A Performance Monitoring System for Large Computing Clusters

Moreno Marzolla
Dipartimento di Informatica, Università Ca' Foscari di Venezia
via Torino 155, 30172 Mestre, Italy
marzolla@dsi.unive.it

and

INFN Padova, Via Marzolo 8, 35100 Padova, Italy

Abstract

In this paper we describe the architecture of PerfMC, a Performance Monitoring system for Clusters of Workstations (CoW); a prototype implementation of the architecture is also presented. PerfMC is driven by an XML configuration file, and uses the Simple Network Management Protocol (SNMP) to collect statistics from each networked equipment. The collected data are maintained on the local disk of the monitoring station in a compact format, and various graphical and statistical analyses can be performed off-line. The monitoring tool embeds an HTTP server which is able to generate various types of graphs from the collected data. Moreover, the HTTP server can generate arbitrary XML pages by dynamically applying XSLT stylesheets to an internal XML representation of the cluster's status. The heavy use of XML-based technologies differentiates the proposed approach to traditional monitoring tools.

1. Introduction

Large clusters with hundreds or thousands of nodes are very difficult to manage due to their size and the complexity of the applications they run. An efficient monitoring system can be very helpful for continuously profiling all components of the cluster. Informations gathered from the monitor can be used both for identifying hardware malfunctions and for verifying the utilization of resources.

A *monitoring system* is a hardware or software component able to observe the activity of a system [8]. A monitor can observe the performances of a system, record statistics, analyze the data and display the results. Monitors are useful for many reasons. Programmers may use a monitor to analyze the resource usage of an application, identifying performance bottlenecks or usage patterns which may suggest better algorithms. System administrators may use a monitor to tune the system by adjusting its parameters in order to improve performances. Monitors can be used to characterize the workload of a system, or to find the parameters to be used on a simulation of a real system. Moreover, they can be used to check for hardware problems such as crashed or unresponsive hosts or broken communication links.

Continuous monitoring of computing clusters is a challenging problem for different reasons. The size (number of components) of the system to be monitored is often beyond the scalability limit of many available tools. These tools are usually tailored for a particular application; they are often closed products, so the user can't adapt them to any variation of the requirements. Moreover, they are usually hard to configure, and provide an inconvenient user interface.

In this paper we present an architecture for a scalable monitoring system for large computing clusters. A prototype tool named PerfMC (Performance Monitoring for Clusters), implementing part of this architecture will be described. PerfMC is driven by an XML [21] configuration file, and uses the Simple Network Management Protocol (SNMP) [16] to gather data from any device containing an SNMP agent. Since SNMP is a standard protocol implemented by many networked equipments, including printers, tape libraries, network switches and routers, the tool is very general and can be used to monitor a wide range of devices. PerfMC embeds a WEB server which is used to generate time-series graphs from the collected data. Also, the WEB server can produce arbitrary XML pages by applying XSLT transformations [20] to an internal XML representation of the cluster's status. In this way it is possible to produce XHTML [22] status pages which can be displayed by ordinary WEB browsers. The tool can be easily expanded by adding other types of interfaces (e.g. LDAP) by simply creating an implementation module.

The paper is organized as follows. Section 2 presents some previous works related to the problem of monitor-

ing computing clusters. In Section 3 the design goals of PerfMC are illustrated and briefly discussed. Section 4 describes the architecture and implementation of PerfMC, and a case study is presented in Section 5. Finally, Section 6 reports the conclusions and future works.

2. Related works

In recent years considerable attention has been devoted to the problem of monitoring the performances of clusters and distributed systems [2, 5, 6, 9, 11, 15, 17, 18, 19, 23]. In [18] the authors describe an agent-based monitor targeted primarily to GRID architectures [7], which are wide area distributed systems where components can be connected to high-latency WANs. The monitoring architecture is based on a producer-consumer paradigm, where individual monitors can subscribe for particular kinds of events, and receive notifications only when such events are generated by some producer.

It should be noted that a computational GRID is very different in size and complexity from a computing cluster, so implementing a monitoring system on them is different. A computational GRID is usually made of an heterogeneous collection of computing systems which are geographically distributed and connected through a WAN. A cluster is made of homogeneous machines, usually residing in the same room and connected together with a high speed LAN. These differences play an important role in defining the requirements for a monitoring system. For example, a monitor for a GRID should be built with security features, given that intrinsically insecure WAN links are used for communications. On the other hand, a cluster can be treated as a single, powerful machine. It should be protected with respect to the outside world, but communications among machines in the cluster can be unencrypted. If the monitoring infrastructure is built inside the cluster, there is no need to protect its control messages. Also, LANs are characterized by low latencies, high bandwidths and low packet loss rates. The size of a cluster is usually orders of magnitude smaller than that of a large GRID. For these reasons, monitoring systems developed for computational GRID environments, such as that described in [18], have been developed with very different requirements in mind with respect to a cluster monitor.

In [14] it is proposed a monitoring and management architecture based on the use of mobile agents written in Java. Mobile agents allow management applications to be moved to the network devices, instead of moving the data provided by the network devices to the Management Stations. The approach based on Java mobile agents obviously requires the network devices to be equipped with a Java Virtual Machine, so that they are able to accept and execute code coming from the Management Stations. This is currently only possible when the monitored elements are general-purpose computers, as other devices are generally unable to run Java code (or any other user program).

Ganglia [2] is a distributed monitoring system for clusters. It requires each node on the cluster to run a dæmon called `gmond`. It collects values from the local machine and broadcasts these values to all the other `gmond` processes running on the cluster. To limit the network utilization, broadcasts happen only when the changes in the observed values exceed a given threshold. `gmond` processes can also communicate with generic Ganglia clients by sending an XML status file over a TCP connection. Ganglia dæmons do not provide any facility to log the recorded data themselves, but rely on external programs to collect statistics, perform management actions when particular events occur, and display the status of the system.

Most monitoring tools use their own data collection protocol over TCP/IP links. One exception is SIMONE [17], which uses the standard SNMP protocol to build a large-scale, distributed monitoring system. Hierarchical monitoring has been employed in other systems as well, such as the one described in [19]. Such hierarchical, tree-based monitoring systems are particularly effective when the user is mainly interested in getting aggregate informations on the cluster's status, such as the average load of all the machines, or the least utilized node of the cluster. This is because the information can be aggregated at each intermediate node of the hierarchy, thus avoiding the possible bottleneck of a single node getting all the data from all hosts. Unfortunately, this strategy does not help when it is necessary to continuously record the values of some parameters for every single machine, for example for producing graphs showing the variation of interesting quantities over the time.

Supermon [12] is a centralized monitoring system, yet it allows efficient and frequent data collection from the nodes of a Linux cluster. The Linux kernel has to be patched for the addition of a new system call which provides status informations. A server program running on each machine collects these informations and can pass them to requesting applications using a telnet-based network protocol. A possible drawback of this approach is the necessity to use a modified kernel on the monitored machines, and the fact that it is necessary to modify the implementation of the system call if additional parameters need to be monitored.

3. PerfMC Design Goals

We identified the following requirements for our monitoring system:

Intrusion-free Monitoring the system should preserve its behavior;

Minimum overhead The monitor should have little impact on the resource utilization of the cluster;

Batch operation The monitor should be able to operate in *batch mode*, without any user supervision. At the same time, a suitable user interface should be provided;

Generality The monitor should be able to deal with a wide range of different networked devices;

Easy Configuration The monitoring tool should be easily configurable. If possible, the configuration file should be based on some standard notation, so the user is not required to learn a completely new one;

Ability to scale The monitoring tool should be as efficient as possible, and able to scale at least up to moderate cluster sizes (some hundred nodes).

Ability to monitor both hardware and software A cluster of workstations is not a mere assembly of hardware. It is used to carry out some specific computational task. The monitoring system should be able to check both the health of the hardware and the resource usage of the software running on it.

Administrators should be kept informed The monitoring system should include an alarm mechanism triggered by some condition, such as a machine not responding, or a process taking too much CPU time.

Intrusion-freeness could or could not be feasible, depending on the specific task the system being monitored performs. An intrusion-free monitor is guaranteed not to have negative impact on the *correctness* of the monitored system's results. If the monitored system strongly depends on hard real-time constraints to operate correctly, then even a small overhead induced by the monitoring activity could affect its results.

A desirable property of any monitor is that of imposing a minimal overhead on the observed system. Software monitors, however efficiently implemented, impose a non-zero overhead on the monitored system. Lower (possibly none) overhead is possible with the use of hardware monitors [8], which are special hardware devices for data collection. Hardware monitors, however, are usually expensive, special-purpose and non-portable devices.

Another important feature of a monitoring system, which is seldom considered, is ease of configuration. There is no standard configuration format used by the existing tools. We decided to use XML [21] as the language in which the configuration file is written. There exist many tools which are able to generate, verify and transform XML documents. Parsing such documents can be done efficiently, as they must obey strict syntactical rules. In Section 4.2 we will give more informations about the structure of the XML-based configuration file.

Execution profiler programs can be considered as specialized monitoring systems. Execution profiles give informations about the time spent on each program block (function or subroutine). Some profilers can also gather detailed informations about pipeline stalls incurred by individual machine-level instructions [4]. Execution profilers usually give detailed informations on single programs (the profiler described in [4] gives information on a whole system). However, we are not interested in obtaining such fine-grained measurements. Rather we want to observe both high-level performance measures (such as CPU utilization, disk I/O) as well as not strictly performance-related measures (such as CPU temperature) from each machine of a large cluster with a time resolution of the order of seconds or tens of seconds.

The ability to monitor different kind of devices is especially important. A computing cluster is usually made of a large number of computing nodes, plus other devices such as network switches, tape libraries, Uninterruptible Power Supplies (UPSes) and so on. The status of those devices needs to be monitored as well. We identified the Simple Network Management Protocol (SNMP) as a suitable candidate for the remote monitoring of a wide range of devices.

4. PerfMC Architecture and Implementation

PerfMC is a tool for medium-grained, continuous monitoring of computing clusters. A prototype implementation has been written using the C language and currently is being tested under the Linux Operating System, but should be easily portable on other flavors of Unix. It is able to monitor any networked equipment implementing an SNMP agent.

PerfMC is made of two threads: an SNMP collector and an HTTP server, as depicted in Figure 1.

Since the monitoring system is required to deal with different kinds of hardware devices, we decided to use the Simple Network Management Protocol (SNMP) for collecting status informations. The SNMP architecture has three components:

- One or more Network Management Stations (NMSs), which are responsible for monitoring and managing other devices;
- Network Nodes, which may be computing nodes or other equipments; each node hosts a software component called *SNMP-agent* which collects local data and answers requests coming from the NMSs;
- A connectionless communication service; SNMP is usually implemented on top of UDP.

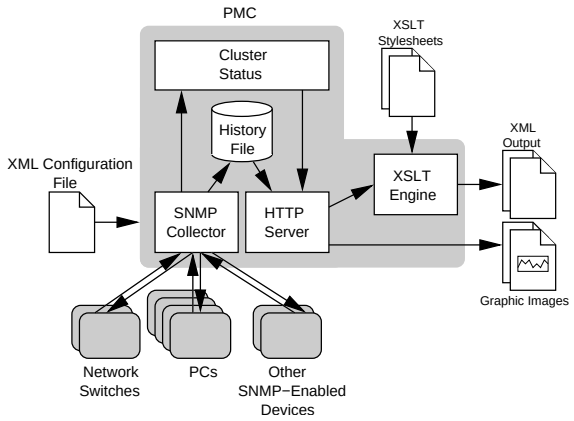

Figure 1. Schematic view of the structure of PerfMC

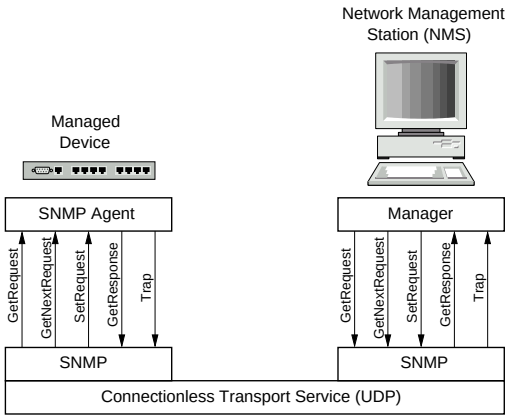

Figure 2. The SNMP Architecture

The SNMP agent running on each node manages a set of local *variables*; the management informations pertaining to a particular class of resources is defined in a Management Information Base (MIB). The NMSs access the MIB by contacting the agent using SNMP primitives such as `get` or `getnext` to read values, and `set` to update values. Figure 2 illustrates the SNMP architecture.

The use of SNMP as a data collection protocol has some drawbacks. The protocol itself is very simple, and requires each Management Station to periodically poll the other nodes. Polling could introduce additional load on the network, caused by a potentially large number of request/response packets. Also, SNMP agents have a simple structure and usually communicate only in response to `get/getnext` requests. They don't perform management actions on their own, but require a NMS to take the decisions about what should be done.

However, SNMP has the advantage of being implemented in virtually every equipment having a network interface (including workstations, network switches/routers, tape libraries, printers, Uninterruptible Power Supplies). Many vendors only include SNMP agents on their devices, and no possibility is offered to use any custom code; SNMP is thus the only way to interact with these devices. Inventing a new protocol for acquiring status informations from heterogeneous devices does not seem an appealing solution, given the diffusion of standard SNMP implementations.

Recent versions of the SNMP protocol have additional features which solve some of the problems above. In particular, SNMPv2 implements *bulk requests*, which can significantly reduce the load on the network by packing several requests into a single datagram. In this way, the Monitoring Stations can obtain the values of different MIB variables of the same host by using a single `get` request. The responses will be contained in a single packet as well.

PerfMC is being tested on a cluster composed of about 170 dual-processor Linux/Intel-based computing nodes, running the freely available SNMP agent developed by the NET-SNMP project [3]. The agent can be very easily extended; in the configuration file it is possible to associate to a given MIB variable (or a whole MIB subtree) an arbitrary executable program which will provide the required values. Our experience shows that the overhead put on the network by the SNMP request/response packets is very low. More details will be given in Section 5.

4.1. The Collector

The SNMP collector thread is responsible for periodically polling the various monitored devices. For each device, the user can specify the list of MIB variables to observe and the frequency of the observations.

In order to improve the efficiency of the collector, multiple hosts are polled in parallel using non-blocking SNMP requests. The maximum number of hosts polled in parallel can be defined by the user, the only limitation being the number of simultaneous opened file descriptors supported by the Operating System.

The SNMP collector stores the observations into a set of Round Robin Databases (RRDs) [13]. A RRD can store time-series data (such as CPU utilization, network load, machine room temperature) in a compact way. Data must be entered into a RRD with a certain frequency. Old data are compacted by means of a *consolidation function* (any of Average, Minimum, Maximum and Last), or discarded. For example, the user may decide to store the average network utilization for the last week with one observation every 10 seconds, and for the last month with one observation every minute. The RRDTool library will take care of compacting observations older than a week by computing the average of six observations. When consolidated data are older than one month they are discarded, so the maximum size of a RRD

is fixed and known in advance. The RRD library provides the capability to plot the collected data in various ways.

The SNMP collector module keeps the status of each machine in the cluster while receiving the observations. Status informations currently include for each host:

- the last observed value for every MIB variable;
- whether the machine is responding to SNMP requests;
- the list of SNMP error messages generated by the machine.

In this way, the collector knows the operational status of each node in the cluster with a maximum delay equal to the time between consecutive polls. and can notify the system managers as soon as a problem arises.

4.2. Configuration File Format

The configuration file for PerfMC is written in XML [21]. XML documents can be created by hand using a generic text editor, or using a specialized XML editor, or automatically generated by an application. The Gnome XML Library [1] is used inside PerfMC to parse, create and transform XML documents. The configuration file obeys the structure declared in the `monitor` Document Type Declaration (DTD), reported in Figure 3.

The configuration file's structure is quite simple. It consists of a sequence of `<host>...</host>` blocks, each one containing informations regarding a specific host to be monitored. Such informations include the list of MIB variables to poll and the list of graphs which can be generated for that host, along with the RRD library commands used to produce the graphs. It is possible to specify the layout of the RRD used to store the collected values and the frequency of observations.

Currently two kinds of graphs can be produced: time-series graphs (specified inside the `<rrdgraph>...</rrdgraph>` tags) for plotting the collected data over a specified interval of time, and Kiviat graphs (inside the `<kiviatgraph>...</kiviatgraph>` tags).

4.3. The HTTP Server

PerfMC provides a WEB interface through an embedded HTTP server, implemented using the SWILL library [10]. The WEB server has access to the in-core status informations about the cluster, which is kept up to date by the SNMP collector thread. Also, the WEB server has read-only access to the Round Robin Databases containing the historical data collected from the cluster. Using the graphing capabilities provided by the RRD library, the WEB server is able to dynamically generate plots from the data.

```
<!ELEMENT monitor ( host )+         >
<!ATTLIST monitor
    numconnections CDATA #IMPLIED
    pmclogfile     CDATA #IMPLIED
    httpdlogfile   CDATA #IMPLIED
    rtrddir        CDATA #IMPLIED
    htmldir        CDATA #IMPLIED
    pmcverbosity   CDATA #IMPLIED
    httpport       CDATA #IMPLIED   >

<!ELEMENT host (description?, mailto?, miblist, archives, graphs) >
<!ATTLIST host
    name      ID       #REQUIRED
    polldelay CDATA    #REQUIRED
    tag       NMTOKENS #IMPLIED     >

<!ELEMENT description  (#PCDATA)*   >
<!ELEMENT mailto       (#PCDATA)*   >
<!ELEMENT miblist      ( mib )*     >
<!ELEMENT mib          EMPTY        >
<!ATTLIST mib
    id        NMTOKEN                       #REQUIRED
    name      CDATA                         #REQUIRED
    type      ( GAUGE | DERIVE | COUNTER ) "GAUGE"
    community NMTOKEN                       #IMPLIED
    options   ( UNKNASZERO )                #IMPLIED
    min       CDATA                         #IMPLIED
    max       CDATA                         #IMPLIED  >

<!ELEMENT archives     ( rra )*   >
<!ELEMENT rra          EMPTY      >
<!ATTLIST rra
    cf          ( AVERAGE | MIN | MAX | LAST )  "AVERAGE"
    xff         CDATA                           #IMPLIED
    granularity CDATA                           #REQUIRED
    expire      CDATA                           #REQUIRED  >

<!ELEMENT graphs ( kiviatgraph | rrdgraph )* >

<!ELEMENT rrdgraph     ( line )+  >
<!ATTLIST rrdgraph
    id          ID     #REQUIRED
    width       CDATA  #IMPLIED
    height      CDATA  #IMPLIED
    seconds     CDATA  #IMPLIED
    title       CDATA  #REQUIRED  >

<!ELEMENT line         (#PCDATA)* >

<!ELEMENT kiviatgraph  ( var )+   >
<!ATTLIST kiviatgraph
    id          ID     #REQUIRED
    title       CDATA  #REQUIRED  >

<!ELEMENT var          (#PCDATA)* >
```

Figure 3. The `monitor` DTD, which defines the structure of the configuration file for PerfMC.

The WEB server can also produce an XML page containing the status of the whole cluster. As described in Section 4.1, the status informations include whether each host is responding to SNMP polls, the last received values of every polled MIB variable and a list of error messages reported by the host. The status document includes also the names of graphs which can be generated for each host in the cluster. An example of XML status document is reported in Figure 4.

The WEB server can apply to the XML status document any user-supplied XSL Transformation (XSLT) [20], and send the result back to the user. An XSL Transformation could produce, for example, an arbitrary XHTML pages from the XML status file, so that normal HTML browsers can display them. Figures 5 and 6 show two XHTML pages generated by applying two different transformations to the same XML status document. Figure 5 shows a page containing the list of all machines in the cluster, with different colors indicating the CPU load of each machine. Figure 6

```
<?xml version="1.0"?>
<hosts>
  <host name="bbr-farm002" status="OK">
    <mibs>
      <mib id="net2Out" lastUpdated="1018016032">534717280.000000</mib>
      <mib id="net1Out" lastUpdated="1018016032">13811037.000000</mib>
      <mib id="net2In" lastUpdated="1018016032">1741169408.000000</mib>
      <mib id="net1In" lastUpdated="1018016032">13811037.000000</mib>
      <mib id="availSwap" lastUpdated="1018016032">530104.000000</mib>
      <mib id="totalSwap" lastUpdated="1018016032">530104.000000</mib>
      <mib id="totalMem" lastUpdated="1018016032">261724.000000</mib>
      <mib id="cachedMem" lastUpdated="1018016032">35376.000000</mib>
      <mib id="bufferMem" lastUpdated="1018016032">14600.000000</mib>
      <mib id="sharedMem" lastUpdated="1018016032">0.000000</mib>
      <mib id="freeMem" lastUpdated="1018016032">97824.000000</mib>
    </mibs>
    <graphs>
      <graph id="hourly.png" title="Hourly data"/>
    </graphs>
    <notifications>
      <msg ts="1017937775.90771 18:29:35.090771" severity="CRITICAL">Timeout</msg>
    </notifications>
  </host>
<hosts>
```

Figure 4. Example of XML status document for a single host

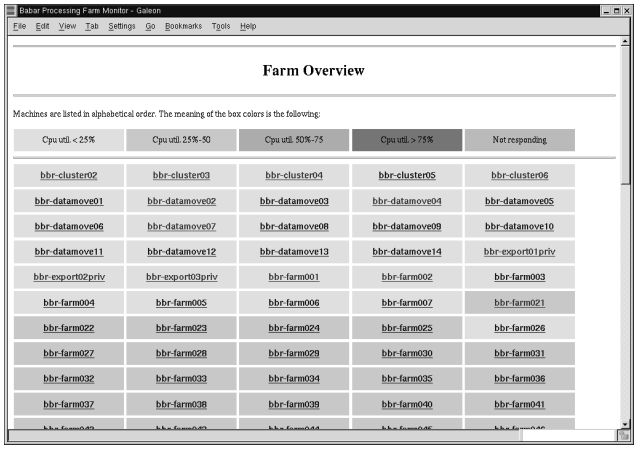

Figure 5.

shows a more detailed view of a single machine, with the latest collected values of all the SNMP variables and some graphs.

The HTTP server recognizes the following types of Uniform Resource Identifiers (URIs) specified in a HTTP GET request:

/<hostname>/<graphname> Returns the graph *<graphname>* for the machine whose identifier is *<hostname>*;

/status.html?applyTransform=<XSLT-file> Applies the XSL Transformation specified in *<XSLT-file>* to the XML document describing the whole cluster status. The result is returned to the user;

/<hostname>/status.html?applyTransform=<XSLT-file> Applies the XSL Transformation specified in *<XSLT-file>* to the XML document describing the status of the single host *<hostname>*. Returns the result to the user.

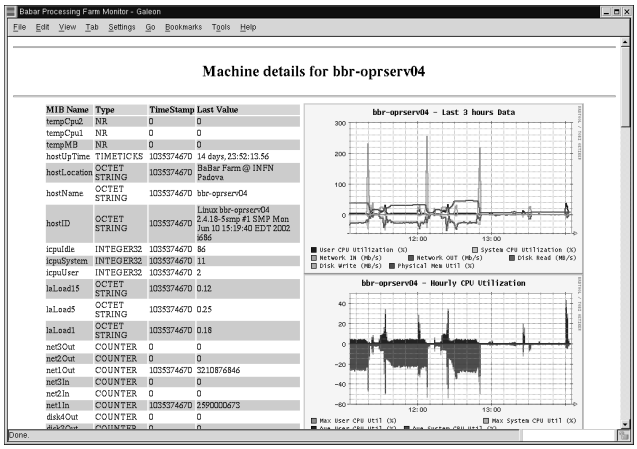

Figure 6.

The possibility of generating XML files by applying user-defined transformations to the status document is particularly useful. The user can customize the appearance of the generated XHTML pages by simply writing a set of XSLT files (which are usually very compact); in this way the XHTML pages are not hard-coded inside PerfMC and can be changed when needed without recompiling the main program. Moreover, being XML a widely accepted standard, it qualifies as a suitable mean for interchanging informations between PerfMC and other applications. XSL Transformations can be used to restructure and filter the data returned by PerfMC before passing them to an application.

5. A Case Study

PerfMC is currently being used to monitor a Linux cluster used for High Energy Physics applications. The cluster is hosted at INFN Padova, Italy and is used to process very high volumes of data using CPU-intensive batch applications. It is made of about 170 dual processor, Linux/Intel machines, partitioned in about 130 clients and 40 servers. Both classes of machines use 1.26GHz Pentium III processors and have 1GB of RAM. Client machines have two fast Ethernet controllers (at the moment just one Ethernet board is used). Server machines have gigabit Ethernet controllers and 1TB of local disk space as EIDE Raid arrays. The cluster is interconnected using a high-performance network switch, and is attached to a tape library.

For each machine in the cluster, the value of the SNMP variables listed in Table 1 are monitored every 30 seconds.

Every quantity, with the exception of host ID, Name, Location and Uptime, is saved in a Round Robin Database

Temp. of the MBoard	Temp. of the 1st CPU
Temp. of the 2nd CPU	Free Memory
Shared Memory	Buffered Memory
Cached Memory	Total Memory
Total Swap	Available Swap
Tot. Disk Blks read	Tot. Disk Blks written
Disk 1–5 Blks read	Disk 1–6 Blks written
Net 1–4 Bytes in	Net 1–4 Bytes out
`/tmp` space used	`/tmp` space avail
`/var` space used	`/var` space avail
`/usr` space used	`/usr` space avail
Load Average last minute	Load Average last 5 mins
Load Average last 10 mins	Host ID
Host Name	Host Location
Host Uptime	

Table 1. Variables monitored for each host on the test cluster

which records its average and maximum value. Data for the last week are kept with the granularity of one observation (average and maximum) every minute. Data for the last month are kept with the granularity of one observation every hour. Data for the last year are kept with the granularity of one observation every day; data more than one year old are discarded. The total size of each Round Robin Database is about 8 MBytes.

All the monitored variables for a single machine are requested with a single SNMP packet. Network statistics collected with the `tcpdump(8)` utility shows that the size of an SNMP request is ≈ 800 bytes, while the size of the response is ≈ 1000 bytes. This gives an average network utilization of approximately 11KB/s. The CPU overhead on the machines caused by the monitoring activity is negligible. The monitoring program (PerfMC) runs on a dedicated machine, with a very low (less than 5%) CPU utilization.

6. Conclusions and future work

In this paper we have illustrated the architecture of a monitoring system for large computing clusters. A prototype written using the C language, has been implemented and is being used to monitor a cluster with 170 dual-processor Linux/Intel machines. The monitoring system uses asynchronous (non-blocking) parallel SNMP bulk requests to collect status informations from a wide variety of networked devices, and incorporates an HTTP server which can generate graphs from the collected data. The HTTP server can also produce an XML encoding of the current cluster status, to which an XSL Transformation can optionally be applied.

We believe that the most essential goals among those stated in Section 3 have been satisfied. The system is not intrusive in that all it needs is an SNMP agent running on each monitored device. At low polling rates (≈ 10 seconds) the overhead on the network and on the observed devices is very low. We do not recommend the use of our system is higher, sub-second polling rates are required; in those cases a more specialized profiling system such as Supermon [12] should be preferable.

The scalability of PerfMC has been obtained by a clean design and an efficient implementation. Polling many hosts in parallel is a very trivial idea which indeed helped very much. The use of SNMPv2 bulk operations allowed to get the values of many variables from a single machine with just a request/response pair of packet, reducing the load on the network. Also, SNMP is a standard protocol and is implemented in virtually every device. The freely available implementation by the NET-SNMP project allows the user to extend the list of standard SNMP MIBs without the need to modify the agent's code. In this way it is possible to monitor everything one could be interested in. We are currently using this feature to monitor the status and progress informations from the processes running on our cluster.

Finally, the heavy use of XML as the format for the configuration file, and that of XSLT to transform the status informations in arbitrary ways proved to be a good idea. XSLT transformations are used at the moment to produce a set of HTML pages showing in different ways the status of our cluster.

At the moment the prototype does not implement the alarm system. An alarm system is obviously needed to notify the system administrators in case of failures, so we are currently working on it. The alarm system will likely be implemented by listing in the configuration file a set of thresholds for each MIB variable. If a threshold is crossed in the specified direction, an alarm will be triggered. An additional alarm will be associated with each machine, and will be triggered if the machine does not reply to SNMP queries. Such a threshold-based alarm system is exactly the same implemented by the RMON protocol [16], so alarms can be triggered directly by SNMP agents implementing RMON specifications.

The prototype implementation is performing well on our cluster, and no scalability limit has been encountered so far. However, it is obvious that a centralized monitoring system, even the most efficient one, cannot scale forever. In particular, we identified the updating of the Round Robin Databases as the most likely candidate bottleneck. As a first solution, we are currently trying to identify possible sources of inefficiencies in the RRDTool package. As a more long-term fix, we are considering the idea of partitioning the whole cluster among different monitors, each one observing a subset of the system. This would alleviate the scalability problem, as arbitrarily large clusters can be mon-

itored by simply adding more monitors running on different machines. On top of these monitors, it is possible to build a hierarchy of *monitoring proxies* which will be used to fetch and consolidate the informations collected from the nodes behind them. The top (root) node will present a global view of the system to the user, or redirect user's requests to the monitor responsible for observing the requested resource. Fault-tolerance can be implemented by means of standard techniques, such as electing a substitute when one of the monitors crashes. The user interface based on HTTP and XML was developed because it could also be used to exchange informations among monitors. The current PerfMC implementation can be extended to cope with a hierarchical monitoring infrastructure. What is required is the addition of a suitable client HTTP interface which can be used to contact the HTTP server embedded in other monitors. In the same way it is possible to add other kinds of interfaces, such as a Lightweight Directory Access Protocol (LDAP) interface, or a text-only user interface.

7. Acknowledgments

This work has been partially supported by the Istituto Nazionale di Fisica Nucleare, Padova. The author thanks Valerio Melloni for the implementation of PerfMC's WEB server with the SWILL library. R. Stroili and the anonymous referees are also acknowledged for their constructive comments.

References

[1] libxml: The XML library for Gnome. http://xmlsoft.org/.

[2] GANGLIA project page. http://ganglia.sourceforge.net/.

[3] NET-SNMP project page. http://net-snmp.sourceforge.net/.

[4] J. M. Anderson, W. E. Weihl, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. A. Leung, R. L. Sites, M. T. Vandevoorde, and C. A. Waldspurger. Continuous profiling: Where have all the cycles gone? *ACM Transactions on Computer Systems*, 15:357–390, Nov. 1997.

[5] R. Buyya. PARMON: a portable and scalable monitoring system for clusters. *Software–Practice and Experience*, 30:1–17, June 2000.

[6] V. Catania, A. Puliafito, S. Riccobene, and L. Vita. Monitoring performance in distributed systems. *Computer Communications*, 19(9–10):788–803, Aug. 1996.

[7] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[8] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley & Sons, 1991.

[9] A. King and R. Hunt. Protocols and architectures for managing TCP/IP network infrastructures. *Computer Communications*, 23(16):1558–1572, Sept. 2000.

[10] S. Lampoudi and D. M. Beazley. SWILL: A simple embedded web server library. In *Proceedings of Freenix'02*.

[11] M. Mansouri-Samani. *Monitoring of Distributed Systems*. Phd thesis, Department of Computing, Imperial College, London, 1995.

[12] R. Minnich and K. Reid. Supermon: High performance monitoring for linux clusters. In *The Fifth Annual Linux Showcase and Conference*, Nov. 2001.

[13] T. Oetiker. RRDtool home page. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/.

[14] A. Puliafito and O. Tomarchio. Using mobile agents to implement flexible network management strategies. *Computer Communications*, 23(9):708–719, Apr. 2000.

[15] C. B. Saab, X. Bonnaire, and B. Folliot. PHOENIX: A self adaptable monitoring platform for cluster management. *Cluster Computing*, 5(1):75–85, Jan. 2002.

[16] W. Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Pub. Co., 3rd edition, 1999.

[17] R. Subramanyan, J. Miguel-Alonso, and J. A. B. Fortes. A scalable snmp-based distributed monitoring system for heterogeneous network computing. In *Proceedings Supercomputing 2000*, Dallas, Texas, USA, Nov. 2000. IEEE Computer Society,.

[18] B. Tierney, B. Crowley, D. Gunter, J. Lee, and M. Thompson. A monitoring sensor management system for Grid environments. *Cluster Computing*, 4(1):19–28, Mar. 2001.

[19] P. Uthayopas and S. Phatanapherom. Fast and scalable real-time monitoring system for Beowulf clusters. *Lecture Notes in Computer Science*, 2131, 2001.

[20] W3 Consortium. XSL transformations (XSLT) version 1.0. W3C Recommendation, Nov. 1999. Available online at http://www.w3.org/TR/xslt.

[21] W3 Consortium. Extensible markup language (XML) 1.0 (second edition). W3C Recommendation, Oct. 2000. Available online at http://www.w3.org/TR/2000/REC-xml-20001006.

[22] W3 Consortium. XHTML basic. W3C Recommendation, Dec. 2000. Available online at http://www.w3.org/TR/xhtml-basic/.

[23] R. Wismüller, J. Trinitis, and T. Ludwig. OCM – a monitoring system for interoperable tools. In *Proc. 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98*, pages 1–9, Welches, OR, USA, Aug. 1998. ACM Press.