IPMI-based Efficient Notification Framework for Large Scale Cluster Computing

Chokchai Leangsuksun[1], Tirumala Rao[1], Anand Tikotekar
Stephen L. Scott[2], Richard Libby[3], Jeffrey S. Vetter[2] , Yung-Chin Fang[4], Hong Ong[2]
[1]*Computer Science Department, Louisiana Tech University*
Ruston, LA 71272, USA
[2]*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
Oak Ridge, TN 37831, USA
[3]*Enterprise Platforms Group, Intel Corporation,*
[4]*Dell, Inc*
[1]*{box, trb013,aat06}@latech.edu* [2]*{scottsl,vetter,hongong}@ornl.gov*
[3]*rml@hpc.intel.com, Yung-Chin_Fang@Dell.com*

Abstract

The demand for an efficient fault tolerance system has led to the development of complex monitoring infrastructure, which in turn has created an overwhelming task of data and event management. The increasing level of details at the hardware and software layer clearly affects the scalability and performance of monitoring and management tools. In this paper, we propose a problem notification framework that directly addresses the issue of monitor scalability. We first present the design and implementation of our step-by-step approach to analyzing, filtering, and classifying the plethora of node statistics. Then, we present experimental results to show that our approach only needs minimal system resource and thus has low overhead. Finally, we introduce our web-based cluster management system that provides hardware controls at both cluster and nodal levels.
Key words: *Scalability, High-Availability, IPMI.*

1. Introduction

Cluster monitoring software is a powerful tool for overseeing resources usage and performance assessment of computing nodes. However, for very large-scale systems, the massive amount of communication and collected data has caused serious concerns for scalability and manageability.

Most developers considered this problem as design issues and thus reengineered their monitoring architecture to handle communication among the increasing number of nodes. The advent of intelligent hardware technologies (e.g. IPMI [12]) for high-end server and cluster infrastructure strive for alleviating system manageability while resulting in the increasing level of hardware detail from cluster nodes.

In this paper, we propose a framework that aims to provide a dynamic monitoring and filtering approach that can reduce the number of nodal information exchanges among individual system and throughout the entire cluster. It is based on user-defined policies for gathering hardware details to support dynamic monitoring and event notification. Furthermore, it analyses the sensor information using the exhibited characteristics and then classifies the information into predefined categories. The classified information is then processed in order to potentially reason anomalous behavior. The proof-of-concept result suggests that our approach can mitigate the scalability issues encountered due to the plethora of information from cluster hardware standards and sheer system size.

[1] Research supported by the Department of Energy Grant no: DE-FG02-05ER25659.
[2] Research supported by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, Office of Science, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

2. Related Work

Beowulf distributions like OSCAR [1][2], ROCKS[3], and SCE[4] are integrated with the capabilities to monitor and manage HPC clusters. Existing monitoring systems, such as ganglia [5], clumon [6], supermon[7], ClusterProbe [8], PerfMC [9] and PARMON [10] possess similar architecture with a central monitoring server and local daemons collecting node information.

However, there are growing concerns caused by overwhelming complexity of monitoring a very scale cluster nodes with increasing amount of data available from individual nodes. Table 1 provides a comparison of the existing monitoring tools based on characteristics of cluster environment.

Table 1. Comparison of cluster monitoring tools

System	Capability	Scalable	Completeness
Ganglia	Monitoring	Medium	Limited
Clumon	Monitoring	Scalable within clusters	400 metrics
Dproc	Monitoring	High	Extensible
Supermon	Monitoring	High	Limited
Big-brother [11]	Monitoring/ Alerting	Low	-
ClusterProbe	Monitoring and management	Yes	-
Ka-admin	Monitoring	Yes	-

The existing cluster monitoring tools face the scalability and performance issues when increasing number of nodes in the systems. In addition, the current alerting mechanism may not be effective as its naïve approach based on fixed thresholds. We introduce a framework that provides a step-by-step approach to gather hardware health characteristics. We also adopt dynamic polling rates based on the current analysis of the cluster and node statistics. The framework uses certain intelligence of hardware parameters apart from the threshold settings in reporting the watchful events. Furthermore, it also exploits the characteristics in tweaking the polling rates for individual sensors.

3. IPMI Background

Intelligent Platform Management Interface (IPMI) [12][13][14] is one such intelligent hardware technology available in most of the current generation of servers across vendors. Its adoption rate has increasingly advanced into cluster environments. IPMI supports both in-band and out-of-band management and takes lead over other industry specifications.

IPMI is an abstraction layer above hardware management interfaces. It defines common commands, data structures, and message formats to monitor server physical health characteristics, such as temperature, voltage and so on. The goal of IPMI is to provide a foundation of interoperability for remote system monitoring/management across different hardware implementation. Figure 1 shows the IPMI management subsystem block diagram.

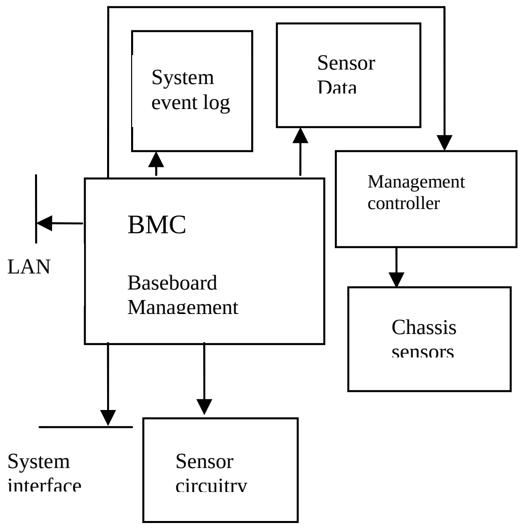


Figure 1 Management subsystem Block diagram

IPMI hardware consists of a microcontroller called Baseboard Management Controller (BMC). The user, application, or any intelligent device can send a request/command to BMC. BMC processes such requests without any additional load on the system CPU. There could be several Management Controllers (MC) that monitor different parts of the subsystem. The sensors can generate critical events asynchronously and report them to the BMC.

In addition, IPMI hardware consists of non-volatile memory called System Event Log (SEL) and Sensor Data Repository (SDR) to store critical events and sensor information respectively. SEL and SDR records can be fetched sequentially from the non-volatile memory by sending IPMI requests to the BMC.

4. Architecture

This section presents the design issues, architecture, and implementation of our framework. It consists of three modules: Data Gathering and Analysis Engine

(DGAE), Policy-Based Monitoring (PBM) and Problem Notification Agents (PNA).

The DGAE is the basic monitoring daemon, which gathers multiple samples of sensor readings, computes certain characteristics and classifies the information into predefined categories.

The PBM agent works with data gathering engine in providing the dynamic monitoring capability. The PNA as the top layer of our framework receives the exhibited behavior of hardware sensors from the analysis engine after determining the cause of such activity.

The framework is capable of providing notifications of unusual hardware behavior that could lead to hardware failure. Figure 2 illustrates the architecture layered over operating system, and hardware management technologies as the foundation.

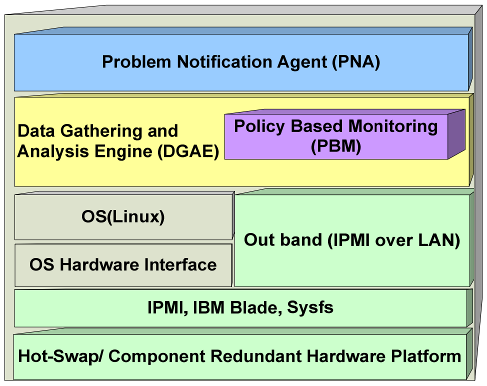

Problem Notification Agent (PNA)
Data Gathering and Analysis Engine (DGAE)
Policy Based Monitoring (PBM)
OS(Linux)
OS Hardware Interface
Out band (IPMI over LAN)
IPMI, IBM Blade, Sysfs
Hot-Swap/ Component Redundant Hardware Platform

Figure 2. Our Framework Architecture

4.1. Data Gathering and Analysis Engine

DGAE provides an interface to hardware management and capability to gather hardware sensor information. It analyzes the information upon consultation with the policy module (PBM). The collecting mechanism is thereby influenced by the policy module. This mechanism enables the DGAE to adapt according to the existing environment. DGAE is involved in the following operations:

- Polling hardware sensors at regulated intervals.
- Analyze the hardware state statistics to uncover unusual characteristics and determine polling intervals
- Log/Retrieve the OS/application state into hardware event logs
- Validate the hardware events

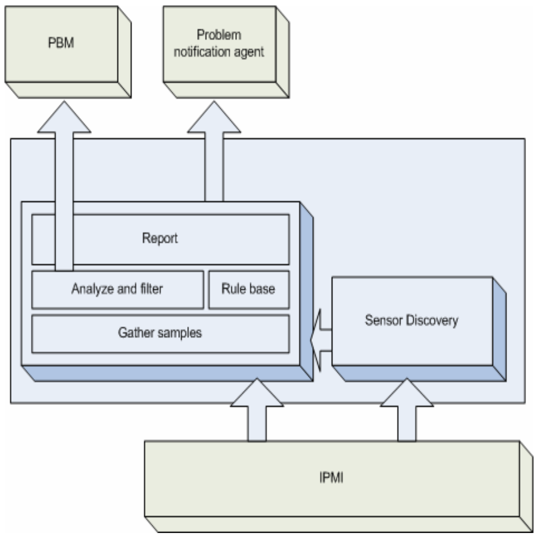


Figure 3 DGAE block diagram

4.2. DGAE

Figure 3 illustrates the DGAE block diagram and the interface to other modules. DGAE invokes the sensor discovery to gather information about existing sensors via OpenIPMI library. The sensor information is categorized based on the sensor device. The device components are represented by an n-tuple $D = (C_1, C_2, C_3 \dots C_n)$ where each component is associated with set of sensors represented by k-tuple $C_i = (s_1, s_2, s_3 \dots s_k)$. For instance, processor component monitored by the temperature, core voltage and booster sensor are represented by the tuple (p_{temp}, p_{volt}, p_{bst}, p_{stat}) and the respective samples are:

```
(31.000000, 31.000000, 1.474200, 6000.00),
(30.000000, 30.000000, 1.474200, 5880.00),
(31.000000, 31.000000, 1.474200, 6000.00),
(31.000000, 30.000000, 1.474200, 6000.00)
```

DGAE collects samples of sensor information and then analyzes certain characteristics like the rate of change, frequent transition, and threshold etc. The analysis triggers the appropriate policy for determining the next monitoring intervals. The metrics that were found to be constant in the sample space are provided longer monitoring intervals than those exhibit major changes.

Based on the observed behavior of sensor samples, these policies could trigger some actions defined in the rule base. The result of such actions can reduce the number of events reported to the problem notification agent. The rule specification is defined as follows:

$$C^1_{temp} \wedge C^2_{temp} \wedge C^3_{temp} \dots \wedge C^T_{temp} \Rightarrow S_{temp}$$

It specifies that if multiple components or devices C^i exhibit abnormal temperature then the DGAE should report the problem. In this example, the system temperature event will be notified.

$$C^1_{fan} \wedge C^2_{fan} \wedge C^3_{fan} \ldots \wedge C^T_{fan} \Rightarrow S_{fan}$$

Similarly, if the analysis engine detects that the processor fan, baseboard fan and other component fans exhibit unusual behavior, it reports the system cooling has a problem.

$$B_{5v} \vee B_{12v} \vee \left(B_{volt1} \ldots \wedge B_{voltT}\right) \Rightarrow S_{volt}$$

The above rule states that the baseboard voltage levels are either the standard or converted voltages exhibiting similar behavior that they are reported as the system voltage problem.

The basic algorithm used by DGAE in reporting critical issues is presented in Figure 4. DGAE continues to poll for hardware sensor information under the direction of the given policies. The critical events are reported to the DGAE asynchronously like redundancy lost, fan failure, device removed. The critical alert may be prone to errors and should undergo the validation process.

```
Input Sample set of component readings
Output  undesirable behavior of the component
AnalyzeFilter( S, rulebase )
01  for all Ci ∈ S
02  for all sensors ∈ Ci
03  aggregate(sensor)
    /* aggregate the collection of sensor
    samples provided by the gathering
    module to compute the rate of
    increase in the reading, frequent
    transition around the nominal value
    and check against the threshold
    levels*/
04  compute(rateofchange, freqtransition,
            checkthreshlevel)
05  end for
06  check(rulebase)
07  classify(sensor)
08  if(undesirable)
    /* If the computations reveal the
    undesirable behavior of the component
    sensor, it is checked against the rule
    base to determine the problem*/
09  report(problem)
10  end if
```

Figure 4. Algorithm for analyze and filter process

The critical events are categorized into the following severity levels; *nominal, non-critical, critical, non-operational* and *failure*.

Table 2 Classification of the events reported by DGAE

Events	Event class
Fully Redundant Nominal sensor readings	Nominal
Fan Redundancy Degrade Memory Parity	Out of nominal range
Power Redundancy Lost Fan Redundancy Lost ECC correctable	non critical range
Processor Intrusion AC lost Critical thresholds	critical range
FRB hang Processor disabled	Non operational range
Processor failure Thermal trip Memory Scrub failure Memory failure ECC uncorrectable	Failure range

Policy based monitor (PBM)

Policy-based monitor (PBM) defines set of policies aiming to reduce overhead while collecting and processing data from the hardware subsystem. PBM also regulates the sensor threshold in the passive monitoring by setting policies. The policy throttles the threshold value to get the prior notification of the increasing sensor value. However, the approach of periodically setting the threshold may not be very efficient.

We resolve the inefficiency with a novel mechanism by making use of the platform event filtering with which will set the thresholds in an intelligent manner. We also make use of system event log for the historical events and correlate them in such a manner that if some of these events materialize then an alert will be fired through the event filter tables. This alert will serve as an indication that one should change thresholds for some sensors to receive any prior notification. For example, considering dependency between different temperature sensors, it is possible that Processor1 and ambience temperature have exceeded the thresholds (which may have been set already) and then the events are fired. We will get an alert through the platform event filtering mechanism signifying that these two events have materialized. We may infer the fact that the processor2's temperature has not exceeded its threshold, which may be a cause of concern. We can then change the threshold to see whether the event is fired or not (through the IPMI alert mechanism). Finally, we may also want to change the thresholds of the two processor temperature sensors if we know that certain related or correlated events (such as ambience temperature increase). It is also possible that the threshold values may change through wear and tear or through the lifetime of the hardware component. Therefore, it is important to set the thresholds so that events would be logged. The thresholds could also be set using a relative frequency approach. This approach observes the relative frequency of a particular event in the system event log using different thresholds.

Figure 5 shows our experimental and benchmark results of the resource load encountered in active and passive monitoring.

Monitoring approach	Resource overhead
Active poll based monitoring	0.0065%CPU/s & 0.1625%MEM/s
Passive monitoring	0%CPU/s & 0.0%MEM/s

Figure 5 Comparison of two monitoring approaches

Figure 6 shows a policy example to define monitoring intervals for the next sample of sensor readings.

```
Watch senreadings
       Service dgae
       Monitor constmetrics
       Interval 3s
       Monitor lowratemetrics
       Interval 2s
       Monitor highratemetrics
       Interval 1s
       Alert policy_module
```

Figure 6 policy definition

Figure 7 provides an insight into the %CPU overhead by the adaptive data gathering and analysis engine through the policy based support. The percentile of sensor activity represents the ratio of sensors exhibiting undesirable characteristics.

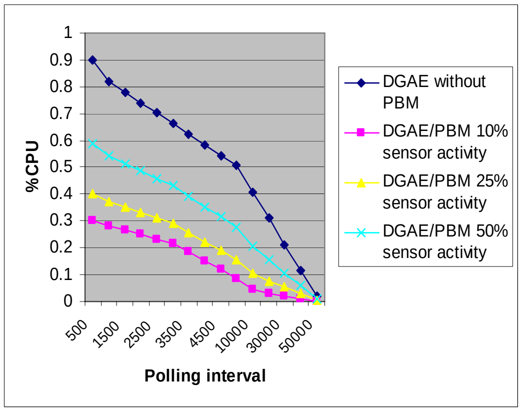
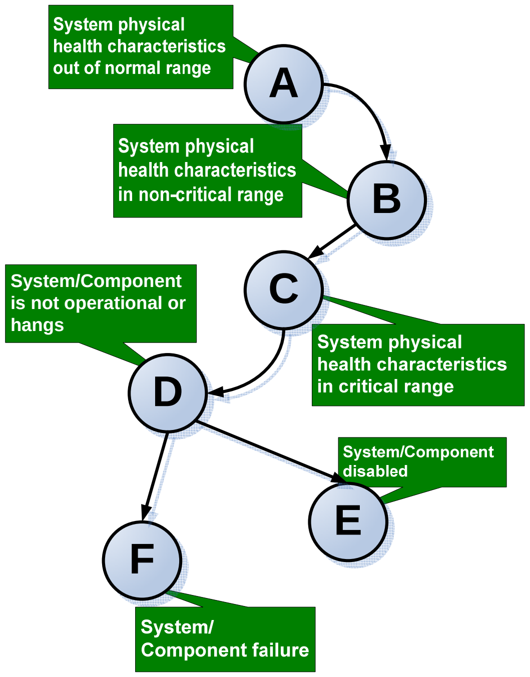


Figure 7. Analysis of policy based effects on data gathering engine

4.3. Problem Notification agent

The undesirable characteristics exhibited by the sensor are classified into the severity class and reported by the problem notification agent. The PNA receives the sequence of events from which the DGAE failure analysis model is based on the Bayesian belief network [15] to learn the system/component behavior and determine the cause of such activity.

PNA proof-of-concept was implemented with the Norsys [15] library providing the Bayesian belief network learning capability. Figure 8 exemplifies a fault propagation model for a given hardware system. The classified events reported by DGAE are mapped to the node variables mentioned in the model.



System physical health characteristics out of normal range
A
System physical health characteristics in non-critical range
B
C
System/Component is not operational or hangs
System physical health characteristics in critical range
D
System/Component disabled
E
F
System/ Component failure

Figure 8. General model to represent the propagation of component failure

5. Experiments and analysis

Figure 9 shows the benchmark comparison of existing IPMI libraries that fetch hardware sensor information. The comparison is based on the sensor retrieval time period, resource overhead and authentication capabilities. Since the retrieval time is a significant criterion in determining the best techniques, we adopt the OpenIPMI library in or DGAE to fetch sensor information. Figure 10 illustrates that DGAE based monitoring obviously provides better retrieval overhead when compared with the existing approach.

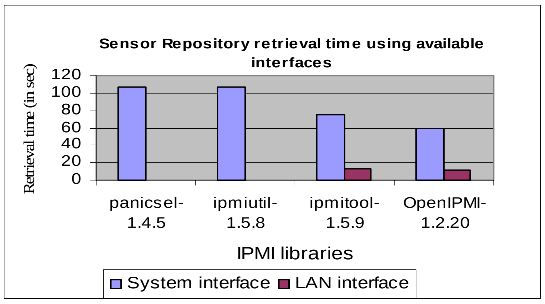


Figure 9 Comparison of the existing IPMI libraries

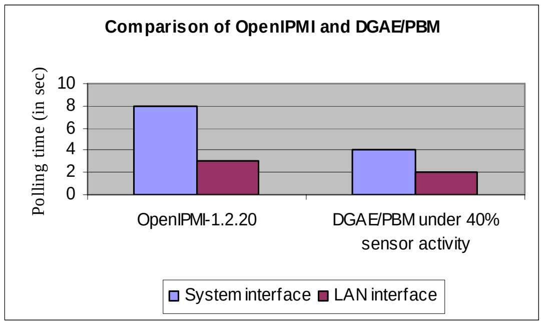


Figure 10 Comparison of DGAE/PBM and OpenIPMI sensor polling duration

The samples similar to those provided in Table 3 are processed through the analysis engine to categorize the "temperature sensor of the baseboard, hot swap and processor component" under "out of non critical range" and the fan speed under "out of nominal range". DGAE will determine the possible cause of the increasing temperature of hardware components and the unusual rise in fan speed. The experimental results from the PNA model specified in Figure 11 are provided in Table 4. The results from PNA model were collected by stimulating external activity like "Low inflow of air" and observing the hardware sensor activity.

Table 3 Sample collected by Data gathering module from the experimental node

Baseboard temp	Hot swap temperature	Processor temperature	Fan speed
37.00	25.00	44.00	8211.00
37.00	26.00	44.00	8415.00
38.00	26.00	44.00	8415.00
38.00	26.00	44.00	8211.00
37.00	26.00	44.00	8415.00
37.00	26.00	44.00	...
38.00	..	...	8415.00
...	26.00	45.00	8415.00
37.00	26.00	45.00	8415.00
37.00	27.00	45.00	8670.00
38.00	27.00	45.00	8415.00

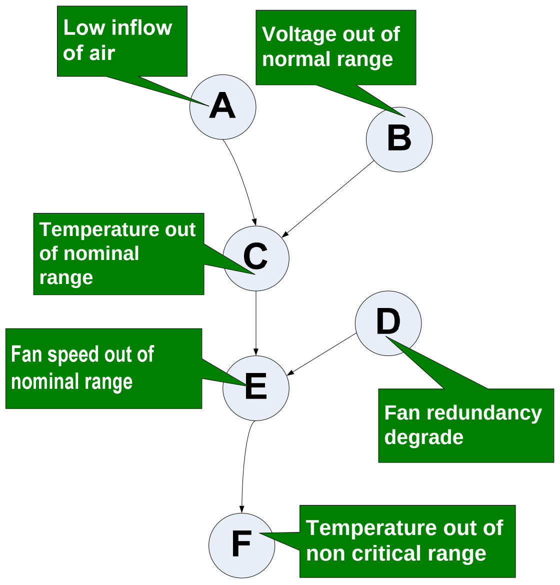

Low inflow of air
Voltage out of normal range
A
B
Temperature out of nominal range
C
D
Fan speed out of nominal range
E
Fan redundancy degrade
F
Temperature out of non critical range

Figure 11 PNA model depicting the fault propagation

Table 4. PNA output based on the observed behavior

Observed behavior	Probability of "Low inflow of air"
temperature out-of-nominal	0.893057
temperature out-of-nominal and voltage ok	0.935664
temperature out-of-nominal, voltage ok and fan speed out-of-nominal	0.991379
temperature out-of-nominal, voltage ok, fan speed out-of-nominal and temp out-of-non-critical	0.991379

6. Future work

Consider the fact that a series of events or problems may be recognized by the DGAE and forwarded to the PNA for further analysis. This strategy may overlook a possibility that multiple events logged into the SEL and the alert generated by the PNA may have a correlation. To alleviate the possible issue, we are looking into a correlation mining approach that would realize such events. The notification events from PNA could also be logged into SEL as higher-level abstraction. This way the complexity of the correlation engine is kept in check. We plan to explore Platform event filtering (PEF) to notify any higher-ordered events. The PEF is a mechanism where events are filtered by the means of event filtering configured into the platform event filter table. The alerts may be trapped using mechanisms such as SNMP. The composite events can be matched using certain filters.

This way notification will be provided for higher possible events using event correlation and PEF.

7. Conclusion

In this paper, we presented the design, implementation, and evaluation of problem notification framework for IPMI-based hardware platform. The intelligent threshold setting, analysis and monitoring features are strengths of the proposed framework. The notification mechanism has proved advantageous over existing solutions with adaptive and high configurable monitoring filtering capabilities. The experimental results suggest that our approach promises reduction in resource overhead when compared to traditional monitoring approaches. The dynamic monitoring approach with filtering mechanism provides effective notification, especially for the large scale cluster system.

8. References

[1].J Hsieh, T Leng, Y C Fang, OSCAR: A Turnkey Solution for Cluster Computing, Dell Power Solutions, Issue 1, pp. 138-140, 2001.

[2].oscar.sourceforge.net/

[3].P.M. Papadopoulos, M.J.Katz, G. Bruno, « NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters," *3rd IEEE International Conference on Cluster Computing (CLUSTER'01)* October 08 - 11, 2001

[4].P. Uthayopas, T. Sriprayoonskul, and S. Phatanapherom, "SCE: A Fully Integrated Software Tool for Beowulf Cluster System," *The Linux HPC Revolution*, June 25-27, 2001

[5].F. D. Sacerdoti, M. J. Katz, M. L. Massie, D. E. Culler, "Wide Area Cluster Monitoring with Ganglia," *IEEE International Conference on Cluster Computing (CLUSTER'03)* pp. 289

[6].NCSA Clumon. http://clumon.ncsa.uiuc.edu/doc-info.html

[7].M.J. Sottile R. G. Minnich, "Supermon: A high-speed cluster monitoring system," *IEEE International Conference on Cluster Computing (CLUSTER'02)*

[8].Z. Liang, Y. Sun, C. Wang, "ClusterProbe: An Open, Flexible and Scalable Cluster Monitoring Tool," *1st IEEE Computer Society International Workshop on Cluster Computing*, December 1999 pp. 261

[9].M. Marzolla, "A Performance Monitoring System for Large Computing Clusters," *Eleventh Euromicro Conference on Parallel, Distributed and Network-Based Processing* , February 2003. pp. 393

[10].R. Buyya. "PARMON: A Portable and Scalable Monitoring System for Clusters," *Software Practice and Experience journal*, 30(7):723–739, 2000.

[11].Big Brother: Web-based systems and network monitor www.quest.com/bigbrother/

[12].IPMI - Intelligent Platform Management Interface www.intel.com/design/servers/ipmi/

[13]. IPMI Specification v1.5 Document Revision 1.1 Published: Feb 20, 2002 www.intel.com/design/servers/ipmi/spec.htm

[14]. IPMI v2.0 specifications Document Revision 1.0 Published: Feb 12, 2004 www.intel.com/design/servers/ipmi/spec.htm

[15]. Netica, Bayesian network development software http://www.norsys.com/