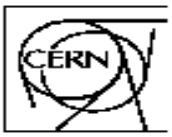
Available on CMS information server **CMS NOTE 2003/013**

The Compact Muon Solenoid Experiment
CMS Note
Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland



May 30, 2003

Resource Monitoring Tool for CMS production

A. Osman[1]
PINSTECH, Islamabad, Pakistan
T. Wildish[1]
Princeton University, Princeton, USA
I. Willers[1],
CMS, CERN, Geneva, Switzerland.

Abstract

A monitoring tool is described which not only tracks and recognises errors but also works together with a management system that is responsible for resource allocation. In cluster/grid computing, the resources of all accessible computers are at the disposal of end users. With that much power at hand, the responsibility of the software managing these resources also increases. The better utilization of resources means that a monitoring system should make the collected data persistent, so that the management system has up-to-date information but also has a meaningful historical record. This database can then be consulted for finding the best available resources in a given scenario, and can also be used for understanding historical trends. The Resource Monitoring Tool, RMT, is such a tool, which caters for these needs. Its framework is designed in such a way that its potential can be enhanced easily by adding more modules.

[1]Working at CMS-CERN, Geneva, Switzerland

Table of Contents

ABSTRACT **1**

RESOURCE MONITORING TOOL FOR CMS PRODUCTION **3**

1. INTRODUCTION **3**

2. ARCHITECTURAL OVERVIEW **3**

3. AGENTRC OPERATIONS **6**

4. FORMAT OF DATA MESSAGES **6**

5. AVOIDING DEADLOCK **7**

6. HEARTBEAT MONITORING **8**

7. SECURITY ISSUES **8**

8. DATA PERSISTENCE **9**

9. DIFFERENT TECHNIQUES FOR LAUNCHING AGENTS **11**

10. CONCLUSION **12**

REFERENCES **13**

RELEVANT WEB SITES **13**

ACKNOWLEDGEMENTS **14**

FIGURES TABLE

Figure 1: Resource Monitor Architecture 4
Figure 2: Modules Spawning and Data Collection 5
Figure 3: Heartbeat Status 6
Figure 4: Working of AgentRC 7
Figure 5: Heartbeat Monitoring 8
Figure 6: Schema for RMT database 9
Figure 7: Web site for displaying monitoring data 10
Figure 8: Plot of historic data for CPU Load 11

Resource Monitoring Tool for CMS production

1. Introduction

CERN/CMS is a collaboration of many member states. Scientists and engineers from many institutes participate in the experiments being performed at CERN. Its structure naturally fits the model of distributed computing. Data being generated from the experiment and moved from one centre to another will be in Petabytes per year, unprecedented in the history of computer science. Different tasks are being performed in parallel: simulation, reconstruction, scheduled and unscheduled analysis. Every physicist should have access rights to use this data and transparent access to dynamic resources. As a result, this data will be migrated to different centers, analyzed and results thus obtained will be exchanged among different regional centers located in every corner of the world.

Distributed computing requires a monitoring tool, which keeps a watch on the resource utilization of the nodes participating in the cluster and keeps track of the resource utilization of these machines. It should attempt to predict any kind of problem occurring on any machine and it should be able to track the trend analysis. The data thus obtained can be utilized for optimal use of the Resources.

This is a short note on the architectural and implementation view of the package. More details are available in User Guide (which will become available from https://savannah.cern.ch/projects/octopus).

2. Architectural Overview

The Resource Monitoring Tool (RMT) uses a Manager/Agent strategy. The Manager/Agent model is similar to the Client/Server model; only in this case Agents are distributed on every client machine that participates in the Farm and whose resources require monitoring. The Manager program is executed on a central machine, which obtains the data sent by these agents, stores them in database and displays the results on the central control machine. These agents are also residing on the devices used for network communications. Normally, these devices support SNMP (Simple Network Management Tool) protocol [1] and their agents are built-in by the manufacturer. Snmpget and snmpwalk utilities can be used to get information from these devices. These SNMP agents are also capable of generating traps and notifications in case of occurrence of predefined events. The definition of these events can be tailored in the configuration files available with the SNMP installation.

SNMP is a good example of a protocol used for resource monitoring. This protocol is used as the basis for collecting resource information from nodes participating in a cluster. Unfortunately, SNMP daemons require root access for their installation, and some sites prefer not to install it for security reasons. On the other hand there are several parameters, which a system manager wants to watch on his daily experience basis. It requires a quick augmentation of package in an easy way. This tool is developed to cater for these needs. It uses SNMP as well as custom made modules to achieve such goals. A new term Remote Control Agent is coined to cater for this aspect of the monitoring needs. Figure 1 depicts the overall architecture showing how

a Manager program launches and controls these agents remotely, as well as the Errors/Information messages, and the Manager collects data sent by agents. The people depicted are observing the statistics collected by the monitoring system running on clusters A and B, via a standard web browser. The DataLogger is recording the information via DataPersistence for later analysis. The ErrorLogger records error situations. The repository for modules that are to be downloaded contains net-stat, read-stat, etc.

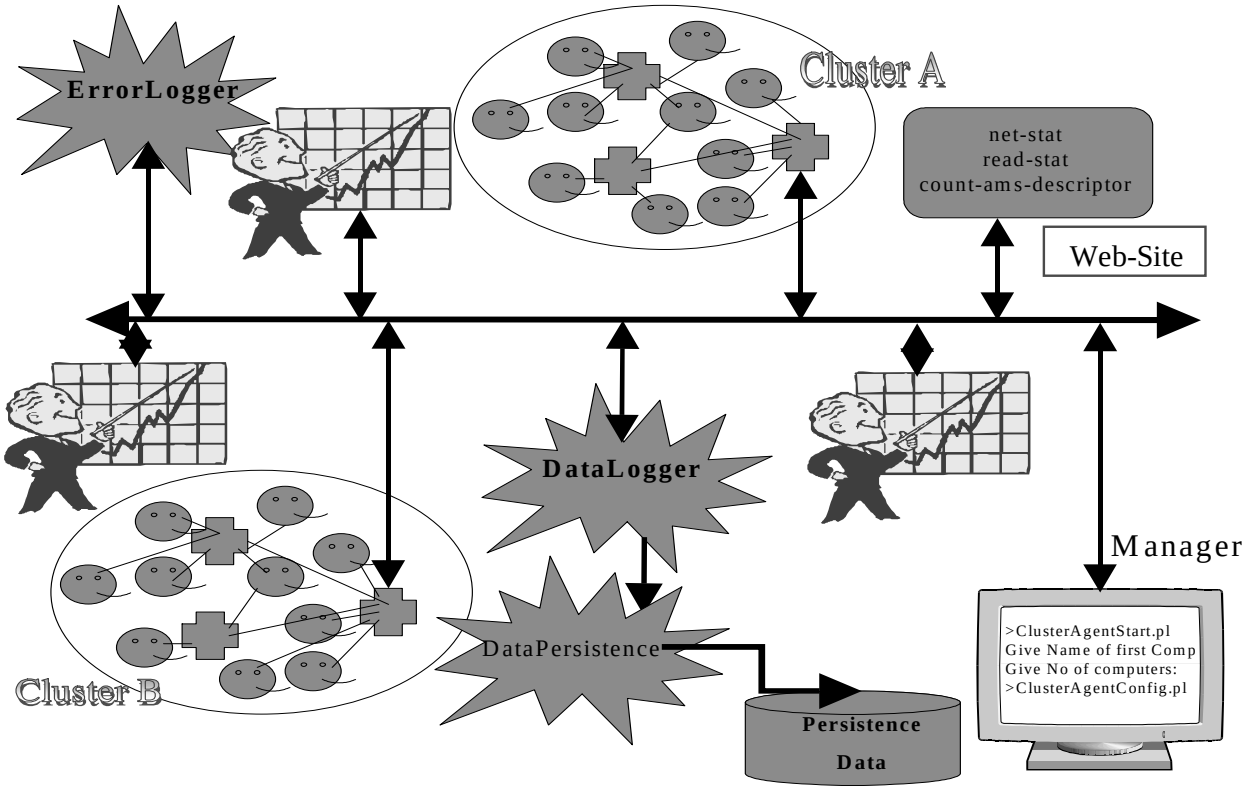


Figure 1: Resource Monitor Architecture

AgentRC is a new term introduced to name the kind of agents which can be remotely controlled and tuned, with the help of commands issued by a Manager. These are the agents, which can be loaded on standard machines supporting standard platforms. These Agents will work as the engine to execute a number of commands being issued by the Manager. For collecting information these Agents will provide the environment for downloading and executing a set of modules from a central repository. These modules can be designed differently depending upon the environment of the node on which it will run.

All the modules developed can be stored on a web site. The AgentRC command “download module” can be issued by the Manager to download any particular module in the directory where AgentRC runs, so that it becomes an add-on extension of the

AgentRC package. The “add” command can spawn this module as a child process of AgentRC. This process is pictorially depicted in Figure 2.

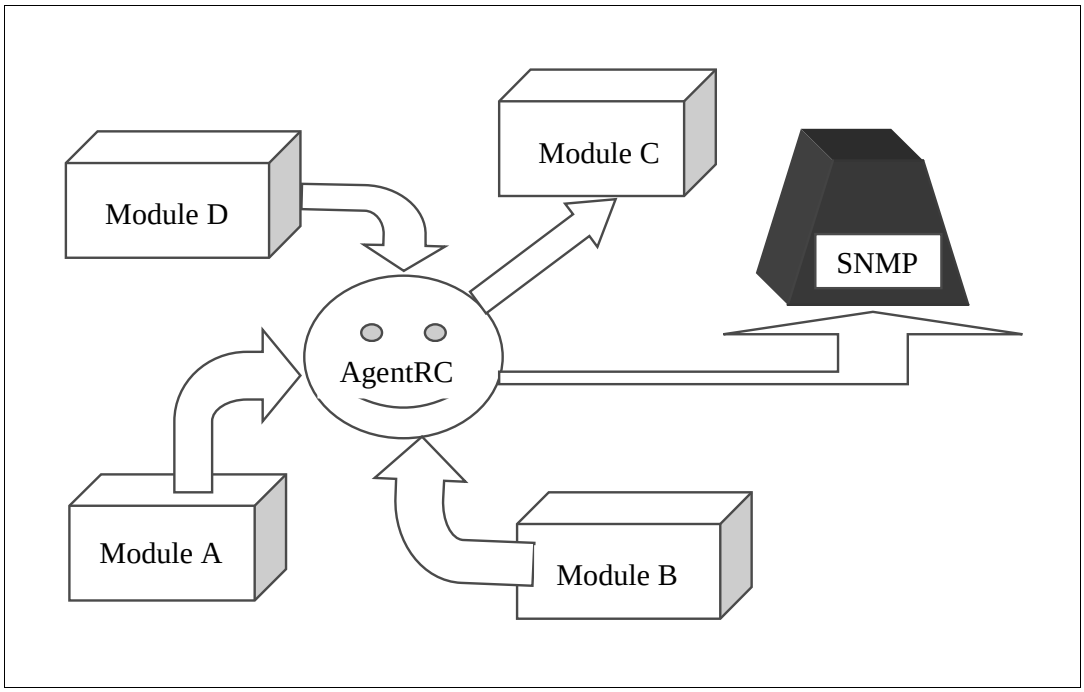


Figure 2: Modules Spawning and Data Collection

These modules can be written in any language, provided it generates data on STDOUT. The data collected by the modules should be in key/value pair form with the first two pairs mentioning the Date+Time associated with the data being generated and the node name. This data is redirected to the AgentRC that spawned the module. This agent wraps the data in SNMP datagrams and sends it over the network via a UDP socket. These data can be collected by the DataLogger, which can display it on the screen or can make it persistent by pipelining it to another script that stores it in a MySQL database.

A web-based interface is available, which can query the database, retrieve the selected data from the database and present it on the web in graphical, textual or postscript form.

This interface also provides the status of all nodes being currently monitored in a similar manner to Microsoft Network Neighborhood, as shown in Figure 3. Normally, all the nodes on this screen are displayed in one color, but its color changes if that node is malfunctioning due to any reason. If one clicks on any of these nodes, detailed statistics are provided concerning the selected node.

Another feature of this package is to raise an Alarm if some monitoring parameters are out of range. These alarms can be configured to automatically generate emails to inform any interested party. The alarms are also stored in a database, from where they can be queried via a web interface, using restricted SQL commands.

Figure 3: Heartbeat Status

3. AgentRC Operations

The AgentRC executes a loop, waiting to accept a command from the Manager on a designated port. If it gets a command within a specified cycle-time, it tries to execute it and again goes into the loop waiting for next command. But if no command is received within a specified cycle-time, it executes the child modules one by one and on finishing one cycle for every module execution, it again goes into a loop. The working of an AgentRC is elaborated in Figure 4.

4. Format of Data Messages

All the messages generated by agents are in the generic Net-Logger form having a "key=value" format. The DATE, HOST and EVNT fields are prefixed to every message. Event name depends upon the type of information gathered. There are some specific events, such as:

- ALARM
- START
- HEARTBEAT
- CONFIG

When these specific events are encountered the DataLogger.pl and DataPersistance.pl programs take specific actions.

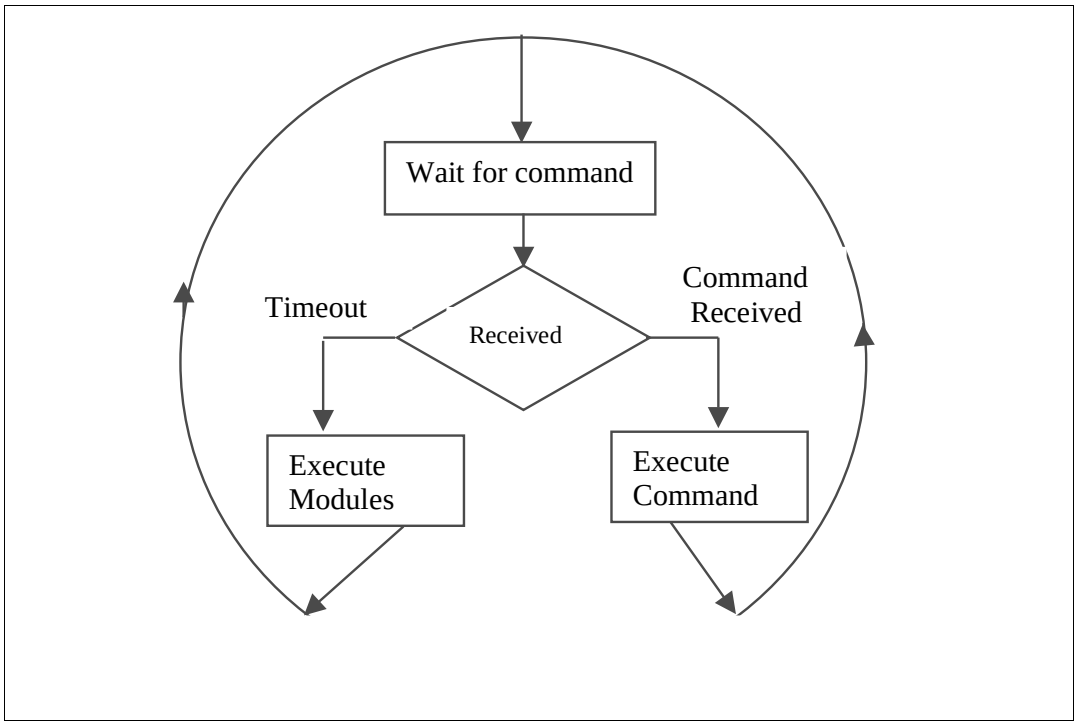


Figure 4: Working of AgentRC

5. Avoiding Deadlock

The RMT package is developed in a hierarchical manner. First some basic commands are defined, which are interpreted and executed by the AgentRC. Based on these programs other scripts are written to perform a task. These programs are further encapsulated within higher-level commands to obtain a higher-level abstraction. Therefore, if any one code in this hierarchy becomes stuck, the rest of the dependent programs also likely to be stuck. Checks are made at various stages to avoid such deadlocks. The following measures are taken to avoid such a situation:

- The UDP protocol is used for communication, so that an agent should not wait for the arrival of datagrams indefinitely
- Child processes are started by opening a pipeline using the open command
- Agents check their child processes periodically and report any deaths by email
- To deliver the data to a parent process, status bits are set by child processes, which ensures that an agent will go and read the information only when there is some data available to read
- Most of the jobs are submitted with a timeout. If a job is not executed and it is timed out after specified time, it will be killed
- System commands wherever used are qualified by a timeout if one is available
- Stuck processes are killed after the completion of each cycle
- Those Agents which are stuck due to any reason are stopped and restarted

6. Heartbeat Monitoring

Each node of the cluster is running an AgentRC. A script defines this agent, which, once started will be executed, in an infinite loop. It is nonetheless possible that things can go badly enough wrong that the AgentRC gets stuck or dies. To keep watch over these types of issues, the AgentRC will keep on sending its heartbeat, so that in case of system failure, an alarm could be generated. DataLogger.pl keeps on accepting data from each node and it builds a hash table recording the status of each node. If the heartbeat coming from any node is not received within the specified time interval, the node is considered to be malfunctioning and it is reflected in the heartbeat status. The RMT package generates an alarm and takes action to rectify the problem. The scheme used for monitoring the heartbeat and for generating alarms is shown in Figure 5.

Here every agent is supposed to send a signal to the DataLogger.pl program after the cycle time has elapsed. A counter corresponding to every node is maintained in DataLogger.pl. This counter will count-up on receiving the heartbeat cycle. Another, master clock is maintained, which will decrement all the counters after every cycle-time. If the value of any counter goes to negative, it implies that there is some problem with that node or agent, so an alarm is generated.

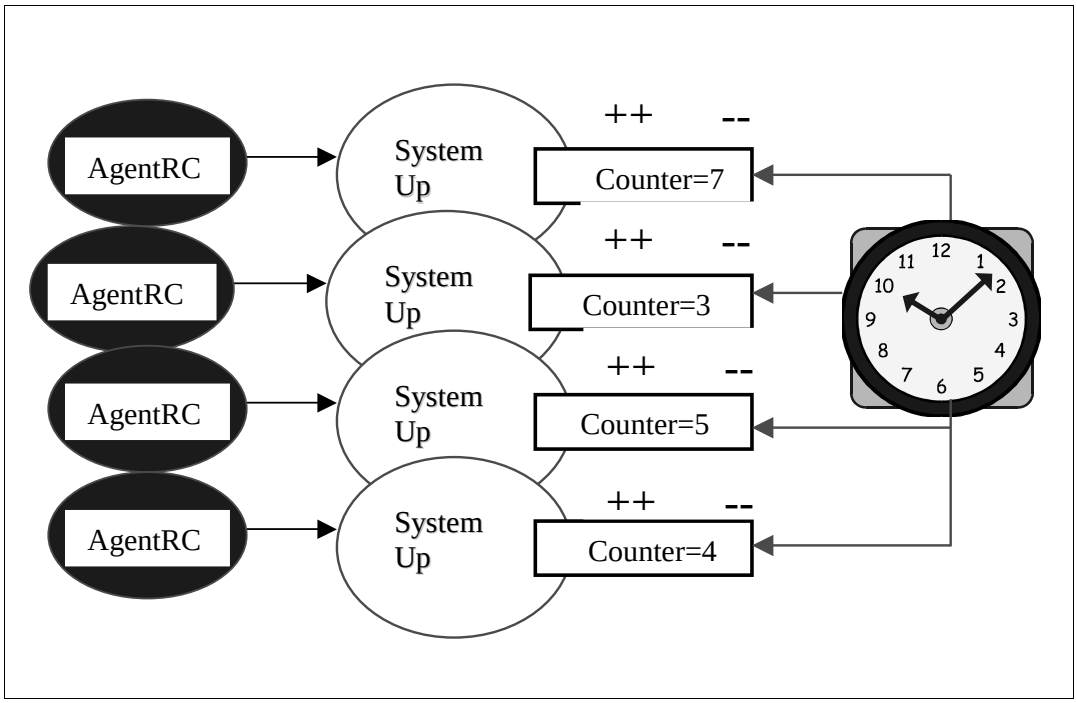


Figure 5: Heartbeat Monitoring

7. Security Issues

AgentRC and all the modules spawned by it are executed under normal user privilege, so there is no privilege command that can jeopardize the security of the system in general. All the data being transmitted will use the SNMP protocol as wrapper, so it is as secure as the SNMP protocol itself. Currently, we are supporting version 1 & 2c,

and we plan to incorporate version 3 also.

Another issue, although not directly related to security issues, was that of termination of jobs after expiration of the AFS token on the Manager machine. This issue was solved by automatic renewal of the token periodically, by using the klog command. This command requires a password as its argument. Hard coding this password as plain text is clearly a security leak. To avoid this problem, a script is written which on its execution, prompts for the password in no-echo mode and forks a child process. This fork process pipes the klog command with the help of open command. It also gets the password from its parent and executes klog in such a way that its password cannot be detected. More details are available in User Guide.

8. Data Persistence

Any data being captured by the DataLogger.pl script can be displayed directly on the screen, but it can also be made persistent by pipelining the data to the DataPersistence.pl script, which will store it in a MySQL database. The schema used for storing data is illustrated in Figure 6.

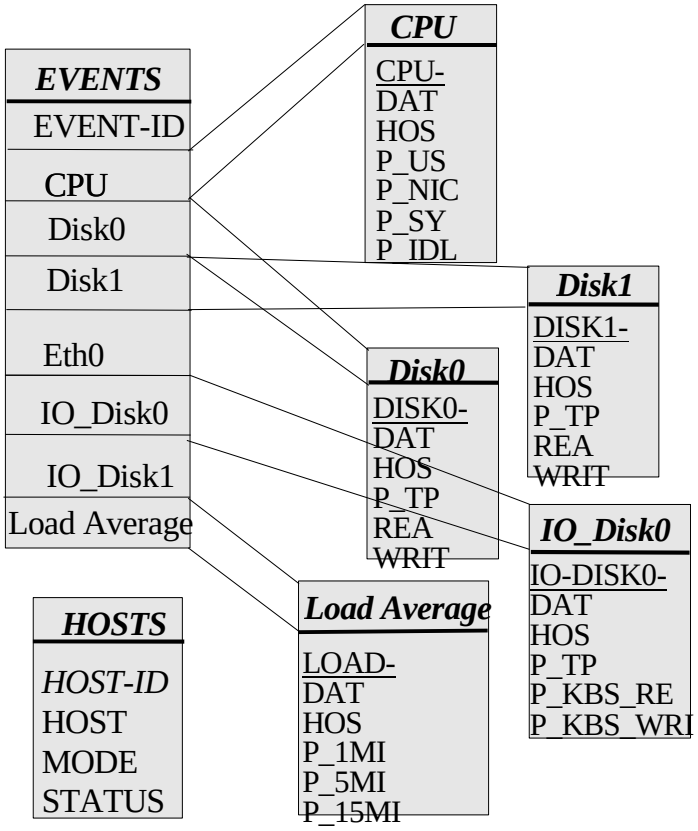


Figure 6: Schema for RMT database

In this schema, all the information is supposed to be retrieved event by event and

there is a separate table for each event type. Every time a new event type is received, a new table with that event name is created dynamically. All the fields of that record correspond to field names that are used to create a schema and a new table is created with that schema. Similarly, when a new computer is being added to the cluster, it is automatically registered in the database.

When any user wants to query the database, the following web site can be used:

http://cmsdoc.cern.ch/cms/production/www/monitoring/RMT/Monitor.html

It displays an interface as shown in Figure 7:

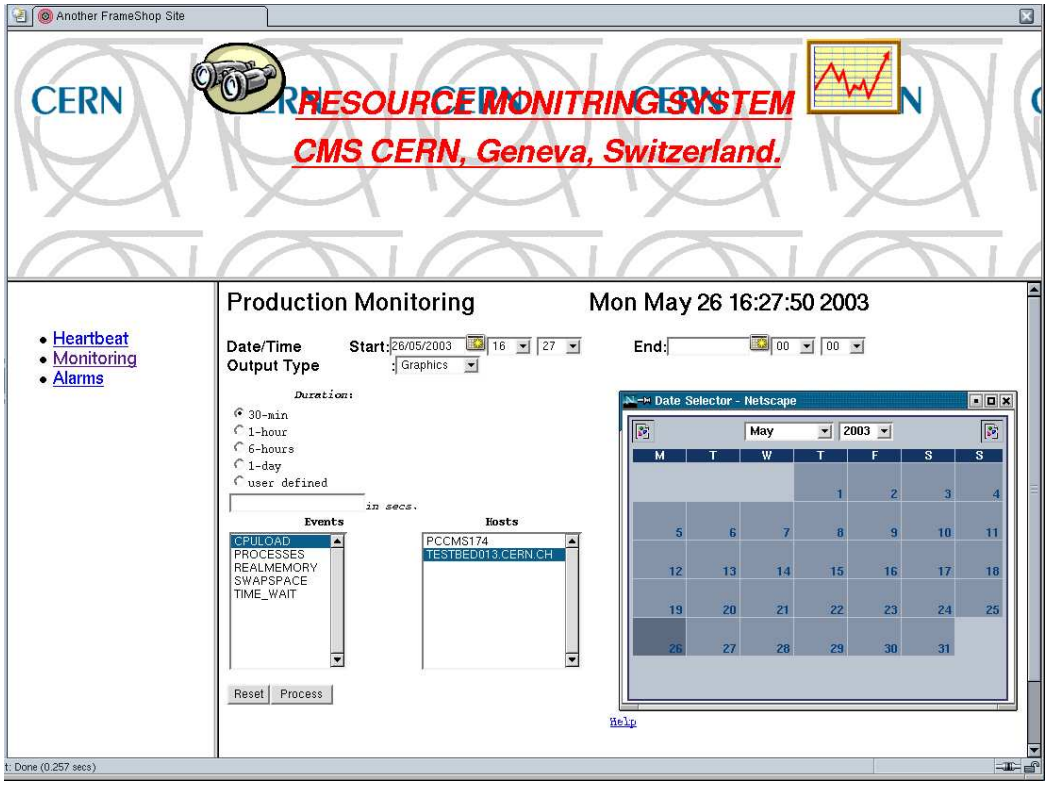


Figure 7: Web site for displaying monitoring data

Internally, depending upon the option being selected by the user, a query is dynamically formed and executed against the database. Results thus obtained are submitted to GNUplot by pipelining, without creating a temporary file. It avoids the problem of synchronization that could happen from the creation of temporary files by many users simultaneously accessing the database. Using this interface one can select the event, computer and the time range for observing the data. The output can be displayed in Graphics, Text or Postscript form. A sample output is shown in Figure 8

Another point to be noted in this package is the need to switch “Off” the cache feature of the HTML, which is set to “On” as default. With this feature “On”, it may display the output of a previous query.

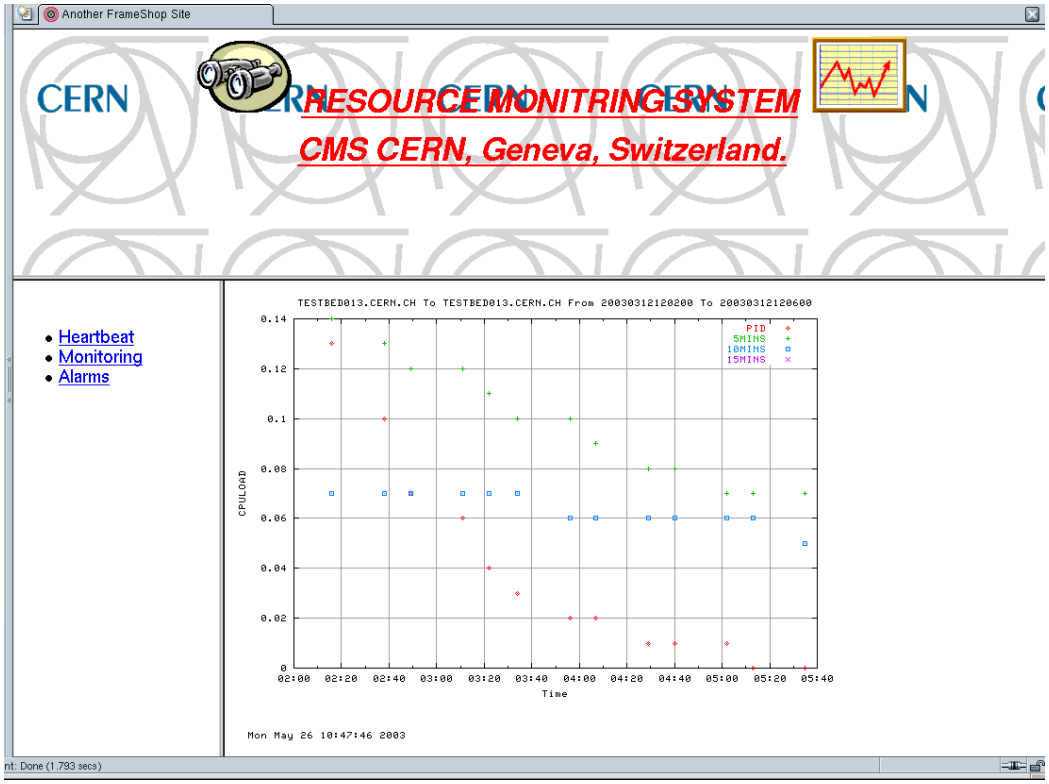


Figure 8: Plot of historic data for CPU Load

9. Different techniques for launching agents

For remotely launching and control of agents on a farm's nodes, one can use different techniques, depending upon the environment available. For example, ssh, lsf, batch mode etc.

This package can launch and execute the agents remotely by issuing ssh commands. At CERN (and probably elsewhere) this command requires a valid AFS token otherwise it does not work. Most of the commands based on ssh are hidden in scripts, which are running periodically. These commands will fail as the token expires. To solve this issue, one has to renew the token periodically to save it from expiry as discussed in section 7. But this is not the only problem with ssh. On the first time a node is contacted, it wants confirmation from the user in order to add the hostname in its cache. During the execution of this package most of the ssh commands are issued by different scripts, and there is no user to prompt. Again, if the key of remote machines changes, ssh generates a warning message and asks for action. At that moment, the entry of the remote machine is required to be removed from ~/.ssh/known_hosts file. These problems are tackled in this package and discussed in the User Guide.

If you have the LSF batch processing system in your environment, you can use its interactive commands for launching the agents. This has the advantage that all the problems faced with ssh are solved. This package can use lsf interactive commands. It can also be tailored to use other batch processing systems, PBS, FBS, and Condor etc.

This package also supports running the agents in the grid environment. In case of the grid, we decided to collect the data on a job-to-job basis. Rather than monitoring the entire grid we choose to monitor only what we are doing on the grid, which makes this very useful for experiment-specific use. This is in contrast to other grid-monitoring packages that monitor machines with little regard to who is doing what on them. This package was adopted in such a way that agents could be packaged along with the job for submission to the grid. For using an agent from a batch job, network connectivity is not guaranteed. So, data is not transmitted by UDP as in single-site operation. Instead, the same data is collected in the form of a file, which is recovered from the grid in the output sandbox and used to update the database afterwards. A further improvement was made on the construction of modules, so that it can collect data on choice and it should be possible to change the logic of alarm notifications through the configuration file. It means you can tailor your collection of data and notification on a job-to-job basis as per requirement. So, the agent's configuration file can be tailored to reflect the actual work performed by the job. For example, depletion of free disk space can be watched for a job, which is hungrier for disk space, while one can give more attention to CPU load or ignore network activity if it is a compute bound job etc.

10. Conclusion

This package has been installed and tested on the local CMS production farm at CERN. It has run on more than 400 computers for several days and performs well. The scalability issue is mostly related to udp datagram losses. This can be tuned by setting the cycle-time of acquiring the data by the modules, but also illustrates why UDP was chosen. It is better to lose a monitoring packet than to block a TCP socket trying to deliver the data!

RMT has been tested on the European Data Grid (EDG) [7] with CMKIN Monte Carlo generation production jobs, and works as required. Now, it is planned to integrate it with the OCTOPUS package of CMS production [8].

References

1. MySQL and mSQL by R. J. Yarger, G. Reese; T. King, O'Reilly & Associates Inc.
2. Programming Perl by L. Wall; Christiansen; R. L. Schwartz, O'Reilly & Associates Inc.
3. Best Unix Tips ever by K. H. Rosen; R. H. Rosinksi; D. A. Host, Osborne.
4. Production Monitoring Tool, User Guide.

Relevant Web Sites

1. http://net-snmp.sourceforge.net
2. http://www-isd.fnal.gov/ngop
3. http://www.grateful.net
4. http://www.wtcs.org/snmp4tpc/mrtg.htm
5. http://www.nagios.org/
6. http://sourceforge.net/projects/nagiosplug/
7. http://marianne.in2p3.fr/datagrid/documentation/EDG-Users-Guide/html_tf.html
8. http://cmsdoc.cern.ch/cms/production/www/html/general/index.html

Acknowledgements

Authors are grateful to Veronique Lefebure, Nick Sinanis, Werner Jank and Julia Andreeva for their valuable technical help.

Acknowledgement is also due to Saima Iqbal and Farooq Ahmed for their valuable comments and suggestions given during the preparation of this document.

Thanks to Yunjun Wu of Fermi Lab., who installed and tested an early version of RMT. We received a lot of valuable suggestions and comments from him.