

df, *see* Unix/Linux programs

DHCP (Dynamic Host Configuration Protocol), 59

Direct Memory Access (DMA), 39

diskless, 23

dissemination of monitoring data, 19, 22

distributed application, 13

Distributed Processing System (DPS), 13

Domain Name Service (DNS), 11

Double Data Rate (DDR), *see* InfiniBand (IB) network

double-precision General Matrix Multiply (DGEMM), 23

downtime (of an application), 2

downtime (of the cluster), 36

duplex network connection, 39

E

e-mail, 21

e-mail client, 26

encryption algorithm

AES (Rijndael), 38

DES (DEA), 38

Serpent, 38

TDES (TDEA), 38

Twofish, 38

enterprise management, *see* integrated management

error counters, *see* InfiniBand port counters

error log file, 11

escalation, 24

horizontal, 21

vertical, 21

escalation procedure, 20

Ethernet

Fast, 23

Gigabit, 23

event classes

alright/okay, 15

critical, 15

unknown, 15

warning, 15

event classification

true negative, 18

true positive, 18

event classification error

¹<http://www.tu-chemnitz.de/>

false negative, 18
 false positive, 18
 event log file, 11
 event monitoring, *see* real-time monitoring
 ExcessiveBufferOverrunErrors, *see* InfiniBand port counters
 eXtensible Markup Language (XML), 33

F

false negative event classification, 18
 false positive event classification, 18
 Fast Ethernet, 23
 fault management, 4, 22
 fault monitoring, *see* real-time monitoring
 Fault, Configuration, Accounting, Performance and Security (FCAPS), 4
 Fault-Tolerant Cluster, 2
 File Transfer Protocol (FTP), 11
 fixed disk, *see* Hard Disc Drive (HDD)
 full-duplex, *see* duplex network connection

G

Ganglia, *see* monitoring software
 gateway, *see* Nagios Remote Plugins Executor (NRPE)
 generation of monitoring data, 10, 22
 GFlops, 23
 Gigabit Ethernet (GbE), 23, 47
 Global Unique Identifier (GUID), *see* InfiniBand (IB) network
 Globus Toolkit
 SweGrid Accounting System (SGAS), 35
 Globus Toolkit², 35
 GNU, xii
 GNU General Public License (GPL), *see* open source software
 Graphical User Interface (GUI), 21

²<http://www.globus.org/>

graphics card, 11, 23
 Graphics Processing Unit (GPU), 11
 grid software, *see* Globus Toolkit
 GroundWork, *see* monitoring software (rejected)
 groupware system, 25

H

Hard Disc Drive (HDD), 11
 bad block test, 29
 hardware, 11, 12
 health counters, *see* InfiniBand port counters
 heating, ventilation and air conditioning (HVAC), *see* air conditioning
 High-Availability Cluster, 2
 High-Performance Cluster (HPC), 2, 23, 51
 high-speed interconnect, *see* InfiniBand (IB) network
 High-Throughput Cluster, 2
 historical monitoring, 8, 19, 21, 22
 horizontal escalation, 21
 Host Channel Adapter (HCA), *see* InfiniBand (IB) network
 HP OpenView, 31
 Network Node Manager (NNM), 31
 HP³ (Hewlett-Packard), 31
 HW, *see* hardware
 HyperText Transfer Protocol (HTTP), 11

I

I/O nodes, *see* CHiC components
 I/O server, 18, 23, 28
 IBM Tivoli, 31
 NetView, 31
 IBM⁴ (International Business Machines), 23
 ifconfig, *see* Unix/Linux programs
 incoming traffic, *see* network traffic
 Indiana University, Bloomington⁵, i

³<http://www.hp.com/>

⁴<http://www.ibm.com/>

⁵<http://www.indiana.edu/>

- InfiniBand (IB) network, 23, 47
 - Double Data Rate (DDR), 39
 - Global Unique Identifier (GUID), 42
 - high-speed interconnect, 39
 - Host Channel Adapter (HCA), 39, 57
 - ib_mad1, 49
 - inter-server communication, 39
 - Local Identifier (LID), 42
 - Management Datagram (MAD), 49
 - processing queue, 56
 - Remote Direct Memory Access (RDMA), 39
 - server-I/O communication, 39
 - Single Data Rate (SDR), 23, 39, 47
 - subnet manager (SM), 40
 - InfiniBand Architecture (IBA), 39
 - InfiniBand network benchmark, *see* benchmark
 - InfiniBand port counters
 - ExcessiveBufferOverrunErrors, 39
 - LinkDowned, 39
 - LinkErrorRecovery, 39
 - LinkIntegrityErrors, 39
 - RcvConstraintErrors, 39
 - RcvErrors, 39
 - RcvRemotePhysErrors, 39
 - RcvSwRelayErrors, 39
 - SymbolError, 39
 - VL15Dropped, 39
 - XmtConstraintErrors, 39
 - XmtDiscards, 39
 - InfiniBand Trade Association (IBTA), 39, 81
 - InfiniBand transmission counters
 - PortRcvData, 39
 - PortRcvPkts, 39
 - PortXmitData, 39
 - PortXmitPkts, 39
 - information management, *see* integrated management
 - inodes, 27
 - Input/Output (I/O), 39
 - instant messages (IM), 20
 - Institute of Electrical & Electronics Engineers (IEEE), 83
 - integrated management, 3, 5
 - application management, 3
 - enterprise management, 3
 - information management, 3
 - network management, 3
 - service management, 3
 - systems management, 3
 - Intel CPU
 - Pentium III, 23
 - Xeon, 47
 - Intelligent Platform Management Interface (IPMI), 23, 24
 - Inter-Process Communication (IPC), 53
 - inter-server communication, *see* InfiniBand (IB) network
 - interconnection network, 2, 23
 - International Electronic Commission (IEC), 4
 - International Organization of Standardization (ISO)⁶, 4
 - International Parallel & Distributed Processing Symposium (IPDPS), 81
 - Internet Control Message Protocol (ICMP), 11
 - Internet Message Access Protocol (IMAP), 11
 - Internet Protocol over InfiniBand (IPoIB), 50
 - interpreter, *see* Perl interpreter
 - iptraf, *see* network monitoring, 62
 - ISO/IEC 7498-1, 4
 - ISO/IEC 7498-4, 4
- J**
- Java, *see* programming language
- L**
- latency, *see* network latency
 - Lemon, *see* monitoring software (rejected)
 - library, *see* Nagios

⁶<http://www.iso.org/>

- limitations of root permissions
 - SELinux, 42
 - setuid, 42
 - sudo, 42
- LinkDowned, *see* InfiniBand port counters
- LinkErrorRecovery, *see* InfiniBand port counters
- LinkIntegrityErrors, *see* InfiniBand port counters
- Linux, 23, 24, 27, 47, 61
- Linux distribution
 - Scientific Linux, 47, 61
- lm_sensors, 24, 51
- Load-Balancing Cluster, 2, 14
- Local Area Network (LAN), xiii
- Local Identifier (LID), *see* InfiniBand (IB) network
- local monitoring, 10, 22
- log file
 - error, 11
 - event, 11
- login, *see* Unix/Linux programs
- login nodes, *see* CHiC components
- login server, 23, 28
- M**
- Mail User Agent (MUA), *see* e-mail client
- mainboard, 11
- management, 3, 16, 22
- management (ISO/OSI)
 - accounting management, 4
 - configuration management, 4
 - fault management, 4
 - performance management, 4
 - security management, 4
- MANagement Datagram (MAD), *see* InfiniBand (IB) network
- management network, 28
- management nodes, *see* CHiC components
- measurement
 - absolute deviation, 54
 - arithmetic mean, 54
 - median, 54
 - outliers, 54
- Megware⁷, 23
- memory, *see* RAM
- Message Passing Interface (MPI), 24, 47, 54
- Midas, *see* monitoring software (rejected)
- Mon, *see* monitoring software (rejected)
- monitoring, 7, 22
 - event, 7
 - fault, 7
 - historical, 8, 19
 - performance, 8
 - real-time, 7, 19
 - to monitor, 7
- monitoring communication
 - alert, 10
 - polling, 10
 - probe, 10
 - pulling, 10
 - pushing, 10
 - trap, 10
- monitoring model, 9, 22
 - Dissemination, 9
 - Generation, 9
 - Presentation, 9
 - Processing, 9
- monitoring policy, 10
- monitoring reports, 19
- monitoring software
 - Big Brother, 24, 32
 - Big Sister, 32
 - Ganglia, 33
 - Nagios, 34
 - OpenNMS, 33
- monitoring software (rejected)
 - Angel Network Monitoring, 35
 - CluMon, 35
 - GroundWork, 35
 - Lemon, 35
 - Midas, 35
 - Mon, 35

⁷<http://www.megware.de/>

- Munin, 35
- Performance Co-Pilot, 35
- PIKT, 35
- Spong, 35
- Supermon, 35
- Zenoss, 35
- monitoring suite, *see* monitoring system
- monitoring system, iv, 8, 10, 11, 13, 15, 18, 20, 22, 26, 29
- Multi Router Traffic Grapher (MRTG)⁸, 34
- multicast, 33
- Munin, *see* monitoring software (rejected)
- MVAPICH2, *see* Abinit

N

- Nagios, 34
 - community, 41
 - Perl plugin library, 41
 - plugin wrapper, 41
 - tutorials, 41
- Nagios add-on
 - Nagios Remote Plugins Executor (NRPE), 48, 60
 - Nagios Service Check Acceptor (NSCA), 38, 61
- Nagios plugins
 - check_load, 51
 - check_log, 51
 - check_mem, 51
 - check_ntp, 51
 - check_ping, 51
 - check_procs, 51
 - check_sensors, 51
 - check_ssh, 51
 - check_tcp, 51
- Nagios status codes
 - Critical, 41
 - OK, 41
 - Unknown, 41
 - Warning, 41
- netgauge, 54

⁸<http://oss.oetiker.ch/mrtg/>

- NetView, *see* IBM Tivoli
- network, *see* interconnection network
- network bandwidth, 39
- network benchmark, *see* benchmark
 - netgauge, 54
- network connection, 16
- network controller, 11
- network driver
 - GbE, 53
 - IPoIB, 53
- Network File System (NFS), 11
- Network Interface Card (NIC), 57
- network latency, 39
- network management, *see* integrated management
- Network Management Software (NMS), 33
- network monitoring
 - iptraf⁹, 59
- Network Node Manager (NNM), *see* HP OpenView
- network switch, 11, 16, 28
- network throughput, 54
- Network Time Protocol (NTP), 26, 51
- network traffic, 11
 - in, 59
 - out, 59
- NSCA communication encryption, *see* encryption algorithm

O

- open source software
 - Big Sister, 32
 - Nagios, 34
 - OpenNMS, 33
- Open Systems Interconnection (OSI), 4
- OpenFabrics Alliance¹⁰, 42
- OpenFabrics Enterprise Distribution (OFED), 42
- OpenIB, 47
- OpenNMS, *see* monitoring software
- OpenView, *see* HP OpenView

⁹<http://iptraf.seul.org/>

¹⁰<http://www.openfabrics.org/>

- Operating System (OS), 11, 24
- Opteron CPU, 23
- OSI Management functional areas, *see* management (ISO/OSI)
- Out-Of-Band (OOB), 48
- outgoing traffic, *see* network traffic
- P**
- Paradyn, 36
- passive monitoring checks, *see* types of monitoring
- PCI Express (PCIe/PCI-E), 39
- PCI eXtended (PCI-X), 39
- Pentium III, *see* Intel CPU
- performance, 11, 22
- Performance Co-Pilot, *see* monitoring software (rejected)
- performance counters, *see* InfiniBand port counters
- performance management, 4, 22
- performance measurement tool, *see* Paradyn
- Performance Modelling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS), 81
- performance monitoring, *see* historical monitoring
- Peripheral Component Interconnect (PCI), 39
- Perl, *see* programming language
- Perl interpreter, 41
- PIKT, *see* monitoring software (rejected)
- ping, *see* Unix/Linux programs
- plugins, *see* Nagios plugins
- policy
 - administration, 26
 - monitoring, 10
- polling, *see* monitoring communication
- Portable Batch System (PBS), 27
- PortRcvData, *see* InfiniBand transmission counters
- PortRcvPkts, *see* InfiniBand transmission counters
- PortXmitData, *see* InfiniBand transmission counters
- PortXmitPkts, *see* InfiniBand transmission counters
- Post Office Protocol (POP), 11
- Power Supply Unit (PSU), 11
- presentation of monitoring data, 22
- presentation of monitoring results, 21, 22
- proactive management, 16
- proactive system management, 8
- probe, *see* monitoring communication
- processing of monitoring data, 14, 22
- programming language
 - Java, 33
 - Perl¹¹, 41
- ps, *see* Unix/Linux programs
- pseudo-encryption
 - XOR, 38
- pull information, 20, 26
- pulling, *see* monitoring communication
- push information, 20, 26
- pushing, *see* monitoring communication
- Q**
- queue, *see* InfiniBand (IB) network
- R**
- rack, 11, 15, 28
- RAM, 11, 47, 61
- RcvConstraintErrors, *see* InfiniBand port counters
- RcvErrors, *see* InfiniBand port counters
- RcvRemotePhysErrors, *see* InfiniBand port counters
- RcvSwRelayErrors, *see* InfiniBand port counters
- reactive system management, 8
- real-time monitoring, 7, 19, 22
- Redundant Array of Independent Disks (RAID), 28
- Remote Direct Memory Access (RDMA), *see* InfiniBand (IB) network

¹¹<http://www.perl.org/>

remote monitoring, 10, 22

 active, 10

 passive, 10

return code, *see* Nagios status codes

Rijndael, *see* encryption algorithm

Round Trip Time (RTT), 51, 54

router, 31

S

scalability, 13, 22

Scientific Linux, *see* Linux distribution

security level, 38

security management, 4

Self-Monitoring, Analysis, and Reporting Technology (S.M.A.R.T.), 11

SELinux, *see* limitations of root permissions

sensor, *see* temperature sensor

Serpent, *see* encryption algorithm

service, *see* daemon

service management, *see* integrated management

setuid, *see* limitations of root permissions

Simple Mail Transfer Protocol (SMTP), 11

Simple Network Management Protocol (SNMP), 11, 31

Single Data Rate (SDR), *see* InfiniBand (IB) network

SMS, 20, 24

software, 11

software licences, 11

Spanish Initiative for Electronic Simulations with Thousands of Atoms (SIESTA)¹², 24

Spong, *see* monitoring software (rejected)

SSH, 10, 51, 59

status code, *see* Nagios status codes

storage system, 18, 23, 28

subnet manager (SM), *see* InfiniBand (IB) network

sudo, *see* limitations of root permissions

supercomputer, 2

Supermon, *see* monitoring software (rejected)

support, 32

SweGrid Accounting System (SGAS), *see* Globus Toolkit

switch, *see* network switch

SymbolError, *see* InfiniBand port counters

symmetric duplex connection, *see* duplex network connection

Symmetric Multi-Processor (SMP), 23

system administrator, *see* administrator

systems management, *see* integrated management

T

tag cloud, *see* weighted list21

TDEA (Triple DEA), *see* encryption algorithm

TDES (Triple DES), *see* encryption algorithm

temperature, 11

 chassis, 11, 15

 computing centre, 11

 CPU, 11, 15, 51

 GPU, 11

 HDD, 11

 mainboard, 11

 rack, 11, 15

temperature sensor, 16

throughput, *see* network throughput

Tivoli, *see* IBM Tivoli

top, *see* Unix/Linux programs

Top500 list, 1, 23

Transmission Control Protocol (TCP), 51

trap, *see* monitoring communication

trend monitoring, *see* historical monitoring

true negative event classification, 18

¹²<http://www.uam.es/departamentos/ciencias/fismateriac/siesta/>

true positive event classification, 18
 tutorials, *see* Nagios
 Twofish, *see* encryption algorithm
 type II error, *see* false negative event classification
 type I error, *see* false positive event classification
 types of monitoring
 active, 37
 passive, 38

U

Unix/Linux programs
 cron, 10, 24
 df, 10
 ifconfig, 10
 login, 10
 ping, 10, 27, 51
 ps, 10, 59
 ssh, 10, 51
 top, 10, 59, 63
 usability, 21, 22, 44
 User Datagram Protocol (UDP) based services
 NTP, 26

V

vendor support, *see* support
 vertical escalation, 21
 visualisation nodes, *see* CHiC components
 VL15Dropped, *see* InfiniBand port counters
 Voltaire¹³
 HCA, 40
 switch, 40

W

web server, 51
 webmail, *see* e-mail client
 weighted list, 21
 wireless local area network (WLAN), 11
 wrapper, *see* Nagios

¹³<http://www.voltaire.com/>

X

Xeon, *see* Intel CPU
 Xiranet¹⁴ storage system, *see* storage system
 XmtConstraintErrors, *see* InfiniBand port counters
 XmtDiscards, *see* InfiniBand port counters
 XOR, *see* pseudo-encryption

Z

Zenoss, *see* monitoring software (rejected)
 zero Nagios plugin execution delay, 54

¹⁴<http://www.xiranet.com/>

Acknowledgements

Thank you for inspiring me, helping me, showing me the pro and contra regarding my decisions, for the substantial discussions and the possibility of sharing their knowledge with me. In short, everyone who was involved in the creation process of this work.

In alphabetic order:

Matthias Clauß
Detlef Heine
Torsten Hoefler
Gerd Kretzschmar
Torsten Mehlan
Frank Mietke
Thomas Müller
Andreas Poller
Wolfgang Rehm
Wolfgang Riedel
Thomas Schier
Ronald Schmidt
Tom Schwaller
Regina Trieder
Jens Wegener