# RVision: An Open and High Configurable Tool for Cluster Monitoring[*]

Tiago C. Ferreto
*Research Center in High Performance Computing CPAD-PUCRS/HP Porto Alegre, Brazil*
ferreto@cpad.pucrs.br

César A. F. De Rose
*Catholic University of Rio Grande do Sul Computer Science Department Porto Alegre, Brazil*
derose@inf.pucrs.br

Luiz De Rose
*Advanced Computing Technology Center T.J. Watson Research Center Yorktown Heights, NY, USA*
laderose@us.ibm.com

## Abstract

*In this paper we present the design and implementation of RVision (Remote Vision), an open architecture, high configurable tool for cluster monitoring. We focus on the description of it's modular architecture, emphasizing the new concepts we are introducing for cluster monitoring, such as monitoring sessions and the support for dynamic linking of monitoring libraries. In addition, we measure intrusion with several benchmarks and applications under different scenarios. RVision distinguishes itself from other available tools for cluster monitoring because of its open architecture, high configurability, and low intrusion. It is being used in production mode, in our research center, and has proven itself as a powerful alternative for cluster monitoring, especially in heterogeneous clusters and cluster of clusters.*

## 1. Introduction

Clusters of workstations are nowadays one of the most used alternative for parallel systems construction mainly because of its high scalability and excellent cost-performance ratio. However, the effective use of clusters depends on efficient resource management and detailed system tuning, which require the observation and optimization of several parameters, such as CPU utilization, network utilization, I/O activity, application parallelism, multiprogramming level, etc. In order to be able to obtain high throughput and/or high response time on clusters, system administrators need tools for monitoring the performance of the system activities and for diagnosing performance problems.

In this paper we presented RVision, an open architecture high configurable tool for cluster monitoring. RVision provides flexibility in selecting events to be monitored and allows users to expand the monitoring capabilities with self-defined procedures for monitoring of specific system hardware or system events. An additional feature of RVision is its flexibility in allowing the monitoring of distinct clusters, cluster of clusters, and heterogeneous clusters. The main contributions of RVision, described in this paper, are the new concepts being introduced for cluster monitoring, namely monitoring sessions and support for dynamic linking of monitoring libraries.

The remainder of this paper is organized as follows. We begin in Section 2 with a brief review of system performance monitoring and some issues that should be taken in consideration when designing and implementing a monitoring system. In Section 3 we address related work. In Section 4 we describe the main aspects of the RVision's design and implementation. In Section 5 we present some performance measurements. Finally, we summarize our conclusions and directions for future work in Section 6.

## 2. Monitoring

System performance monitoring is the act of collecting system performance parameters such as node's CPU utilization, memory usage, I/O and interrupts rate, and present them in a form that can be easily understood by the system administrator [4]. This service is important for the stable operation of large clusters because it allows the system administrator to spot potential problems earlier. Moreover, other parts of the systems software can also benefit from the information provided. For example, the information can be used to modify the task scheduling, in order to improve load balancing. We review next some important issues that should be considered in the project and implementation of such monitoring systems.

**Intrusiveness:** To obtain the highest possible performance in a cluster, the parallel application should be able to get all of the available processing power. However,

[*] Work partialy supported by HP Brazil and Fapergs.

the monitoring system has some processing and communication costs and it will compete with the running application. Therefore, to enable high performance execution in the presence of monitoring, the monitoring tool should have low intrusion, producing minimal interference.

**Network Traffic:** The network is one of the main bottlenecks in most parallel systems. The following techniques can be used to reduce network intrusiveness generated by the monitoring system:

- Adjusting the monitoring frequency: with properly monitoring frequency adjustments the monitored data will be delivered in larger periods of time.
- Data set selection: the definition of the needed monitoring information in a case by case basis decreases the amount of data sent from the resources.

**Web Technology:** Since the advent of the Internet, an increasing number of applications are designed to enable the utilization of Web technology. Following this trend, many monitoring systems provide an Internet layer to enable remote monitoring. This allows a user to monitor a cluster from any place, just needing an Internet connection.

## 3. Related Work

There are several cluster monitoring tools freely available. The most used are Bwatch [1], PARMON [7], and SCMS [14]. BWatch (or beoWatch) is a simple Tcl/Tk program that produces a screen showing load and memory statistics for each node of the system. It does not use any daemons in the cluster nodes; instead it acquires the information executing remote shell (rsh) commands to all nodes listed in a configuration file. The information acquisition is triggered by an explicit user request (i.e., the information is acquired when the user pushes the refresh button showed in the BWatch graphical interface) or when this is automatically done by a defined auto refresh delay in the configuration file.

PARMON uses the client-server model. It allows the monitoring of system resources in three different levels: entire system, node, and component levels. Monitoring multiple instances of the same component such as CPU in a SMP (Symmetric Multiprocessor) node is also possible. The parmon-server runs on each monitored cluster node and provides system resource activities and utilization information. It was developed in C, using POSIX/Solaris threads. The parmon-client is a GUI based client, developed in Java, responsible for the interaction between the parmon-server and the users, for data-gathering in real-time and presenting information graphically for visualization.

SCMS (SMILE Cluster Management System) is an open-source management and monitoring tool for beowulf clusters. It provides a set of tools and subsystems such as: real-time monitoring system, alarm system services, performance logging, parallel Unix command, system management tools, and Web, VRML based, monitoring tool. Its monitoring architecture can be divided in three modules: the Control and Monitoring Agent (CMA), the Systems Management Agent (SMA), and the Resources Management Interface (RMI). The CMA runs on each node and collects system statistic continuously from a software layer called HAL (Hardware Abstraction Layer). This information is collected by the SMA module, which acts as a centralize resource manager and helps to keep track of system information for later retrieval. The upper level module, which consists of an API called RMI, is responsible to provide access to the system monitoring services. This API interface is available in C, TCL/TK, and Java.

These tools represent the current state of the art in the usual implementation of Linux clusters monitors. However, they are only able to monitor a pre-defined set of information selected from a hard-coded list and this information is acquired at the same moment in all specified nodes. Moreover, users are not able to easily expand the monitor with additional acquisition routines for specific system hardware for system events. The information acquisition routine is usually initiated by the host and relative to all of the selected information, and it is not possible to have different acquisition methods for different sets of information.

## 4. RVision's Design and Implementation

### 4.1 Motivation

The main motivation for the development of RVision was the inflexibility of the freely available cluster monitoring tools. RVision's open architecture supports generic information monitoring, which allows the utilization of user defined acquisition libraries in the monitored resources, as well as, communication to user developed monitoring clients that connect to the monitoring kernel over the Internet. In addition, several information acquisition and transport mechanisms are supported. This high level of configurability pays off especially in the monitoring of heterogeneous clusters and cluster of clusters. Different sets of information to be monitored may be defined with specific frequency acquisition. These sets are then associated to the monitored resources, being even possible to have multiple sets associated to the same resource.

## 4.2. Concepts and Features

During the RVision design some new monitoring concepts have been introduced to enhance the system functionality. These concepts are:

**Monitoring Sessions** The monitoring session is a self contained dedicated monitoring environment specified in a configuration file. These sessions have all the necessary information to start the monitoring, like a list of monitored resources, the information to be monitored in each resource, and the acquisition mechanism for each data. After a session is defined, monitoring may be started and stopped. Online reconfiguration is also possible, with changes taking effect after a restart.

**Monitoring Library** The monitoring library consist of a collection of routines responsible for information acquisition. These libraries may be implemented by the user to add functionality to the monitor and are dynamic linked to the architecture at runtime. The main advantage of this approach is the separation between the monitoring mechanisms, like information transport and monitoring frequency, and the information acquisition, the latter being dependable on resource peculiarities.

The main features provided by RVision are:

- Multiple Monitoring Sessions: when a monitoring client connects with the monitor's kernel an individual monitoring session is created. Several sessions may be active at the same time (from different clients). These sessions may have several distinct monitoring configurations, called sections. This approach enables parallel monitoring sessions using different configurations which are described in a Monitoring Session Configuration file (MSC file). An example of the MSC file is presented below:

```
BEGINLIBDECL    # Monitoring Libraries used
# <index> <monlibname>
 0 librvision
ENDLIBDECL

BEGINMONSESSION ONLINE # ONLINE or POSTMORTEM
# Collection of sections

 BEGINSECTION
  NODES       # Nodes to be monitored
#  <node-name>
   node01
   node02
   node03
  TYPE        # Monitoring type
#  cyclic <period>         - period in seconds
#  client-request <group_id> - group id as an integer
#  alteration <period> <percentage> - period in
#          seconds and percentage as a real number
   cyclic 4.0
  INFO        # Information
#  <monlibname> : <infoname>
   librvision : cpuused
   librvision : memused
 ENDSECTION
ENDMONSESSION
```

Between the "`BEGINMONSESSION`" and "`ENDMONSESSION`" tokens, several concurrent monitoring sections can be defined. The information to be monitored is declared with the corresponding library name and its identifier (described below in the MLD file). Both names are used by the monitor kernel to identify the information when sending it to the monitoring client.

- Generic Information Monitoring: if a monitoring client wants to acquire information that is not implemented in RVision's default monitoring library (`librvision`), it may include a new library with the needed functionality to the system. This new monitoring library will be dynamically linked to RVision by demand. This feature allows the development of specific libraries for new hardware technologies (e.g. Myrinet [9] and SCI [11] network boards). The monitoring library needs to define a vector with `void` pointers named `functions` to enable the dynamic linking with the RVSpy, each function of this library needs to be declared in this vector. The monitoring library uses a monitoring library description file (MLD File) to identify the available monitoring information to the system and acts as the link between the information identifier and the correspondent function pointed by the `functions` vector. An example of a basic monitoring library header is presented below:

```
/* Monitoring Library Header */
int function_1();
double function_2();
char *function_3();

void *functions[] = { function_1,
                      function_2,
                      function_3 };
```

The presented monitoring library has the following description file (used type substitutions in the MLD showed in Table 1):

```
# Monitoring library description file - MLD
# <index> <type> <identifier>
# where:
#   <index> is the position of the function pointer
#                              in the functions vector
#   <type> the type returned (see Table 1)
#   <identifier> identifier to the function, usually
#                                   the information name
0  int     info_1      # provided by function_1
1  double  info_2      # provided by function_2
2  string  info_3      # provided by function_3
```

- Instant of Information Analysis: RVision supports online and post-mortem monitoring analysis. Online analysis corresponds to the visualization of the information being monitored during the monitoring session. Post-mortem analysis stores all information monitored in a file called post-mortem file (PM file), making the monitoring session less network intrusive, since it does not send the monitored information to the monitor kernels over the network. On the other hand, the monitoring client has access to the PM file only after the monitoring stops.

**Table 1. Type Substitutions in the MLD File**

| C Type | Number of Bytes | Type in MLD File |
|---|---|---|
| unsigned char | 1 | uchar |
| char | 1 | char |
| unsigned short | 2 | ushort |
| short | 2 | short |
| unsigned int | 4 | uint |
| int | 4 | int |
| unsigned long | 4 | ulong |
| long | 4 | long |
| float | 4 | float |
| double | 8 | double |
| char * | 255 | string |

- Monitoring Frequency: the monitoring frequency can be selected between client request, regular time interval, or percentual change in value. The former is client initiated, while the latter two are initiated in the resource side. In the client request mode, the monitoring client sends a request to the resources when the information is needed. Two messages (the request and the reply) are sent over the network. In the regular time interval mode, the resource side sends the needed information to the monitoring client periodically after the start of a monitoring session. Since the time control is done in the resource side, only one message to the monitoring client is needed. The percentual change in value mode works like the regular time interval mode, but it only sends the information when a value changes by more than a threshold specified in the session file (e.g., CPU usage changed more than 10%). Since the threshold control is done also in the resource side, only one message to the monitoring client is needed.

## 4.3. Design and Implementation

The RVision architecture is based in the classic master-slave model [10] utilized by most of the resource monitors found in the literature. This centralized approach was chosen because it is less complex then a distributed model and therefore more efficient in small and midsize clusters. It has been implemented on Linux/GNU using the C language. The architecture is composed of five modules presented in Figure 1. The implementation of RVision is Open Source and uses the GPL License.

### 4.3.1 RVCore

The RVCore, presented in Figure 2, is the monitoring kernel, which corresponds to the master of the architecture. It realizes the control of all monitoring sessions and sends the monitored information to the respective monitoring clients.

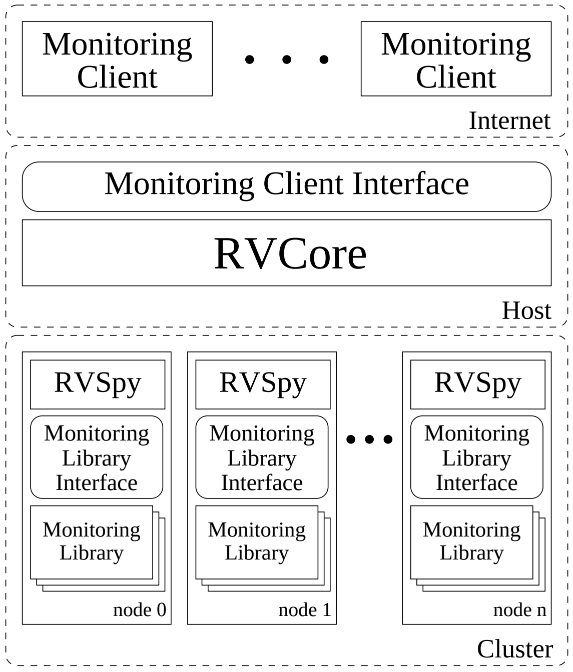


**Figure 1. RVision Architecture.**

The kernel is divided in RVCore Manager, Authentication Module, Monitoring Session Manager, RVSpy Group Manager, and Data Receiver Module. It is implemented using POSIX threads [12] and GNU/Linux sockets. Control messages between the RVCore and the monitoring clients are sent using the TCP protocol. Data messages can be sent with TCP or UDP. The transport mechanism is chosen by the monitoring client during execution time. All communication between the RVCore and the RVSpy's (resource side) utilizes the UDP protocol. The choice for communication using the UDP protocol is due to it's low overhead and because clusters are usually connected by LAN networks where package loss is minimal [13].

The RVCore resides in the host machine, waiting for connections. When a connection is established, the RVCore Manager checks the user login and password using the Authentication Module. It uses the *passwd* file or *shadow* depending on which authentication mechanism the RVision has been compiled with. After the authentication, if the user and password are valid, it creates the Monitoring Session Manager as another thread and passes to it the socket for client communication as an argument. If the password is not valid the connection is closed. The Monitoring Session Manager waits for monitoring requests (described in Section 4.3.4, which presents the monitoring client interface). If the request is for configuring the transmission of

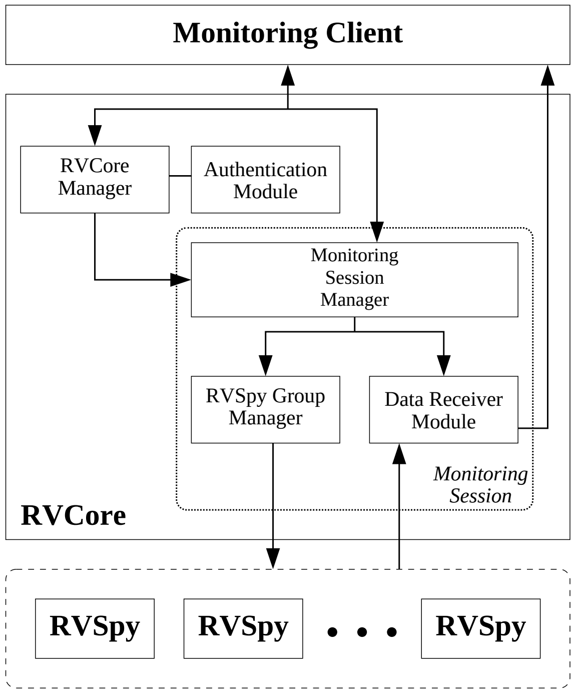


**Figure 2. RVCore**

the monitoring data to the client, it just receives and stores the values. When the request is for a session configuration, it parses the file indicate in the argument and accumulates the configuration. When the request is for a session start, it creates the RVSpy Group Manager and the Data Receiver Module. The RVSpy Group Manager access all RVSpy's for each node indicated in the MSC File, pass the configuration for each node and starts the monitoring. The Data Receiver Module waits for data and forwards it to the Monitoring Client, acting as a proxy. When the request is for a session stop, the RVSpy Group Manager send a request to stop the monitoring for each RVSpy and suspends the Data Receiver Module. If the request is a client request, it just forwards the request to the RVSpy's with the corresponding client request group. Finally, if the request is for session close, it frees all memory utilized, closes the connection, and destroys the monitoring session.

### 4.3.2 RVSpy

The RVSpy is the slave of the architecture and resides on each node of the cluster. It is responsible for data acquisition and transmission to the RVCore. It is implemented using POSIX threads and GNU/Linux sockets (UDP protocol) being divided in the RVSpy Manager, Data Collectors Manager, Dynamic Linker, and a group of Data Collectors.

When a packet arrives to one RVSpy, it creates the Data Collectors Manager, receives the monitoring configuration for the node, executes the dynamic linkage, and waits for the monitoring starting signal. When this signal arrives, it creates the Data Collectors, which are implemented as POSIX threads. Each Data Collector is responsible for a different monitoring frequency, and can monitor more than one event. They are implemented as a loop that catches the information and sends to the RVCore, when it is configured for online analysis, or stores it on a file, for post-mortem analysis. When the percentual change in value mode is used, it computes the threshold and sends the information only if the diference is higher than the threshold specified by the user. After that, the Data Collector can realize one of two operations: sleep for some defined time (regular time interval and percentual change in value modes) or wait for a signal to send more information (client request mode). If the Data Collectors Manager receives a stop monitoring request, it cleans up the used memory, and stops the Data Collectors.

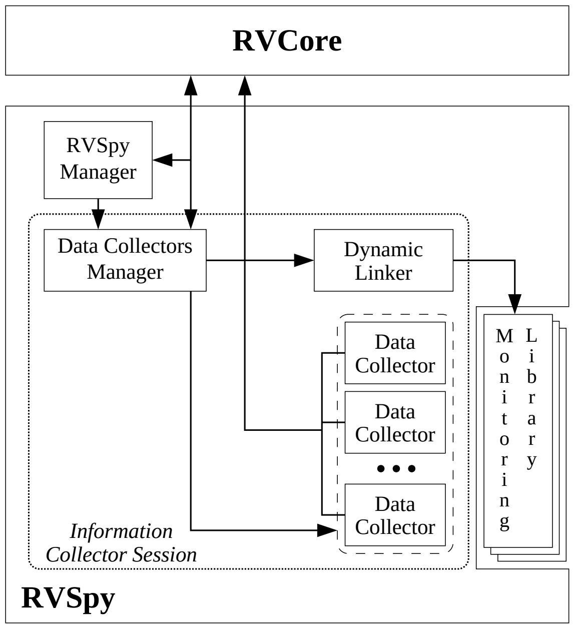


**Figure 3. RVSpy**

### 4.3.3 Monitoring Libraries

The Monitoring Library is responsible for the data acquisition in the cluster nodes. It is dynamic linked to the RVSpy at runtime so that several libraries may be available being only loaded if the functions they contain are referenced in a monitoring session. This open end enables the monitoring of additional information with no modifications in the RVision code.

The Monitoring Library consists of an usual Linux/GNU shared library using some definitions to enable dynamic linking and a description file to provide information about the compiled library to the RVCore. The RVision monitoring tool provides a basic monitoring library called `librvision`. It acquires system information from the `/proc` directory [5]. This library enables the monitoring of usual system information such as: CPU utilization, memory utilization, swap space and network utilization.

### 4.3.4 Monitoring Client Interface

The Monitoring Client Interface defines the functionalities provided by the RVCore describing function parameters and return values. These functions are sent to the RVCore using sockets. The functions provided by the monitoring client interface are presented in Table 2.

**Table 2. Monitoring Client Interface Functions**

| Commands | Parameters |
|---|---|
| RECVCONFIG | protocol, port |
| SESSIONCONFIG | MSC file |
| STARTMON | – |
| STOPMON | – |
| CLIENTREQ | group id |
| CLOSESESSION | – |

The *RECVCONFIG* command defines the protocol (TCP or UDP) and port utilized by the client for receiving the monitoring data. This information is sent to the RVCore as two integers, the first one indicates the protocol, while the second identifies the port. The *SESSIONCONFIG* command specifies the monitoring libraries used for monitoring, the type of monitoring (online or post-mortem), the monitoring frequency, and the nodes and information monitored. This information is defined in the MSC file, described in Section 4.2. The client sends the name of the MSC file as a string. The *STARTMON* and *STOPMON* commands start and stop the monitoring, and the *CLIENTREQ* command grabs the client request information defined in the MSC file. This command takes one parameter to indicate the client request group to be captured (i.e., different groups of information can be defined and captured using explicit client request). When the RVCore receives this value, it just sends to the monitoring client the information defined in this group. The *CLOSESESSION* command closes the monitoring session.

### 4.3.5 Monitoring Client

The monitoring client is responsible for data presentation and user interaction. It uses GNU/Linux sockets for communication with the RVCore. Users are able to implement their own monitoring clients building on RVCore functions accessed through the monitoring client interface (Section 4.3.4). The structure of a basic monitoring client is presented in Figure 4 and consists basically of two threads: one for management and the other one for data reception. Figure 5 displays a screenshot of the Monitoring Client developed for the CPAD main cluster (http://www.cpad.pucrs.br).

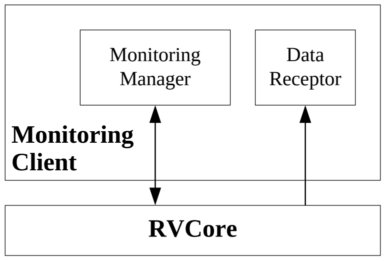


**Figure 4. Basic Monitoring Client Architecture**

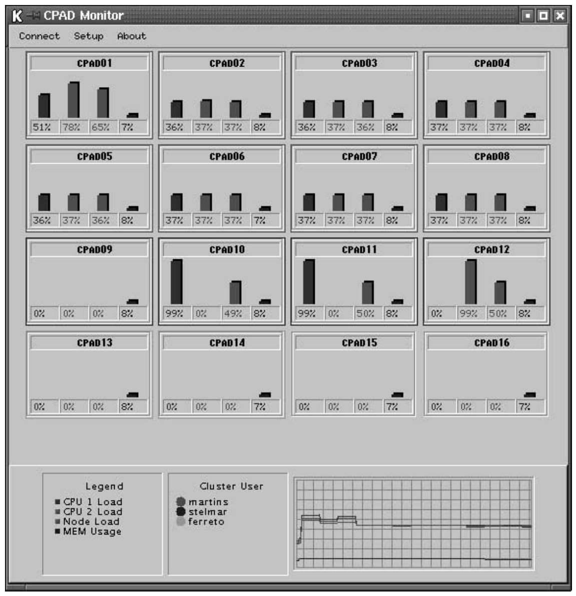


**Figure 5. CPAD Monitoring Client**

## 5. Performance Measurement

The main concern when a tool is used to monitor some resource is how much the readings are being affected by this tool. Being a parallel application itself, the monitoring tool, when active, is consuming cluster resources like node CPU and network bandwidth. This is called intrusion and should be minimized to guarantee that the monitored data is accurate and that the behavior of the other application running in the cluster are not considerably affected.

We measured RVision's intrusion defining a monitoring session with 12 values for all cluster nodes and requesting this data using online monitoring with a regular time interval. Different applications are executed in the cluster with and without monitoring and we compared the execution times. We varied the time interval from 5 seconds to 500 milliseconds to simulate a worst-case scenario. Data was acquired from the `/proc` file with the `librvision` monitoring library and the session configuration file used is presented below.

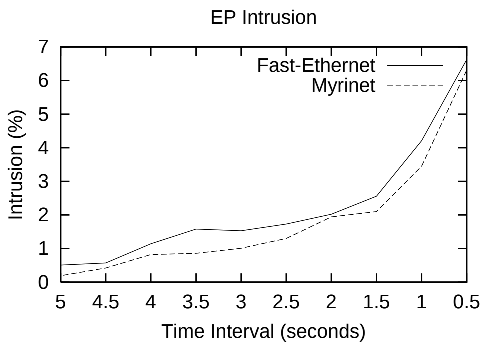
```
# Monitoring Session Configuration File
BEGINLIBDECL
 0 librvision
ENDLIBDECL

BEGINMONSESSION ONLINE
 BEGINSECTION
  NODES
   cpad01 cpad02 cpad03 cpad04
   cpad05 cpad06 cpad07 cpad08
   cpad09 cpad10 cpad11 cpad12
   cpad13 cpad14 cpad15 cpad16
  TYPE
#    cyclic <variable period of time, 0.5 - 5.0>
    cyclic 0.5
  INFO
   librvision : usercpu_perc
   librvision : nicecpu_perc
   librvision : systemcpu_perc
   librvision : idlecpu_perc
   librvision : cpuused_perc
   librvision : usedmem_perc
   librvision : freemem_perc
   librvision : sharedmem_perc
   librvision : buffersmem_perc
   librvision : cachedmem_perc
   librvision : usedswap_perc
   librvision : freeswap_perc
 ENDSECTION
ENDMONSESSION
```

To evaluate RVision under different workloads four MPI applications with different characteristics were used for performance measurements: Povray [2], IS and EP from the NAS parallel benchmarks [3], and Linpack [8]. Povray is a ray tracer based in the master-slave model. The size of the test image was 800x600 pixels. IS sorts a vector of integers using data parallel computations with synchronization points. In our test we used IS with class C. EP is a random number generator based in the master-slave model. In our test we used EP with class B. Finally, Linpack is one the most used benchmarks to evaluate cluster performance implementing an LU factorization on a matrix using parallel data computation with synchronization points. In our test we used the values 5000, 32, 4 and 8, for the variables N, NB, P and Q respectively. IS is network bound while EP is CPU bound. Linpack and Povray have a good balance between calculation and communication.

The tests were conducted on the CPAD main cluster, which has 16 dual-processor (Pentium III-550MHz) nodes, each node with 256 MBytes of main memory. It utilizes two types of interconnection networks, Myrinet and Fast-Ethernet both switched. Since, RVision uses sockets, the measurements were made with the applications running on Myrinet and on Fast-Ethernet to see how this affected intrusion, while we ran RVision only on Fast-Ethernet.

Table 3 presents the intrusion results for all test cases. In each case the test application was executed 10 times for each time interval with monitoring turned on. With the time interval of 1 second intrusion was around 10% for all applications. This is a very good result considering the large configuration file used and that in some cases application and monitor use the same network for communication (Fast-Ethernet). We observe also that intrusion is higher on Linpack and IS applications. This is expected because these applications have higher network usage. To have some worst-case results we reduced the time interval to 0.5 seconds. Even in this case intrusion results where lower than 10% except for Linpack and IS.



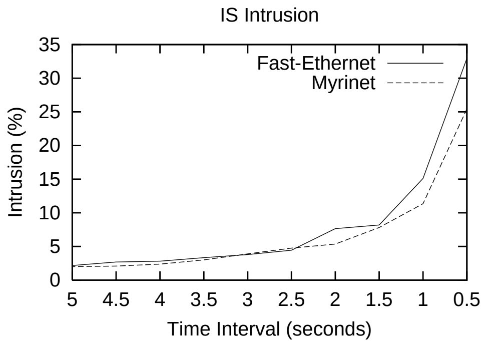


**Figure 6. IS and EP Intrusivity**

Figure 6 presents intrusion results for IS and EP applications in relation to the time interval. These applications have been specially analyzed because of their characteristics as network bound (IS) and CPU bound (EP). From the results we concluded that in our experiment network interference was much greater than processor interference, with IS generating 3 times more intrusion than EP.

**Table 3. Applications Intrusion Measurements**

| | Povray | | IS | | EP | | Linpack | |
|---|---|---|---|---|---|---|---|---|
| Time Interval (sec) | Myrinet | Fast-Ethernet | Myrinet | Fast-Ethernet | Myrinet | Fast-Ethernet | Myrinet | Fast-Ethernet |
| 5.0 | 0.90% | 0.90% | 1.99% | 2.17% | 0.19% | 0.51% | 0.87% | 2.17% |
| 4.5 | 1.07% | 1.14% | 2.08% | 2.68% | 0.41% | 0.57% | 0.92% | 2.68% |
| 4.0 | 1.08% | 1.35% | 2.37% | 2.80% | 0.81% | 1.14% | 1.10% | 3.45% |
| 3.5 | 1.27% | 1.58% | 2.98% | 3.33% | 0.85% | 1.58% | 1.28% | 3.79% |
| 3.0 | 1.73% | 1.99% | 3.89% | 3.79% | 1.00% | 1.52% | 1.45% | 4.05% |
| 2.5 | 1.85% | 2.17% | 4.73% | 4.42% | 1.29% | 1.73% | 1.92% | 4.42% |
| 2.0 | 2.47% | 2.69% | 5.34% | 7.64% | 1.94% | 2.02% | 2.64% | 6.41% |
| 1.5 | 2.73% | 3.33% | 7.81% | 8.18% | 2.10% | 2.56% | 3.37% | 9.25% |
| 1.0 | 3.52% | 4.22% | 11.34% | 15.10% | 3.45% | 4.20% | 5.36% | 13.74% |
| 0.5 | 7.64% | 8.18% | 25.43% | 32.93% | 6.30% | 6.61% | 12.06% | 29.35% |

## 6. Conclusion and Future Work

Due to its favorable cost-performance ratio, cluster computing is becoming more utilized in several application areas. This higher utilization increases the need and importance of cluster monitoring for system tuning and application development. In this paper we presented RVision, an open architecture high configurable tool for cluster monitoring, which provides flexibility in selection of events to be monitored and allow users to expand the monitoring capabilities with self-defined procedures for the monitoring of specific system hardware or system events. An additional feature of RVision is its flexibility in allowing the monitoring of distinct clusters, cluster of clusters, and heterogeneous clusters. In our experiments using RVision the measured intrusion was around 10% with the default monitoring library (`/proc`) for different kinds of applications.

RVision has being used at our research center (CPAD) in production mode for the last six months. New development is underway, some of them are: including a mechanism for dynamic configuration during the monitoring session, defining a more generic representation of the returned values by the monitoring library functions, enabling an extended set of data types, such as “structs”, suporting to PAM (Plugable Authentication Module) in the options for system authentication, and changing the format of all configuration and output files to the XML standard [6]. There are also monitoring clients and monitoring libraries under development, such as a monitoring client applet (for Webpages), a monitoring library using SNMP library and a monitoring library for Myrinet technology. More information about RVision and the package for download are available in *http://www.cpad.pucrs.br/rvision/index.html*.

## References

[1] bwatch 1.0.3. http://www.sci.usq.edu.au/staff/jacek/bWatch, 2001.

[2] POV-Ray - the persistence of vision raytracer. http://www.povray.org, 2001.

[3] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS parallel benchmarks 2.0 report NAS-95-020. Technical report, NASA Ames Research Center, December 1995.

[4] M. Baker. Cluster computing white paper, 2000.

[5] T. Bowden, B. Bauer, and J. Nerin. The /proc filesystem. Linux Kernel Documentation, 2000.

[6] T. Bray, J. Paoli, and C. Sperberg-McQuee. Extensible markup language (XML) 1.0. http://www.w3.org/TR/1998/REC-xml-19980210.html. W3C Recommendation - 10-Feb-1998.

[7] R. Buyya. PARMON: a portable and scalable monitoring system for clusters. *SP&E*, 30(7):723–739, 2000.

[8] J. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. In W. J. Karplus, editor, *Multiprocessors and Array Processors. Proceedings of the Third Conference*, San Diego, CA, USA, 1987. SCS.

[9] C. L. S. et al. Myrinet - a gigabit-per-second local-area network. *IEEE Micro*, vol. 15(1), 1995.

[10] K. Hwang and X. Zhiwei. *Scalable Parallel Computing: technology, architecture, programming*. Boston: Wcb-Mc Graw-Hill, 1998.

[11] IEEE standart 1596-1992, New York. *IEEE: IEEE Standart for Scalable Coherent Interface (SCI)*, 1993.

[12] B. Lewis and D. Berg. *Multithreaded Programming with Pthreads*. Sun Microsystems Press, 1998.

[13] A. Tanembaum. *Computer Networks*. Prentice-Hall, 3th edition, 1996.

[14] P. Uthayopas, J. Maneesilp, and P. Ingongnam. SCMS: An integrated cluster management tool for beowulf cluster system. In *PDPTA 2000*, Las Vegas, Nevada, USA, 2000.